Fiona Wolfe - 15316812

Measuring Engineering – A Report

Deliver a report that considers the ways in which the software engineering process can be measured and assessed in terms of measurable data, an overview of the computational platforms available to perform this work, the algorithmic approaches available and the ethics concerns surrounding this kind of analysis.

Introduction

In this report, I will discuss the above four sections. Measurable data will be looked at in terms of the different types of data available and how much of it is important to measure. I will then look at the computational platforms available, including some case studies of companies and research projects that have measured software engineering and have had results which I believe to be interesting. Next, the algorithms that are used to collect and analyse the data discussed above will be looked at. I will describe some of these in detail and will make a conclusion as to whether these algorithms can appropriately and accurately assess software engineering in the way that is required today. Finally, I will talk about some of the ethics concerns regarding the measurement of data in the workplace and how these concerns may be alleviated.

Measurable Data

Within the software engineering process, there is a huge amount of data that can be measured. So much, in fact, that a specialist looking at this information must decide which of it is useful and which is not. To do this, we must first classify the data.

The data produced by a software engineer can be grouped in many different ways, but the way which I have chosen to do it is to split it into *code* and *non-code*. Data that falls under the *code* category may be things like the number of lines of code written by a developer in any given day, the number of times s/he committed the code, the technical debt, to name but a few. *Non-code* data would be things like the number of emails the developer sent regarding the project, the number of meetings attended, the amount of times s/he asked a peer to review their work. Throughout this report, I will refer to *code* and *non-code* data as two distinct categories of data that need to be measured accordingly.

With regards to collecting *code* data, "Searching under the Streetlight for Useful Software Analytics", by Philip M. Johnson, University of Hawaii at Manoa discusses his team's research into some of the ways of gathering data from software engineers.

In measuring *code* data, the team from the University of Hawaii discussed a variety of data collected under various different methods. The Personal Software Process (PSP) had several versions, but in essence it used forms to collect information from developers, which included a project plan summary, a defect-recording log, a design checklist, and a code checklist amongst others. The forms were not automated, a feature which was justified by the creator Watts Humphrey when he said, "It would be nice to have a tool to automatically

gather the PSP data. Because judgement is involved in the most personal process data, no such tool exists or is likely in the near future." The manual nature of PSP made it flexible, but ultimately resulted in data quality problems.

Next came the Leap toolkit, which tried to address these data quality problems by automating the process. A lot of data was still manually entered, but Leap provided an analysis which PSP was unable to. It was lightweight, portable and provided reasonably in-depth conclusions. However, through the introduction of automation, Leap made certain aspects of software development very difficult to analyse. The flexibility of PSP was lost as if a developer wanted to check for an issue, they would have to design and implement a whole new Leap toolkit rather than just filling out a form as was done in PSP.

On the opposite end of the spectrum, there was Hackystat. The Hackystat technology had four main design features – client and server-side data collection, unobtrusive data collection, fine-grained data collection and personal and group-based development. It collected information automatically, rather than the developer entering it manually, and had the ability to update every second. However, it analysed much of the same information as PSP – time spent on the project, size of the project, commits – as well as some more advanced data, such as code coverage and complexity.

Collecting this *code* data appears to be relatively straightforward. We can rely on the programmer to tell it to us, or we can install a program on their computer which will do that for us. In practice, with the current state of technology, the second option is a far more accurate, advanced and still inexpensive way of collecting the data, although there are some ethics issues relating to this which will be explored later on.

However, it would be wrong to proceed with only *code* data, analyse it and form some sort of conclusion. A developer who spends lots of time on their project and commits regularly, but doesn't respond to emails, communicate with colleagues or take advice, will appear to be a good software engineer under this regime. This is obviously incorrect. The *non-code* data, although more difficult to measure and quantify, is just as important.

The paper 'Network Effects on Worker Productivity' by Matthew J. Lindquist and Jan Sauermann explores this topic. The group looked at how co-worker productivity impacts on worker productivity via network effects, by using a dataset from a call centre for a period of two years. This dataset had information about the performance of each worker, their characteristics, team affiliation and when they clocked in and clocked out of work.

Through running two regression experiments, it was found that co-worker productivity did have a significant effect. In fact, a 10% increase in productivity of a co-worker's network led to a 1.7% increase in the worker's own productivity. Furthermore, the addition of one trained co-worker to a worker's network increased this worker's productivity by 0.7%. These findings show how the impact of *non-code* data to any analysis of productivity – the issue is now, what is the best way to find this data?

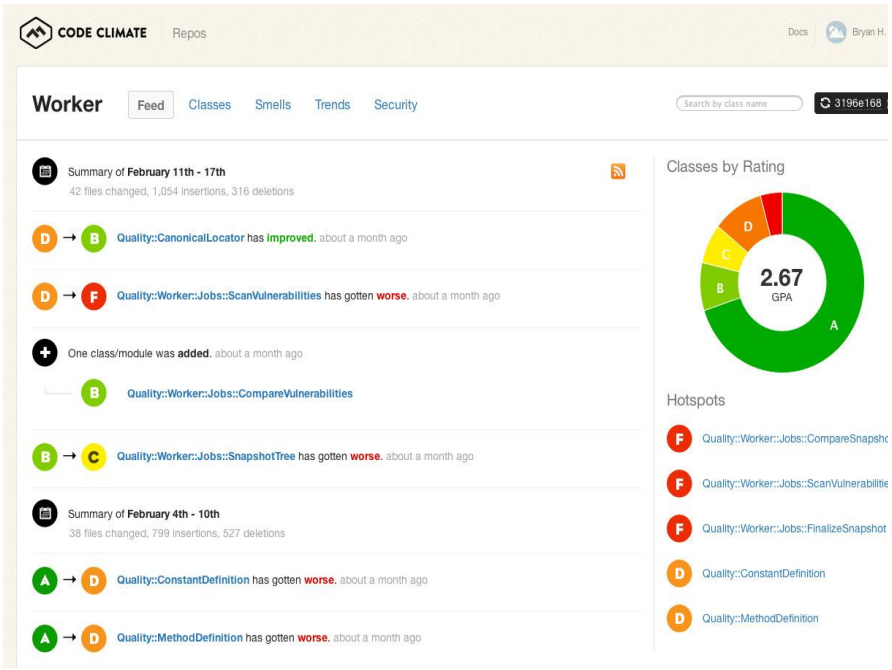
In my view, measuring communication levels and networks is more difficult than measuring an engineer's code, but certainly not impossible. For example, a simple attendance list at meetings could be used to analyse individual developers and gauge their performance. A program could monitor the outgoing emails from an individual's computer, and the number of minutes (or hours) spent per day on Facebook or gaming websites. This is all important *non-code* data that would be needed to complete a full analysis on a software engineer, no matter what type.

The measuring of the soft data is more of an issue. How do we measure how social an engineer is? How well s/he can take criticism and advice and therefore improve their projects? This data exists and is important, but is not something that can be determined by a program in a computer. One way of collecting this would be a manual form, either personal or a peer review analysis, to grade an individual on a scale on various different aspects of their personality that may impact their work.

To conclude this section, it is safe to say that there is a lot of data to be processed, and our job is to find which of it is useful and which is not. In my opinion, any analysis of any aspect of a worker, be it their productivity, success rate or otherwise, should include *code* and *non-code* data. At the moment, the majority of the studies of software engineers seem to be focused on the code that they are writing on their computer. This appears to me to be a narrow-minded approach, as there are so many other factors which impact upon a worker's output.

Computational Platforms

Once it is decided what data to collect, the next task is to determine where and how to compute it. There are many different companies around that offer data analytics as a service. The website clutch.co listed 209 companies in 'Best Big Data Analytics Companies – 2017 Reviews'. These companies came from 22 distinct industries, including education, financial services, health care, legal and media.



Example Code Climate output

With relation to data analytics companies that specialise in software engineering, the pool is smaller. Automated code review tools are becoming increasing popular. These are programs which take in code and assess it on a number of different levels, then make recommendations about what can be improved. An example of a firm that does this is Code Climate. This company offers a service which can be used in conjunction with Github, which will show a user their code coverage, technical debt and a progress report, amongst other. This is a user friendly, accessible way to view the kind of *code* data which was spoken about in the first section.

> *"Used by over 100,000 projects, and analyzing over 2 billion lines of code daily, Code Climate incorporates fully-configurable test coverage and maintainability data throughout the development workflow, making quality improvement explicit, continous, and ubiquitous."* – **codeclimate.com**

Code Climate has many competitors providing similar analytics. It has the advantage of supporting many different languages, technologies and frameworks, as well as being known for its reliability, stability and test coverage. However, it is expensive when compared with other products such as Codebeat, which is dynamically growing, also has wide language support and metrics customisation. Codacy is another product which has similar features. These services are in high demand from software engineers as this section of software analytics is arguably the most applicable and useful to developers in their day to day work.

Software analytics is not limited to these automated code review tools however. In 2012, a group from North Carolina State University developed “StackMine – Performance Debugging in the Large via Mining Millions of Stack Traces”. They had found debugging before the release of software to be ineffective and insufficient at ensuring satisfactory software performance. The team began by collecting runtime traces from millions of users, analysing this data and finding the program execution patterns which was the root cause of the worst software performance problems. The solution then involved callstack pattern mining and clustering, which allowed them to precisely measure the impact of each pattern and to fix the issues quickly and precisely.

Their research, when conducted on Windows 7, cut the time spend on performance debugging from 20+ days to 18 hours, achieved 58.26% of performance bottleneck coverage and only required on average 6.4% of the trace streams requited by other techniques. This study clearly shows how data analytics can be used in the real world to improve code.

Another example of the usage of data is XIAO, developed by a team at Microsoft Research Asia, to complete a code clone analysis. Code cloning is essentially reusing lines of code in a program without modification, through copy and paste or another means. The team identified several instances where a developer may want to identify code cloning – to fix a bug in a fragment of code with duplicated copies in the program, and to reduce cloning in order to maximise memory space being examples.

XIAO was developed with a number of key requirements based on the above scenarios. These were the ability to identify near-miss code clones, scalability, ease of use and ease of deployment. The first version was released within Microsoft, and the team collected data from engineers using the software to improve the product going forward. For example, one specific engineer used XIAO to identify potential bugs caused by inconsistent code clones, and discovered that 10% of the clone groups within the program had potential code defects caused by code cloning. This was the kind of data used by the team to develop the product.

These projects are just examples of the kind of work that is undertaken by data analytics companies. They collect data from developers about their work, and use this to develop a product that will solve a specific problem. Often, this data is of the *code* type and is volunteered by the engineers. In my opinion, the work that these platforms complete is highly technical and useful. It is a growing field and there are a huge number of firms and projects which are exploring new data and coming to new conclusions. Outsourcing data analysis to an outside firm seems to be becoming increasingly popular, and the benefits are clear, as the data analysis completed is hugely comprehensive. The algorithms used within the data analysis techniques are often complicated, and will be explored in the next section, with some examples.

Algorithmic Approaches

Before getting into the algorithms themselves, it is important to note that there are a huge number of analytic techniques available, some more relevant than others. A particular area which I will focus my attention on is predictive analysis, which looks at statistical methods which analyse data in order to make predictions about patterns that may occur in the future. Many of these algorithms can be classified into two distinct groups – regression techniques and machine learning techniques. Under machine learning falls *supervised* and *unsupervised* machine learning algorithms. Supervised learning occurs when there is an input and an output variable, and an algorithm is used to map the input to the output (i.e. what happens to the input to product the output).

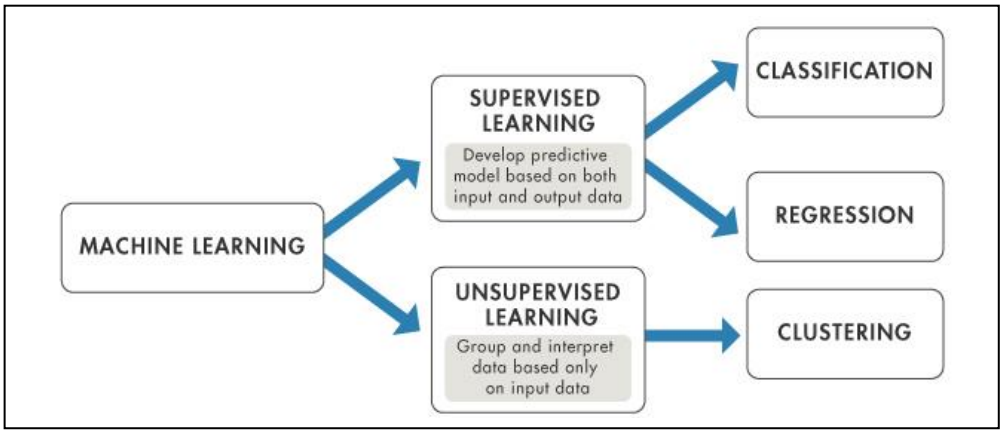
Y = f(X)

- Mapping function for an input variable (**X**) and an output variable (**Y**).

The idea of supervised learning is that, after some trial and error, the mapping function will be so well approximated to reality that when there are new inputs, the appropriate outputs can be predicted. It gets the name 'supervised' from the fact that the learning process is being taught. Every prediction is corrected and used to increase the accuracy of the next prediction. Learning ends when the algorithm's performance is deemed acceptable.

Unsupervised learning, on the other hand, is the situation where there is an input variable but no corresponding output. The task is to try and find out more information about the data by modelling the underlying structure. It is 'unsupervised' as there is no teaching process and no correct or incorrect results. The data must be analysed by an appropriate algorithm to discover some fundamental structure, but there are no rules on the form which the structure must take.

Within supervised and unsupervised learning, there are some categories. Supervised learning problems could be *classification* problems, where the output data is a categorical variable, or *regression* problems, where the output is a numerical value. Unsupervised learning problems include *clustering*, where the input data is placed into distinct groups based on similarities and differences, and *association*, where computations are performed to discover rules that define substantial amounts of the data (e.g. if something is X, it is also Y).



An example of a supervised learning algorithm is linear regression. Although it is more commonly known as a statistical technique, it is very applicable in applied machine learning as well. It involves numeric input values (**X**) and corresponding numeric output values (**Y**) combined together in a linear equation. This equation assigns a coefficient to each input value as a scale factor, and another coefficient as the y-axis intercept. For a simple regression problem with a single **X** and **Y**, the model's form is: **Y = B0 + B1*x**, where B0 and B1 are the relevant coefficients. Using a linear regression model within machine learning requires providing an estimate of the coefficients which are relevant to the input data, to provide an output. There are a number of different techniques used to estimate these coefficients and prepare the model, a few of which will be discussed now.

Simple Linear Regression is the first technique, which is used when there is a single input. Statistical properties from the data such as mean, standard deviation, covariance and correlation can be used to make a prediction of the value of the coefficients. This is the simplest technique, but not widely used as it is usually not advanced enough for the data.

When there is more than one input value, Ordinary Least Squares can be used. This approach calculates the distance from each data point to the regression line, squares all these distances, and then sums them together. This sum is known as the sum of squared errors, and Ordinary Least Squares aims to minimise this. The coefficients at the minimal value are the ones which should be used. Other methods include Gradient Descent and Regularisation, but these will not be discussed in detail here.

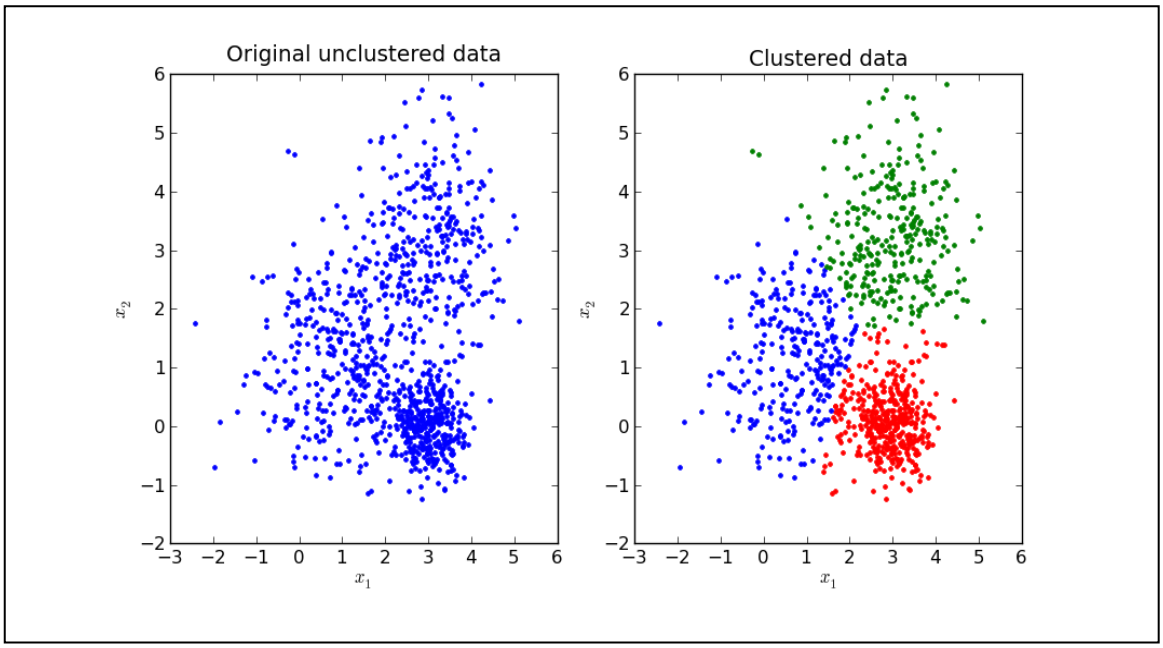
Once the data is prepared appropriately to be used in a linear regression analysis, it can be used to make predictions. This model can be very useful in the field of software engineering, as it can be applied to almost any problem – measuring productivity of developers in the future or estimating the time it will take to write a particular program being just two examples.

Now I will move to my chosen example of an unsupervised learning algorithm, which is K means clustering. Clustering algorithms are very useful in data analysis as anything –

developers, programs, teams – can be divided into specific clusters based on their characteristics which can help the analyst to understand their behaviour better. The K means algorithm divides data into clusters based on the local maxima in each iteration.

The first step in K means is to specify the number of clusters **k** (e.g. k = 2) and randomly assign each data point to a cluster. Once the clusters are created, the central data point in each (known as the cluster centroid) must be computed. Then, re-assign each data point into a cluster based on which cluster centroid it is closest to. Re-calculate the cluster centroids for the new clusters and repeat until no improvements are possible – i.e. the cluster centroids do not change between each iteration.

By following these steps, we will successfully have assigned each data point into a distinct cluster, an example of which can be seen in the graph below.



Of course, these machine learning algorithms are just a few examples of many that can be used to undertake an analysis on software engineering. In fact, the best way to proceed would be to use a variety of different algorithms in order to get a more measured result. In my opinion, the predictive algorithms available are strong, tried and tested and will give accurate results. However, they have a limited use in that they are most suited to measure *code* data. Measuring *non-code* data such as social interactions cannot be done using the algorithms discussed above, or any machine learning algorithm as it is so difficult to quantify. In my view, in order to provide a more accurate analysis measures such as human judgement must be used in conjunction with statistical algorithms. Although more difficult to quantify, the human component is necessary to capture all of the available data.

Ethics Concerns

Now that the measurable data, computational platforms and algorithmic approaches have been explored, the final issue to look at is ethics.

Returning to the article "Searching under the Streetlight for Useful Software Analytics" which was discussed earlier, a number of crucial ethics concerns were raised. With regards to the Hackystat technology, developers who were using the package raised a number of issues that they had. One of Hackystat's main features was unobtrusive data collection, which some programmers flagged as a bug. They didn't want to install a program that would collect data about them without informing them of it. Secondly, the fine-grained, comprehensive data collection created some disruption within development groups. It was coined "Hacky-stalk" due to the details it provided on each developer. Lastly, developers were unhappy with their data being shared with management, despite management promising to use it appropriately.

This issue was explored by Robert D. Austin in his book, "Measuring and Managing Performance in Organizations". It was written in 1996, so parts of it are not applicable to today's technology centred workplaces, but some of the core ideas still stand. Austin's main point is that measuring employee performance can have a counterproductive effect and stop a company from achieving its desired results. Incentives for employees can distort the balance of their work and impact upon quality. For example, if there is a reward for the developer who writes the most lines of code in one day, engineers will be writing a lot more code with diminishing value. Austin recommends that tapping into the internal motivation of the worker is a better way to increase productivity, rather than measuring specific target goals.

Austin makes some interesting points, but in my opinion his view is outdated. Measuring performance has become commonplace and now that the process is so automated, it is easy for companies to do and the cost of measurement is not such an issue. The more pressing problem faced today is not measurement itself, but the accuracy of measurement, the degree to which people are measured and what companies are doing with the data.

There are a number of laws in place which limit what companies are allowed to measure. In May 2018, the current European Data Protection Act (DPA) will be replaced by the EU's General Data Protection Regulation (GDPR). The DPA was first put in place in the 1990s, when data collection was only done by large companies and large-scale data analysis was practically unheard of. It is clear from what we have seen so far that this is not the case anymore. The GDPR updates the DPA in a number of areas.

First of all, companies must now keep a record of how and when a person gives consent to store and use their personal data. Consent means that the individual must actively agree to their data being used, and it can be withdrawn at any time. Withdrawing consent means that details must be permanently erased. This means that companies must know exactly what data they hold, how they are going to use it, where it is located and how to

remove it completely. This transparency means that it is less likely data will be hacked or used for the wrong purposes.

Further changes include increased territorial scope, which means that the GDPR applies to all companies processing data of individuals who live within the European Union, regardless of where the company is based. This data sovereignty regulation has an implication on global cloud based services, as being able to access data "anywhere, anytime" may be a breach of confidentiality agreements. Also, under the new regulations organisations in breach of the GDPR can be fined the greater of 4% of annual global turnover, or €20 million.

These watertight laws help to remove the stigma around data collection and make workers feel more at ease. The panel debate 'Sizing up the Workplace – the role of measurement and data in the workplace experience" by Interserve (21 February 2017) has some very good points on how to use data to help workers in their day to day life. For example, it was suggested to analyse a working day as a series of 'journeys', and to make these journeys as diverse and interesting as possible. For example, an engineer going to several meetings a day that are all scheduled in different meeting rooms will meet different people on his/her way each time, and have new social interactions. Data can be used in two ways – to help management to track performance and increase productivity, but also to help workers to plan their time more effectively and increase their efficiency.

In my opinion, data collection and measurement are only going to become more prominent in working environments going forward. In the arena of software engineering, measuring engineering means more effective programs and increased innovation. The next step is to get workers on board with it and to use data within their terms. Earlier on, it was mentioned that people felt measurement was 'stalking' and they weren't comfortable with their information being shared. I think that these ethical issues can be overcome by complete transparency, in line with the new laws, so that people know exactly what parts of their work are being assessed and they can easily raise any concerns with management. Anonymity is also a huge factor as workers are more likely to agree to being assessed if their name is not attached to their data. Finally, if the data is also used to help workers and make their lives easier, it will help to alleviate concerns and break down the stigma that data collection is all bad. It is paramount for companies to ensure that information is used accurately and for the right purposes. I would suggest that the GDPR is a step in the right direction, but further laws should be passed to guarantee fidelity of measurement.

Conclusion

In this report, I have discussed four aspects of measuring engineering – what data to collect, where to compute the data, the algorithms used to do this and the ethical concerns surrounding measurement. With regards to data collection, different types of data were discussed, and it was decided that a mix of binary, numerical and categorical data from *code* and *non-code* categories was needed in order to get a true reflection of anything that was being assessed. A number of companies and research projects that undertook engineering measurement were examined in the next section and it was thought that the kind of work being done, if utilised properly, could be ground-breaking in terms of increasing productivity and innovation in the workplace. A selection of the algorithms used for data collection and analysis were looked at next and comprehensive as they are, I felt that they need not allow enough for *non-code* data and somewhat underestimated its importance. Finally, some ethical concerns surrounding measurement of data were addressed, and I concluded that it is fundamental for measurement to be accurate, legal and transparent for it to continue to grow in the future.

Bibliography

1. http://www.citeulike.org/group/3370/article/12458067

2.https://www.researchgate.net/profile/Jan_Sauermann2/publication/285356496_Network_Effects_on_Worker_Productivity/links/565d91c508ae4988a7bc7397.pdf

3. http://sei.pku.edu.cn/conference/ishcs/2013/StackMine-han.pdf

4. https://www.microsoft.com/en-us/research/wp-content/uploads/2011/05/Code_clone_detection_experience_at_Microsoft.pdf

https://codeclimate.com/about

5. https://www.itproportal.com/features/comparison-of-automated-code-review-tools-codebeat-codacy-code-climate-and-scrutinizer/

6. https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/

7. https://machinelearningmastery.com/linear-regression-for-machine-learning/

8. https://www.analyticsvidhya.com/blog/2016/11/an-introduction-to-clustering-and-different-methods-of-clustering/

9. http://www.telegraph.co.uk/connect/small-business/business-networks/bt/data-protection-laws-changing/

10. https://www.forbes.com/sites/forbestechcouncil/2017/04/11/is-data-sovereignty-a-barrier-to-cloud-adoption/#20ec3f041c82