UNIVERSITY OF THE WEST of SCOTLAND
UWS

BSc (Hons) COMPUTING HONOURS PROJECT

Machine Learning in Cyber Threat Intelligence

Michelle Pantelouris

B00389651

1st April 2022

Supervisor: Sean Sturley

Moderator: Graham Parsonage

Declaration

This dissertation is submitted in partial fulfilment of the requirements for the degree of [Programme of Study] (Honours) in the University of the West of Scotland.

I declare that this dissertation embodies the results of my own work and that it has been composed by myself. Following normal academic conventions, I have made due acknowledgement to the work of others.

Name: Michelle Pantelouris

(In Capitals)

Signature: mpantelouris

Date: 01/04/2022

Contents

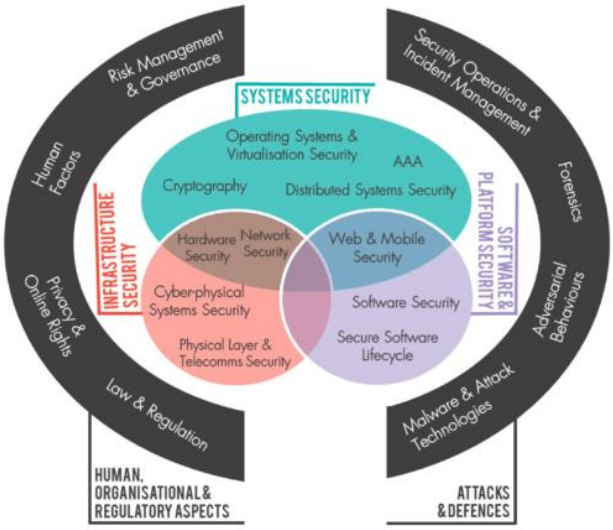
Introduction

In this report Cybok will be discussed as it was where I got my idea for this project. The history and advantages and disadvantages of python, machine learning and threat intelligence will also be discussed. After this is done the implementation stage of this project will be shown and an explanation of each step taken will be written with descriptions of the sections that needs to be described in more detail. The results of this will be analysed and given at the end of the implementation stage. A self-appraisal will be written, discussing the problems that were in this project. A research hypothesis will be written at the end of this report.

Cybok

The idea for this project came from Cybok. The Cyber Security Body of Knowledge is a collective proposal that was mobilised in 2017. Their aspiration was to "*codify the foundational and generally recognized knowledge on Cyber Security*." Version one was first released in 2019 and was became more public in January 2020. It has not captured the attention that it deserves through the industry.

CyBOK has over eight hundred pages that are split into nineteen top level Knowledge Areas (KAs). This is then split into five overarching categories as shown in the picture below. (Macrae, A. (2020)).



(Macrae, A. (2020)).

A lot of this is familiar for quite a few security professionals. A few of them have even questioned if it is not simply "reinventing the wheel" which has established a 'Common Body of Knowledge' which is widely recognised for its Certified Information Systems Security Professional accreditation. The categories for CISSP CBK are:

- *'Security and Risk Management (including Legal & Regulatory, Personnel Security, Threat Modelling)*
- *Asset Security (including Data Management, Privacy)*

- *Security Architecture and Engineering (including Security Models, Cryptography, Physical Site)*
- *Communication and Network Security*
- *Identity and Access Management (including IAM, IDaaS)*
- *Security Assessment and Testing*
- *Security Operations (including Incident Response)*
- *Software Development Security (including Malware)'* (Macrae, A. (2020)).

Origins and Definitions

It originally originated in the early 1990s. This was before the term 'Cyber' was used for IT related security matters. CBK was originally known as 'Common Body of Knowledge for Information Security'.

CyBOK starts off by offering a distinctive definition for 'Information Security' and 'Cyber Security' , which presents the former as a contributor to the latter. There is an overlap of knowledge and topics that are unavoidable, and they are across both taxonomies, the same as with their actual practices in the real world.

The definition that CyBOK uses as their introduction is :

"Cyber security refers to the protection of information systems (hardware, software and associated infrastructure), the data on them, and the services they provide, from unauthorised access, harm, or misuse. This includes harm caused intentionally by the operator of the system, or accidentally, as a result of failing to follow security procedures."

This can apply just as much to a lot of CISSP CBK. When the subjective lines are blurred further, this promotes CISSP as being 'the world's premier cyber security certification'.

A differentiator that is more useful and less semantic that is considered more, is that the CISSP CBK is a curriculum for the certification. At times it is described as being 'an inch deep and a mile long', but realistically it is more 'a mile wide and a foot deep in certain places.

Through a structured framework CyBOK will seek to map established knowledge sets. It can then be used for the informing and underpinning of educational and professional training in the cyber security sector.

It is acknowledged by the opening of the Law and Regulation category by denying itself to be a starting point instead of an ending point. It can also apply throughout CyBOK.

However, it is not just a dry reference guide to other works. The narratives of the expository that accompany each knowledge areas are original, insightful, and highly readable. It is important not to underestimate the quality of expertise drawn upon to create the diverse 'Knowledge Areas' each in their own right and to then bring them together into one coherent publication.

Furthermore, it will position itself as a vector agnostic, academically independent and while it is being sponsored by the UKs National Cyber Security Programme, it is a transnational effort rather than a regional or national focus. (Macrae, A. (2020)).

Knowledge Areas

The Cyber Security Body of Knowledge seeks to collect inputs from academia and industry. It comes across as being more at home within a classroom or a laboratory more than the operational, business driven frontline. because of its use of functional equations and theoretical models that are throughout the text.

Cryptography

This approach as with CISSP CBK is both appropriate and fairly inevitable for some areas. Cryptography is an important mathematical rooted are of subject. KA within cryptography will warrant a scholarship approach that is suitable to curation and the prefatory descriptions of primary concepts that are cyber security related.

Malware

The malware KA will descriptively dissect characteristics and tactics of various malware families while it is discussing analysis techniques that are used to understand them. This is done with its lab eye view of the subject matter. It then clearly includes concise explanations of the common anti-analysis and evasion techniques for example, packing which compresses or encrypts pieces of the code. The concepts are base concepts but are sometimes more sales focused with industry publications on malware.

The main technical considerations will then be complemented and contextually that is framed with a short introduction of the Underground Eco-System driving the malware lifecycle by itself. Cross-cutting concepts such as underground economics, monetization and black-market operating models are all discussed elsewhere, such as in the KA for 'Adversarial Behaviour' that follows.

Forensics

With forensics KA it also offers high quality summaries of key concepts, tools and methods that are used to determine evidence within legal proceedings. What it will do next is introduce any relevant cognitive or conceptual models. For example, sense making and foraging loops and then it will move into describing certain analytical techniques and methods. This will bring the subject matter up to date and will be concluded by the acknowledgment of the transition and challenges that cloud computing and IoT will bring to digital forensics.

Security Operations and Incident Management

What the KA of the security operations and incident management does is provide a solid statement of several key principles and components that would be anticipated to be included for SOC type considerations. Examples of these are architectural principles to logs, network flows, anomaly detection, IDS/IPS, SIEM, SOAR.

It will then lead to an overview of the Incident Management planning and process groundwork. Within certain places it can be well-trodden ground which can benefit from a broader and more varied contemporary industry input. Only a certain amount of consultation is actually feasible and affordable for a particular project and undertaking this is easier said than done.

The things that are covered are covered very well. Due to it being precise and authoritative narration describes how good practice looks whilst it acknowledges the many inherent fallibility tools, techniques and processes that are used to detect and stop all threats or to achieve the nirvana of total security. This state is impossible because it is acknowledged from the outset in the referencing of a report from 1981 by James Anderson. (Macrae, A. (2020)).

Machine Learning in Cyber Threat Intelligence

In the progression of threat intelligence organisations added machine learning technologies which improves attack detection. It allows computers to analyse data and learn why it is significant. The reason machine learning is used in threat intelligence is help computers detect attack more rapidly than humans and will stop the attacks before any more damage is caused. Due to the amount of threat intelligence being large, traditional detection produces too many false positives. Machine learning reduces the false positives by analysing the threat intelligence and compressing it into smaller sets, which makes it easier to look for things.

Machine learning threat intelligence works by taking inputs, analysing them, and producing outputs. The machine learning inputs for attack detection include threat intelligence and the outputs include either alerts indicating attacks or automate action stopping attacks. If there are any errors within the threat intelligence, the information it gives back will be bad, which means the output for machine learning algorithms may be bad.

Organisations use multiple threat intelligence sources such as, feeds. This consist of signs of attacks that are machine-readable. For example, an IP address of computers issuing attacks and the file name of the malware. Services are another form of threat intelligence. This provides human-readable prose that is used to describe new threats. Machine leaning is only able to use feeds. (Scarfone, K. (2020)).

Threat intelligence feeds for machine learning should be of high quality. Certain characteristics should be considered:

- The feed will need to be updated frequently because the threats can change quickly.
- Is the feed data accurate? E.g., is the IP address issuing attacks.
- How comprehensive is the feed? Are threats from around the world covered. Are types of information about the threats included that the threat detection tools need? (Scarfone, K. (2020)).

It can be difficult to directly assess the quality of threat intelligence but can be evaluated indirectly with the number of false positives.

If things machine learning is doing automatic block attacks, having false positive can be a concern. Benign activity could be disrupted if mistakes are made, and the operation could have a negative effect.

Threat intelligence is only one section of a risk assessment. Understanding the context is another important part. An example of this is the role, importance, and operational characteristics of each computer. If contextual information is provided to machine learning then more values can be received from threat intelligence. Consider the case where threat intelligence indicates a particular external IP address is malicious. Detecting outgoing network traffic from an internal database server to such an address might trigger a different action than detecting the same traffic from a server that sends file to subscribers every day.

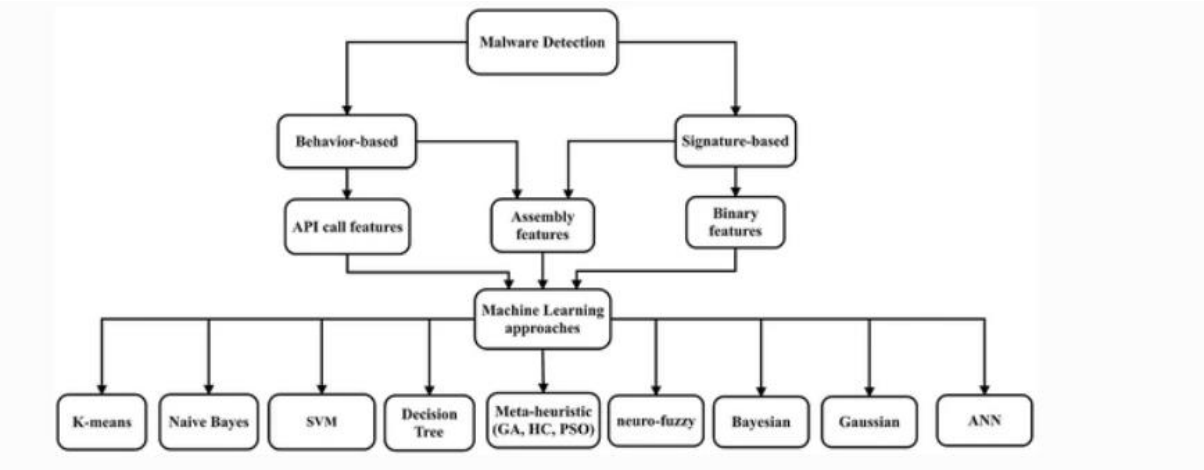
To avoid depending on machine learning to detect attacks, conventional wisdom is used, due to the concern of false positives. In some environments this would make sense, but not all. If the detection techniques are older then it could most likely miss the newest attacks. Machine learning is good for helping to find the newest attacks but has a higher rate of getting false positives. Automated machine learning may be used if missing attacks are more of a concern than investigating false positives.

For something organisations will probably not use machine learning and for other purposes they will get machine learning-generated insights. Machine learning could be used to find out what things could be investigated, that would be more difficult to find with larger data sets. (Scarfone, K. (2020)).

Machine Learning for Malware Detection

Due to the development of malware in innovation, obscure malware protection information is a fundamental topic in malware recognition according to machine learning strategies. These strategies are split up into supervised and unsupervised learning classes. With malware detection approaches they are split up into behaviour-based and signature-based methods. Static and dynamic malware analysis are usually performed in finding malicious applications.

Below is a diagram of a detection taxonomy on machine learning approaches. The diagram shows that the malware detection methods are the API calls features, assembly features and binary features. They all use machine learning for predicting and detecting malicious files. (Souri, A & Hosseini, R. (2018)).



(Souri, A & Hosseini, R. (2018)).

Machine Learning Approaches for Malware Detection

Decision Tree

This type of algorithm is made up of different node segments. For malware detection systems to be implemented using a decision tree, the sample data is repeated divided into nodes segments. Root node, internal node and leave are all included. It is mostly used for classification and regression tasks, for example human activity recognition, image classification and malware detection. They do not need assumption in order to predict on the training features if the malware is benign or malicious. It is simple to understand and efficient. Overfitting can reduce the performance of a decision tree algorithm. (ALO, U - NWEKE, H - ELE, S. (2021)).

Support Vector Machine

This type of algorithm is based on statistical learning and is a classification and regression algorithm. Training malware samples are separated when hyperplanes are deployed. It does this by using maximal margin. This will split the data into two malware classes, benign and malicious. Because of its diversity, robustness, and the capability of solving small scale or

high dimensional data, it is mainly used for malware detection. In order for it to achieve optimal detection accuracy it needs high and extensive data processing. (ALO, U - NWEKE, H - ELE, S. (2021)).

Logistic Regression

It provides an easy interpretation of a training sample feature vector. It is used to detect if a sample is malicious or benign. The performance of this model is affected by non-linear data but that does not stop it from being applied in the implementation of malware detection system. This is because of its low computational resource's requirements. (ALO, U - NWEKE, H - ELE, S. (2021)).

Random Forest

This algorithm is a collection of several decision trees. Each tree consists of random vectors that are distributed identically and independently. The results will be combined using voting techniques. It is split into ensemble learning and implementation categories which include '*training data collection decision split computation and combining the individual decision tree using majority voting'*. (ALO, U - NWEKE, H - ELE, S. (2021)).

Naïve Bayes

This is an algorithm that is used for supervised learning and pattern recognition. The algorithm can be trained quickly, simply, and efficiently. It cannot be used if the feature sets are correlated. In order for it to achieve optimum results, it needs extensive calculations of prior probability. (ALO, U - NWEKE, H - ELE, S. (2021)).

K-Nearest Neighbours

It is nonparametric and is based upon the principles of instance learning that will store malware data instances and classify new training data that use similarity index measures. Euclidean distance is the most common similarity index used in K-NN. It works best with handling large data samples and deliver fast and accurate detection of malicious samples. The major disadvantage is high computation, and the performance is affected by skewed data. (ALO, U - NWEKE, H - ELE, S. (2021)).

Deep Neural Networks

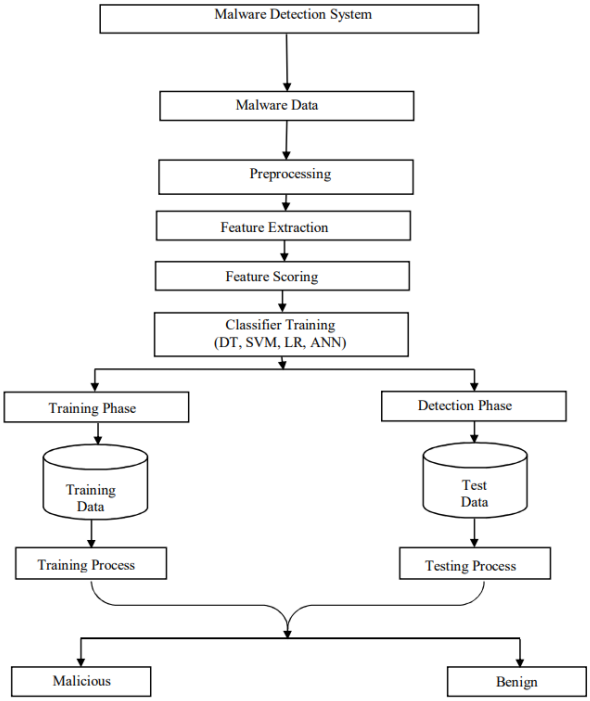
This algorithm uses AI to apply human learning process in order to gain insights from data. It consists of artificial neurons that identify information store. Artificial Neural Networks contains input layer, hidden layer, and output layer, which are used to transform input data to the desired outputs. Hidden layers are stacked in artificial neural networks to form deep neural networks. It is mainly used for data classification and predictions. It has recently been implemented for automatic feature representation. It does this by using deep learning methods. Discriminant features from training data are automatically discovered with deep learning algorithms. The approaches are then used for human activity identification, image classification, and natural language processing and recently in malware detection. Deep learning models such as deep neural networks, convolutional neural networks and long short-term memory have been investigated for malware detection and classification. Due to it being able to automatically learn high level feature vectors it makes it the main attraction

for using it in malware detection. It learns this from unlabelled data which produces high accuracy. On the downside, they require vast hyper-parameter tunning. The more hyper-parameters that are used it will increase the computation time of deep learning. (ALO, U - NWEKE, H - ELE, S. (2021)).

Multiple Classifier Systems

This system includes the integration of individual machine learning algorithms that will arrive at consensus so it can improve accuracy, robustness, and performance. Uncertainty and ambiguity can be reduced with homogeneous and heterogeneous classification algorithms. It does this with the integration of outputs that are generated by individual classification algorithms, which improve the results that are detected of each classification algorithm. Combination strategies are used to combine individual classification algorithm results. This consists of majority voting, posterior probabilities, mean aggregation, weighted summation, and Dempster -Shafer theory. (ALO, U - NWEKE, H - ELE, S. (2021)).

Below is a picture of a malware detection system:

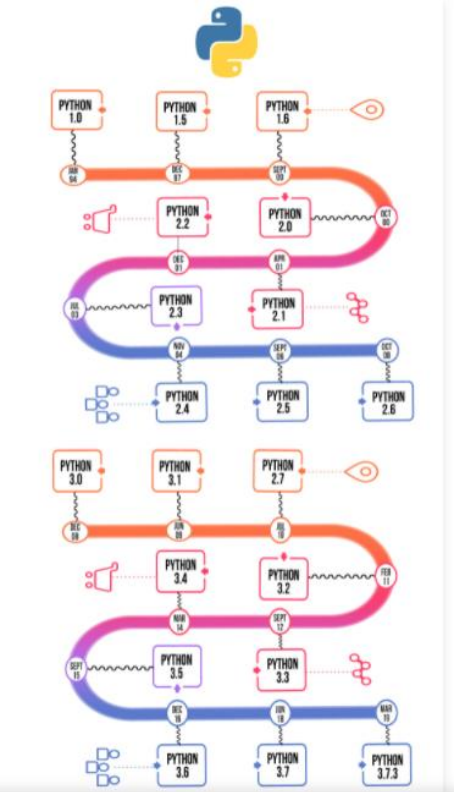


(ALO, U - NWEKE, H - ELE, S. (2021)).

History of Python

Python was designed in 1991 by Guido van Rossum and was then developed by the Python Software Foundation. The main reason it was developed was for emphasis on code readability. Programmers cab express concepts in a few lines with syntax.

Python started to get worked on in the late 1980s and the base application came into development in December 1989 by Centrum Wiskunde and Informatica, which is based in the Netherlands. It was only meant to be a hobby project. It was succeeded the ABC programming language and runs on the Amoeba Operating System. It featured exception handling. Guido had helped create ABC and knew all of the problems with it. What he did was take the ABC syntax and the features that were good. Due to the amount of complaints he had to fix a lot of issues. After he did this he had created a good scripting language that had no flaws. He called the language python because he was a fan of the show Monty Python, and he wanted a short and unique name. It was released in 1991 and only needed to use a few lines of code to express concepts whereas ither languages like Java, C++ and C needs more lines of code. Providing code readability and advances developer productivity is the main goal for python. Upon its release it had sufficient capability to provide classes with inheritance, core data types of exception handling and functions. Below is a picture of the various version of python and the timeline. (Pramanick, S. (2022)).



(Pramanick, S. (2022)).

Advantages of Python

Free and Open Source

Being free and open source is what makes python stand out. Being able to download it free means that developers can downloads its source code, make modifications, and distribute it. It comes with a large collection of libraries which helps in to support with carrying out tasks.

Collection of Inbuilt Libraries

It offers a lot of in-bult libraries that can be used for data manipulation, data mining and machine learning. Examples of these are:

- NumPy: this is used for scientific calculations.
- Scikit-learn - this is used for data mining which will optimise pythons machine learning usability. High performance structures and data analysis are offered by pandas and what this does is help to minimise the time it takes to implement the project.
- SciPy: this is used for advanced computation.
- Pybrain – this is used for machine learning.

Moderate Learning Curve

Python is an easy language to learn and use. Its main focus is code readability and is a versatile and structures language. Python can be simple of the material used is good or if a decent teacher teaches it.

General-purpose Programming Language

This means that python can be used to build nearly everything. It can be very useful for Backend Web Development, Artificial Intelligence, Scientific Computing, and Data Analysis. It is mainly used for web development, system operations, Server & Administrative tools, scientific modelling and can be used to build productivity tools, desktop apps, and games by several developers.

Easy to integrate

It is used as an integration language in various places, which wills tick the existing components together. It can be easily integrated with low level languages such as C, C++, or Java. A data scientist's work can have python-based stacks incorporated into it.

Easy to Create Prototypes

Prototypes can be created and tested quickly because python requires less code. It is easier to created then with python than any other language. It saves the developers time and minimises a company's overall expenditure. (Rose, S. (2019)).

How is Python good for Machine Learning?

An Amazing Collection of Libraries

Its great library is the one of the main reasons why machine learning uses and prefers python. Machine learning needs continuous data processing, and for this to be effective, python libraries lets people access, handle, and transform data.

These libraries are:

- Scikit-learn: this is used to handle basic machine learning algorithms such as clustering, regression, linear and logistic regressions, classification, etc.
- Pandas: this is used for data structures and analysis that are high level.
- TensorFlow: this works with deep learning and does this by setting up, training, and employing artificial neural networks that have large datasets.
- Keras: this is used for deep learning and allows fast calculation and prototyping. It utilises the GPU apart from the computers CPU.
- Matplotlib: this is used in the creation of 2D plots, histograms, charts, etc.
- Scikit-image: this is used to handle image processing.
- NLTK: this is used with language recognition, computational linguistics, and processing.
- PyBrain: this is used for neural network, unsupervised learning, and reinforced learning.
- Caffe: this is used for deep learning and allows the CPU and GPU to be switched between. It also processes over 60 million images a day by using only the NVIDIA k40 GPU.
- StatsModels: this is used to perform statistical algorithms and data exploration. (Rose, S. (2019)).

A Lower Barrier to Entry

Learning python is usually said to be like learning the English language. All that will need to be done is install Python and start using it for the development of machine learning because it would not take much to learn this language. The syntax is simple and lets developers work with complex systems very conveniently and ensuring that there are clear relations between the system elements.

Despite it being easy and having simple vocabulary it is still a high-level language regardless. It does most things and unlike with C++ developers will not be stuck in the minutiae.

Flexible

Another reason python is good for machine learning is how flexible it is. It lets developers chose the programming style that they think is the easiest for them to use. These programming styles can be combined and used to solve several types of problems in a productive way in a productive way.

Some of these styles are:

- Imperative style: this is used to define sequences of computations that happen like a change in the program state.
- Functional style: this is used to declare what the operation needs to perform without considering the state of the program. The statements are declared in the form of mathematical equations.
- Object-oriented style: this is based on two parameters, such as class and object.
- Procedural style: this style is more common with beginners because it performs tasks in a step-by-step format. It is used for sequencing, modularization, iteration, and selection. (Rose, S. (2019)).

Cross-Platform Compatibility

Not only is it known for being flexible and easy to use, and it is also well known for being versatile. This means that a python program that has been written on a Windows PC can run on any other operating system such as macOS, Linux, Unix and 21 other platforms and vice versa. Developers will need to perform a variety of micro-level changes in order to transfer from one program to another. Code will also need to be modified to create an executable form of code for the platform that is chosen.

PyInstaller can help prepare the code for running on different platforms. Not only does it save time and money it makes the process more concise.

Good Visualization Options

All AI and Machine Learning developers need to realise that data interpreted by the technologies are beyond human comprehension unless it's represented in a manner that is organised and a human readable form. Some of the libraries that python offers are great visualisation tools. Some of the most popular tools are matplotlib, seaborn, plot. It lets data scientists build charts, histograms, and plots to make it easier to understand the data. There are a lot of libraries for a lot of different types of data visualisation needs. (Rose, S. (2019)).

Disadvantages of Python

It Can Make Other Languages Harder to Learn

Due to python being simple and easy to learn ot makes learning new languages difficult for programmers. An example of this is adding curly braces or openly declaring types of variables might seem a gruelling task for python programmers.

It's Slower Than Compiled Languages

It is slower to compile than other languages because it is performed by an interpreter rather than a complier. With other languages such as C++ and Java are known as compiled languages and the code in the application is compiled to native system code before it runs, so the results of the application are optimised. Python on the other hand the code is interpreted at runtime and then will be converted into native system code. this means it will take more time to achieve.

Not Recommended for Mobile Computing

Unlike mobile applications that consume limited memory and CPU time, python on the other hand consumes large memory and CPU time to run. This means that python is rarely used to create mobile applications.

Run-Time Errors

There are a lot of design restrictions within python because it is dynamically typed. This is in reference to the evaluation of variable types at runtime as opposed to compile time. More time is needed to test applications that are written in python. There are times when the error shows up at the end pf the application when it is nearly complete. (Malik, U. (2019)).

History of Machine Learning

Machine learning might seem like a modern concept, but it dates back to the 1940s. Machine learning was not seen working until the 1950s. There is not one person who invented machine learning. However, the person that stands out is Arthur Samuel. He was a computer scientist and worked at IBM. He was also a pioneer in AI and gaming. He created the term Machine learning in 1952. The first thing he designed was a program for playing checkers. He make it so the more you played the more it learned from experience. This is done with a minimax algorithm that studies moves to come up with the winning strategy.

Machine learning did not take off until the late 1990s. This happened when IBM developed its Deep Blue supercomputer. There was a showdown between Deep Blue and world chess champion Garry Kasparov. They proved that machines were capable of being human-like intelligence by beating Kasparov with chess computer. AI technology would not be possible without machine learning. (Pandio. (2021)).

Advantages of Machine Learning

1. Machine learning is good because it is able to analyse large amounts of data and can detect any trends or patterns that cannot be detected with the human eye. An example of this is when Netflix or Amazon Prime use machine learning it used to understand the user's browsing habits and watch history. This will provide them with personalised recommendations.
2. The models are only as accurate as the data that is provided. The more experience the algorithm gains the accuracy and efficacy improves. Predictions will be more accurate in less time when the amount of data grows larger.
3. Machine learning is excellent for dealing with multi-dimensional and multi-variety data. It can be done in situation that are dynamic or unpredictable.
4. It does not need human attention or supervision. All a person needs to do is set up the model in order for it to be trained and to provide it with data for training, testing and validation. The worst thing that could happen is intervention involving diagnosing and remedying errors. When the model has been set up, human intervention is no longer needed.
5. One of the main advantages is it can be used in almost every field. Everything is made easier when machine learning is applied. For example, identifying the target user base which will provide reports, and seamless workflow. (Nixus. (2022)).

Disadvantages of Machine Learning

1. Data can be come inconsistent because the amount of data in a machine learning model that needs to be trained and tested is a lot. This dataset will need to be comprehensive, impartial, and high quality. If data is not updated regularly then the old and the new data might have different findings. The datasets could consist of false or incorrect information that could have an effect on the accuracy.
2. It takes quite a bit of time for the machine learning algorithm to learn and mature until it reaches their goal with a high level of accuracy. It can take a lot longer for

data to learn and process if there is a large amount of data and needs a lot of resources in order for it to run. This could cause the CPU to consume a lot of power. This is why its recommended to use GPU's.

3. It is very time consuming to solve machine learning problems because the model uses a range of algorithms which will determine which accuracy is the best. The only time an algorithm can be chosen is after it has been run and tested on the data to see if it will fit best with the model. The best way to choose these algorithms is with the accuracy of the results.
4. Predictions can be biased or incorrect if using the dataset for training an algorithm is not large enough to be comprehensive of all variations. This results in irrelevant recommendations. From a machine learning point of view, a mistake like this could cause a lot of errors that could be undiscovered for a long time. Eventually when they are detected it could take a while to discover what caused the problem and even more time to fix it. (Nixus. (2022)).

History of Threat Intelligence

A number of years ago IP and URL blacklists which is the progenitors of threat intelligences, was starting to emerge. These were then incorporated into tools such as security information and event management platforms and firewalls. Alerts and reports were created by this. Threats were manually searched for, and daily updates were communicated to customers by security researchers.

The use of the dark web and malicious activities started decades ago. This showed how very few security tools they had back then. The tools they did have were not able to identify or process the amount of indictors of compromise, IPs, malicious domains, and any other threats that were growing. This helped to make the cyber security industry become aware of the rising concern.

After they became aware organisations began to develop Artificial intelligence and machine learning capabilities that would correlate and automate the data on a different level. Large amounts of threat information were collected using millions of sensors and big data tools for processing and analysis. They way complex detection across different surfaces are performed is with big data tools. This led to the creation of threat intelligence.

After the discovery that big data was giving a lot of false positives, cyber security professional realised that human intervention was needed. Because of this discovery threat intelligence evolved. After some time, they started to manage the threat intelligence collections that were used to reduce the dales positives and to help better understand the threats and attack methods that are specific to an organisation. (Cyware. (2021)).

Advantages of Threat Intelligence

Cost Effective

This technology might seem like it costs a lot but in reality it can help to save a company millions in costs. Ponemon Institute confirmed that data breaches on average lost eight million dollars due to the cost of lawsuits, fines, fees, and a loss of sales because of a loss in reliability. Cyber threat intelligence can help to reduces these costs because it can help to create a plan of action that can be used to prevent and minimize any damage that was caused by attacks.

Improves the Efficiency of Your Security Team

Cyber Threat Intelligence is used to identify any potential threats to an organisation and will then detail the threats that need urgent attention, which in turn aids the security team in preparing adequately. This system help to increase efficiency among the team and does this by identifying threats, which will result in the team being able to focus on the important security threats. All the security team will need to do is check if there are any false positives, then the system will do the rest.

Lowers Risks

Cyber threat intelligence will inform any business of potential vulnerabilities that are present within their cyber security system, which will prevent hackers from using those weaknesses if immediate action is taken. Because of this the loss of data is minimised and the operations that are done every day can continue normally.

Prevents Data Breach

The way cyber threat intelligence prevents data breaches is by thoroughly checking for suspicious links, domains, or IP addresses that are trying to gain access to the enterprise. If the IP address is cynical , threat intelligence will ban it from entering the network which will prevent any loss of data.

Collaborative Knowledge

Cyber threat intelligence systems allow the sharing of cyber security practices and information threats within an organisation. With this organisations can learn any new threats that are affecting other organisations, which can help them to prepare appropriately. Companies can also share their tips and ways to prevent attacks with each other to make sure that everybody is unites against threat actors.

In-Depth Cyber Threat Analysis

Cyber threat intelligence will store a variety of information from previous attacks which can help to inform an organisation on the methods needed to carry out such an attack. It can also help to establish ironclad security protocols, which means the organisation will be prevented from suffering from these attacks. (Roohparvar, R. (2021)).

Disadvantages of Threat Intelligence

Mismatch with Particular Cybersecurity Needs

A lot of cyber security teams see threat intelligence as a quick fix that protects them from hackers and scammers. Have this hight of an expectation is highly overinflated. There is in fact no one size fits all threat intelligence.

They however need to be implemented as per the security requests for each organisation, suborganisations or departments. Or whatever is being achieved is collecting data that is not relevant and it gives a false sense of security.

No Resources to Act Upon Threat Intelligence

Forty-four percent of daily security threats will not be investigated, and the data within threat intelligence might end ip unutilized as well for several reasons.

Reasons for this could be that no one is knows how to interpret what they are looking at, let alone acting on it. A lack of leaderships commitment to the cause can also be the problem.

Knowing that there is a problem but not understanding the security flaws or having the means to solve them the situation will not reduce the prevalence or the intensity of cyber-attacks.

Treating Threat Intelligence Like Any Other Cybersecurity Effort

The connection between threat intelligence and cyber security is undeniable. Threat intelligence provides direction to security awareness undertakings, spot server misconfigurations and to keep up to date of any new types of malware. Because of this it is easy to assume that cyber security professionals are ready to handle threat intelligence like a professional. There is a large difference in orientation and methodology.

Failing to Integrate Threat Intelligence

The way to make sure cyber security staff use threat intelligence insights is by linking innovations to whatever the user knows already. When there is a lack of integration it makes threat intelligence less effective and adds more to the cyber security team's workload that will need to be manually assembled and data will be compared from another course to evaluate the well-being of the infrastructure. (Zhang, J. (2018)).

Implementation

During this section of the report, the steps taken during the implementation of this project and the pictures of the code used and an explanation of what each of them does and the results will be written below.

Model 1

Step 1

Firstly, the important libraries were imported.

```
import pandas
import numpy
import matplotlib.pyplot as plt

import sklearn.ensemble as ek
from sklearn.feature_selection import SelectFromModel

from sklearn.model_selection import train_test_split
from sklearn.tree import plot_tree
from sklearn.metrics import confusion_matrix, plot_confusion_matrix
```

Step 2

The pandas.read_csv was used to import the dataset and dataset.head() to print you the dataset and all of its values.

Dataset 1

```
[4]: dataset = pandas.read_csv('Malware dataset.csv')
```

```
[5]: dataset.head()
```

[5]:

	hash	millisecond	classification	state	usage_counter	prio	static_prio	normal_prio	policy	vm_pgoff	.
0	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	0	malware	0	0	3069378560	14274	0	0	0	.
1	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	1	malware	0	0	3069378560	14274	0	0	0	.
2	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	2	malware	0	0	3069378560	14274	0	0	0	.
3	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	3	malware	0	0	3069378560	14274	0	0	0	.
4	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	4	malware	0	0	3069378560	14274	0	0	0	.

5 rows × 35 columns

Dataset 2

	Name	md5	Machine	SizeOfOptionalHeader	Characteristics	MajorLinkerVersion	MinorLinkerVersion	SizeOfCode	SizeOfI
0	memtest.exe	631ea355665f28d4707448e442fbf5b8	332	224	258	9	0	361984	
1	ose.exe	9d10f99a6712e28f8acd5641e3a7ea6b	332	224	3330	9	0	130560	
2	setup.exe	4d92f518527353c0db88a70fddcfd390	332	224	3330	9	0	517120	
3	DW20.EXE	a41e524f8d45f0074fd07805ff0c9b12	332	224	258	9	0	585728	
4	dwtrig20.exe	c87e561258f2f8650cef999bf643a731	332	224	258	9	0	294912	

5 rows × 57 columns

Dataset 3

	protocol	flow_duration	total_forward_packets	total_backward_packets	total_forward_packets_length	total_backward_packets_length	forward_packet_length_m
0	17	2468	4	0	1580.0	0.0	39
1	17	133	4	0	5888.0	0.0	147
2	17	33509	200	0	88000.0	0.0	44
3	17	288495	200	0	88000.0	0.0	44
4	17	9	2	0	2062.0	0.0	103

Step 3

During the next step the describe function was used to outputs the mean, median, standard deviation, minimum, maximum and percentiles.

In [6]:
```
dataset.describe()
```

Out[6]:

	millisecond	state	usage_counter	prio	static_prio	normal_prio	policy	vm_pgoff	vm_truncate_count	task_size	...	ni
count	100000.000000	1.000000e+05	100000.0	1.000000e+05	100000.000000	100000.0	100000.0	100000.0	100000.000000	100000.0	...	100000.00
mean	499.500000	1.577683e+05	0.0	3.069706e+09	18183.900070	0.0	0.0	0.0	15312.739510	0.0	...	32.95
std	288.676434	9.361726e+05	0.0	2.963061e+05	4609.792765	0.0	0.0	0.0	3256.475008	0.0	...	52.73
min	0.000000	0.000000e+00	0.0	3.069190e+09	13988.000000	0.0	0.0	0.0	9695.000000	0.0	...	0.00
25%	249.750000	0.000000e+00	0.0	3.069446e+09	14352.000000	0.0	0.0	0.0	12648.000000	0.0	...	1.00
50%	499.500000	0.000000e+00	0.0	3.069698e+09	16159.000000	0.0	0.0	0.0	15245.000000	0.0	...	9.00
75%	749.250000	4.096000e+03	0.0	3.069957e+09	22182.000000	0.0	0.0	0.0	17663.000000	0.0	...	46.00
max	999.000000	4.326605e+07	0.0	3.070222e+09	31855.000000	0.0	0.0	0.0	27157.000000	0.0	...	365.00

8 rows × 33 columns

Step 4

The groupby command is used to detect what is malware and what is benign. I used the classification column which shows it has malware in it. Below are the results of all the datasets and they show how many are benign and how many are malware.

Dataset 1

```
dataset.groupby(dataset['classification']).size()
```

```
classification
benign     50000
malware    50000
dtype: int64
```

Dataset 2

```
dataset.groupby(dataset['legitimate']).size()
```

```
legitimate
0    96724
1    41323
dtype: int64
```

Dataset 3

```
dataset.groupby(dataset['label']).size()
```

```
label
BENIGN        1128
DrDoS_DNS    32797
dtype: int64
```

Step 5

Next the data was split. Doing this is important because it will give an unbiased evaluation of the performance. The dataset will be split into three subsets. Training set, validation set and test set.

Training set is used to train or fit a model. An example of this is to find an ideal weight or coefficient for linear regression, logistic regression, or neural networks.

Validation set is used during hyperparameter tuning to evaluate a model unbiasedly. An example of this is experimenting with different values to try and find optimal neurons within a neural network or to find a kernel that is the best for a support vector. For each hyperparameter setting, a training setting will need to be fitted into the model and then a validation set will be used to assess the performance.

Test set is used to evaluate the final model and cannot be use for fitting or validation. (Stojiljković, M. (2021)).

```
In [9]: X_train, X_test, y_train, y_test = train_test_split(X, y ,test_size=0.2, shuffle=True)
```

Step 6

Next the number of samples that are in the training and test set were printed out. Below are the about of training and test datasets for all three of the datasets.

Dataset 1

```
print("Number of samples in Training Set =", X_train.shape[0])
print("Number of samples in Test Set =", X_test.shape[0])
```

```
Number of samples in Training Set = 80000
Number of samples in Test Set = 20000
```

Dataset 2

```
print("Number of samples in Training Set =", X_train.shape[0])
print("Number of samples in Test Set =", X_test.shape[0])
```

```
Number of samples in Training Set = 110437
Number of samples in Test Set = 27610
```

Dataset 3

```
print("Number of samples in Training Set =", X_train.shape[0])
print("Number of samples in Test Set =", X_test.shape[0])
```

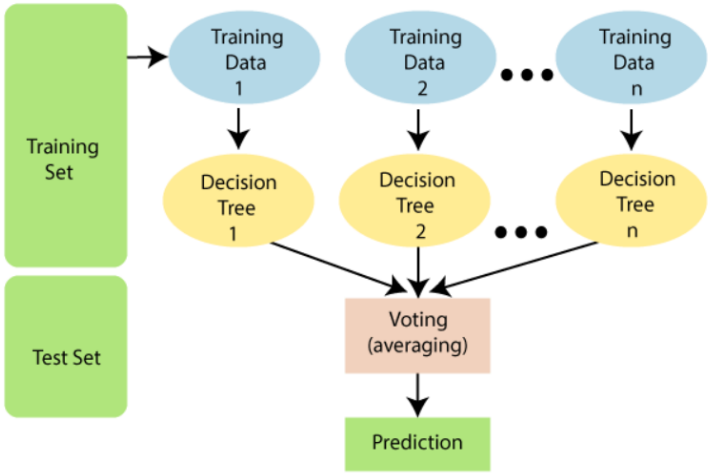
```
Number of samples in Training Set = 27140
Number of samples in Test Set = 6785
```

Step 7

Next the random forest algorithm was used, which is a supervised learning technique. It can be used for classification and regression. It is based on a process that has a combination of multiple classifiers that are used to solve complex problems and improve a model's performance.

It contains several decision trees on a variety of subsets of a dataset and the accuracy will be improved by using the average of the dataset. It does not reply on one decision three but will take prediction from each of them depending on the predictions majority votes and it will then predict the final output. The more decision trees there are, means the accuracy will be higher and will help stop any overfitting problems. (Christopher, A. (2021)).

Below is a diagram of how the random forest algorithm works and the code that was used.



(Christopher, A. (2021)).

```
In [12]: forest = ek.RandomForestClassifier(n_estimators=100, oob_score=True,)
         forest.fit(X_train,y_train)

Out[12]: RandomForestClassifier(oob_score=True)
```

Step 8

Next I used the forest.score to calculate the accuracy of the random forest then used print to print the score. As shown below the accuracy for this dataset is 1.0.

Dataset 1

```
In [13]: score = forest.score(X_test,y_test)
         print("Random Forest Accuracy:", score)

         Random Forest Accuracy: 1.0
```

Dataset 2

```
Random Forest Accuracy: 0.9949293734154292
```

Dataset 3

```
Random Forest Accuracy: 0.999852616064849
```

Step 9

Next the confusion matrix function was used which is used to predict the results of a classification problem. Any correct or incorrect predictions will be summarised with the count vale and will then be split into each class. This is important in confusion matrix. Not only does it show the errors that the classifier is making but also the kinds of errors it is making. Because of this using makes it easier to use the classification accuracy on its own. (Brownlee, J. (2016)).

How to calculate a confusion matrix:

1. A dataset or a validation dataset will need to be tested with expected outcome values.
2. A prediction will need to be made for each row within the test dataset.
3. The number of correct and incorrect prediction from each class will come from the expected outcomes and the predictions count. (Brownlee, J. (2016)).
   In the code below I used the confusion matrix to predict if there were any false positives or false negatives. A false positive is where the model predicts a positive class incorrectly and a false negative is where the model predicts the negative class incorrectly. Below are the results for all three datasets.

Dataset 1

```
In [14]: predicted = forest.predict(X)
         matrix = confusion_matrix(y, predicted)
         false_positive = (matrix[0][1] / float(sum(matrix[0])))*100
         false_negative = (matrix[1][0] / float(sum(matrix[1])))*100
         print("False positive rate: " + str(false_positive) + "%")
         print("False negative rate: " + str(false_negative) + "%")

         False positive rate: 0.0%
         False negative rate: 0.0%
```

Dataset 2

```
False positive rate: 0.08270956536123403%
False negative rate: 0.1524574692060112%
```

Dataset 3

```
False positive rate: 0.08865248226950355%
False negative rate: 0.0%
```

Step 10

Next a plot_confusion_matrix was created which shows the predicted labels and true labels. The model accuracy which is 1 was then printed. The accuracy can be obtained from the confusion matrix by using the calculation:

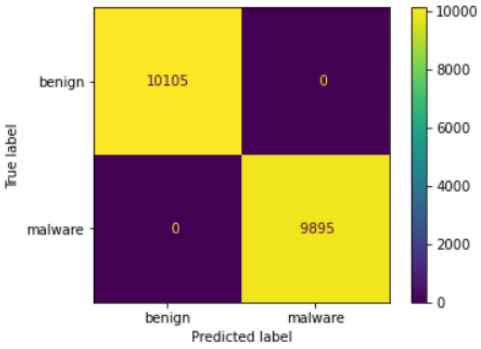
Accuracy = (Sum of values on the main diagonal)/(Sum of all values on the matrix). (Fish, D. (2020).

Below are the results for all three of the datasets:

Dataset 1

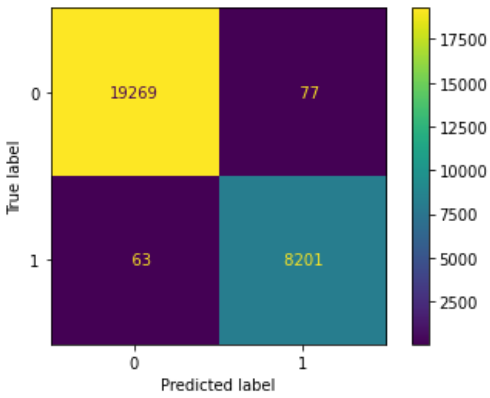
```
confusionmatrix = plot_confusion_matrix(forest, X_test, y_test, values_format='5d')
print("Model Accuracy: %f" % forest.score(X_test, y_test))
plt.show()
```

```
Model Accuracy: 1.000000
```

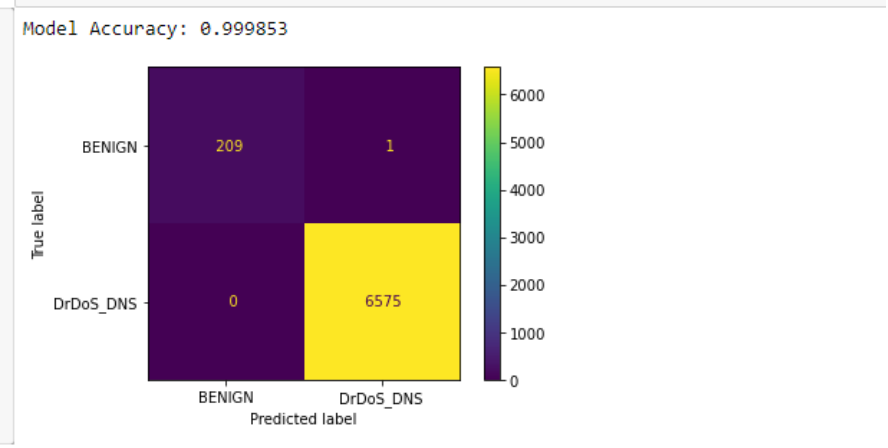


Dataset 2

```
Model Accuracy: 0.994929
```



Dataset 3



Step 11

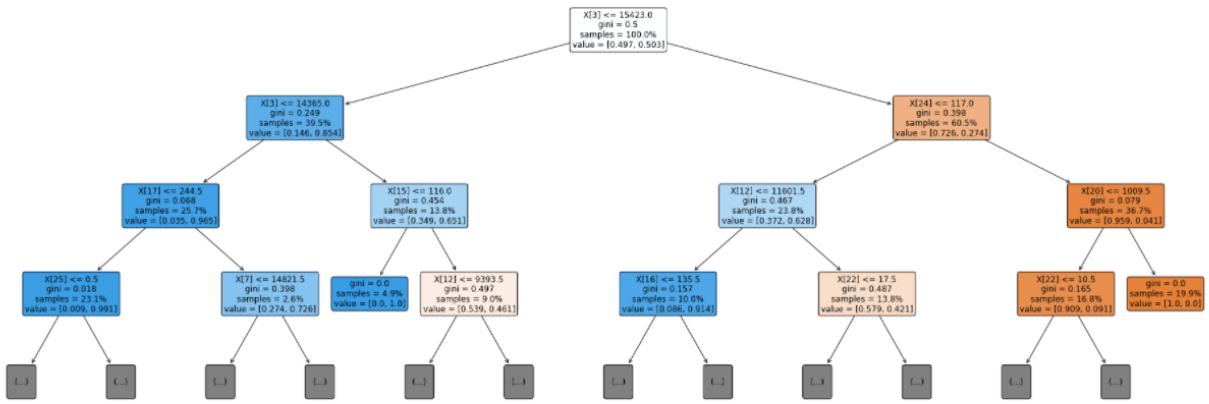
Next the plot_tree function was used to create a decision tree. The type of approach used to for decision trees is a top-down approach.

- A leaf is used to provide the classification of any given instance.
- A node is used to specify a test of attributes of the instance.
- A branch will correspond to possible values of an attribute.
- The way an instance will be classified is by starting at the root of the tree, the attribute that is specified by the node will then be tested, moving down the branch in the tree that corresponds to the value of the attribute in the given example. (Hjorth-Jensen, M. (2021)).

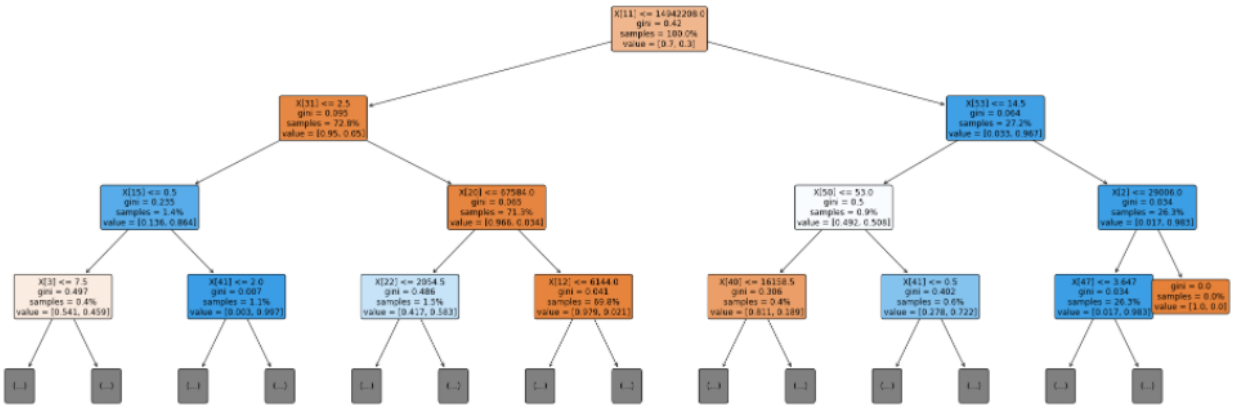
Dataset 1

```
In [16]: plt.figure(figsize=(33,12))

         plot_tree(forest.estimators_[0], max_depth=3, filled=True, proportion=True, rounded=True, fontsize=12)
         plt.show()
```



Dataset 2



Dataset 3

Step 12

Next pickle was imported. Python object structures will be serialised or de-serialised with pickle. Serialising is used to convert an object to a byte stream within memory and can be stored on a disk or sent to a network. When the character stream is received it will be de-serialised back into the python object. Pickle cannot be mistaken with compression. Objects are converted from one representation ((RAM)) to another (text on disk), while the latter is the process of encoding data with fewer bits, to save disk space. (Camp, D. (2018)).

The *pickle.dump* command is used to write the python object into the file object or BytesIO object or any destination object that has a *write()* method interface that accepts byte arguments. (Pythontic. (2020)).

```
In [17]: import pickle

         pickle.dump(forest, open('model.pkl', 'wb'))
```

Step 13

Next the pickle.load function was used. What this does is it reads from a file one or more pickle byte stream. The *load()* method will be called several times when several objects are expected from the byte stream. (Pythontic. (2020)).

```
In [18]: forest = pickle.load(open('model.pkl', 'rb'))
```

Step

Step 14

Next the *print(forest)* function was used to print the OOB score, which is used to validate the random forest model.

```
In [19]: print(forest)

         RandomForestClassifier(oob_score=True)
```

Step 15

Lastly, the random forest score was printed out. What this does is measures the amount of labels the model go right out of all the predictions.

Below are the results for each dataset that I have used:

Dataset 1

```
In [20]: print(forest.score(X_test, y_test))

         1.0
```

Dataset 2

```
In [25]: print(forest.score(X_test, y_test))

         0.9949293734154292
```

Dataset 3

```
In [16]: print(forest.score(X_test, y_test))

         0.999852616064849
```

(premnagdeo . (2020)).

Model 2

Step 1

The first step was importing the libraries needed for this model.

```
#importing the required Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
```

Step 2

Next the dataset needed was imported and generated.

Dataset 1

```
In [2]: #Reading CSV Files
        data=pd.read_csv("Malware dataset.csv")
        data.head()
```

Out[2]:

	hash	millisecond	classification	state	usage_counter	prio	static_prio	normal_prio	policy	vm_pgoff	.
0	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	0	malware	0	0	3069378560	14274	0	0	0	.
1	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	1	malware	0	0	3069378560	14274	0	0	0	.
2	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	2	malware	0	0	3069378560	14274	0	0	0	.
3	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	3	malware	0	0	3069378560	14274	0	0	0	.
4	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	4	malware	0	0	3069378560	14274	0	0	0	.

5 rows × 35 columns

Dataset 2

	Name	md5	Machine	SizeOfOptionalHeader	Characteristics	MajorLinkerVersion	MinorLinkerVersion	SizeOfCode	SizeOfI
0	memtest.exe	631ea355665f28d4707448e442fbf5b8	332	224	258	9	0	361984	
1	ose.exe	9d10f99a6712e28f8acd5641e3a7ea6b	332	224	3330	9	0	130560	
2	setup.exe	4d92f518527353c0db88a70fddcfd390	332	224	3330	9	0	517120	
3	DW20.EXE	a41e524f8d45f0074fd07805ff0c9b12	332	224	258	9	0	585728	
4	dwtrig20.exe	c87e561258f2f8650cef999bf643a731	332	224	258	9	0	294912	

5 rows × 57 columns

Dataset 3

	protocol	flow_duration	total_forward_packets	total_backward_packets	total_forward_packets_length	total_backward_packets_length	forward_packet_length_m
0	17	2468	4	0	1580.0	0.0	3
1	17	133	4	0	5888.0	0.0	14
2	17	33509	200	0	88000.0	0.0	4
3	17	288495	200	0	88000.0	0.0	4
4	17	9	2	0	2062.0	0.0	10

Step 3

Next the data.shape function was used to check the dimensions of the dataframe.

Dataset 1

```
data.shape
```

```
(100000, 35)
```

Dataset 2

```
df.shape
```

```
(138047, 57)
```

Dataset 3

```
data.shape
```

```
(33925, 16)
```

Step 4

Next the data.isnull().sum() function was used. What this does is count the number of values that are missing within the dataframe. Insull() is used to identify these values then sum is used to count them. (Ebner, J. (2021)). As shown in the pictures below there are no missing values in any of the datasets.

Dataset

```
data.isnull().sum()
```

```
hash                0
millisecond         0
classification      0
state               0
usage_counter       0
prio                0
static_prio         0
normal_prio         0
policy              0
vm_pgoff            0
vm_truncate_count   0
task_size           0
cached_hole_size    0
free_area_cache     0
mm_users            0
map_count           0
hiwater_rss         0
total_vm            0
shared_vm           0
exec_vm             0
reserved_vm         0
nr_ptes             0
end_data            0
last_interval       0
nvcsw               0
nivcsw              0
min_flt             0
maj_flt             0
fs_excl_counter     0
lock                0
utime               0
stime               0
gtime               0
cgtime              0
signal_nvcsw        0
dtype: int64
```

Dataset 2

```
Name                         0
md5                          0
Machine                      0
SizeOfOptionalHeader         0
Characteristics              0
MajorLinkerVersion           0
MinorLinkerVersion           0
SizeOfCode                   0
SizeOfInitializedData        0
SizeOfUninitializedData      0
AddressOfEntryPoint          0
BaseOfCode                   0
BaseOfData                   0
ImageBase                    0
SectionAlignment             0
FileAlignment                0
MajorOperatingSystemVersion  0
MinorOperatingSystemVersion  0
MajorImageVersion            0
MinorImageVersion            0
MajorSubsystemVersion        0
MinorSubsystemVersion        0
SizeOfImage                  0
SizeOfHeaders                0
CheckSum                     0
Subsystem                    0
DllCharacteristics           0
SizeOfStackReserve           0
SizeOfStackCommit            0
SizeOfHeapReserve            0
SizeOfHeapCommit             0
LoaderFlags                  0
NumberOfRvaAndSizes          0
SectionsNb                   0
SectionsMeanEntropy          0
SectionsMinEntropy           0
SectionsMaxEntropy           0
SectionsMeanRawsize          0
SectionsMinRawsize           0
SectionMaxRawsize            0
SectionsMeanVirtualsize      0
SectionsMinVirtualsize       0
SectionMaxVirtualsize        0
ImportsNbDLL                 0
ImportsNb                    0
ImportsNbOrdinal             0
ExportNb                     0
ResourcesNb                  0
ResourcesMeanEntropy         0
ResourcesMinEntropy          0
ResourcesMaxEntropy          0
ResourcesMeanSize            0
ResourcesMinSize             0
ResourcesMaxSize             0
LoadConfigurationSize        0
VersionInformationSize       0
legitimate                   0
dtype: int64
```

Dataset 3

```
protocol                     0
flow_duration                0
total_forward_packets        0
total_backward_packets       0
total_forward_packets_length 0
total_backward_packets_length 0
forward_packet_length_mean   0
backward_packet_length_mean  0
forward_packets_per_second   0
backward_packets_per_second  0
forward_iat_mean             0
backward_iat_mean            0
flow_iat_mean                0
flow_packets_per_seconds     0
flow_bytes_per_seconds       0
label                        0
dtype: int64
```

Step 5

Next the data.columns function was used to show all of the columns within the dataset, as using data.head() only provides a few of them.

Dataset 1

```
data.columns
```

```
Index(['hash', 'millisecond', 'classification', 'state', 'usage_counter',
       'prio', 'static_prio', 'normal_prio', 'policy', 'vm_pgoff',
       'vm_truncate_count', 'task_size', 'cached_hole_size', 'free_area_cache',
       'mm_users', 'map_count', 'hiwater_rss', 'total_vm', 'shared_vm',
       'exec_vm', 'reserved_vm', 'nr_ptes', 'end_data', 'last_interval',
       'nvcsw', 'nivcsw', 'min_flt', 'maj_flt', 'fs_excl_counter', 'lock',
       'utime', 'stime', 'gtime', 'cgtime', 'signal_nvcsw'],
      dtype='object')
```

Dataset 2

```
df.columns
```

```
Index(['Name', 'md5', 'Machine', 'SizeOfOptionalHeader', 'Characteristics',
       'MajorLinkerVersion', 'MinorLinkerVersion', 'SizeOfCode',
       'SizeOfInitializedData', 'SizeOfUninitializedData',
       'AddressOfEntryPoint', 'BaseOfCode', 'BaseOfData', 'ImageBase',
       'SectionAlignment', 'FileAlignment', 'MajorOperatingSystemVersion',
       'MinorOperatingSystemVersion', 'MajorImageVersion', 'MinorImageVersion',
       'MajorSubsystemVersion', 'MinorSubsystemVersion', 'SizeOfImage',
       'SizeOfHeaders', 'CheckSum', 'Subsystem', 'DllCharacteristics',
       'SizeOfStackReserve', 'SizeOfStackCommit', 'SizeOfHeapReserve',
       'SizeOfHeapCommit', 'LoaderFlags', 'NumberOfRvaAndSizes', 'SectionsNb',
       'SectionsMeanEntropy', 'SectionsMinEntropy', 'SectionsMaxEntropy',
       'SectionsMeanRawsize', 'SectionsMinRawsize', 'SectionMaxRawsize',
       'SectionsMeanVirtualsize', 'SectionsMinVirtualsize',
       'SectionMaxVirtualsize', 'ImportsNbDLL', 'ImportsNb',
       'ImportsNbOrdinal', 'ExportNb', 'ResourcesNb', 'ResourcesMeanEntropy',
       'ResourcesMinEntropy', 'ResourcesMaxEntropy', 'ResourcesMeanSize',
       'ResourcesMinSize', 'ResourcesMaxSize', 'LoadConfigurationSize',
       'VersionInformationSize', 'legitimate'],
      dtype='object')
```

Dataset 3

```
data.columns
```

```
Index(['protocol', 'flow_duration', 'total_forward_packets',
       'total_backward_packets', 'total_forward_packets_length',
       'total_backward_packets_length', 'forward_packet_length_mean',
       'backward_packet_length_mean', 'forward_packets_per_second',
       'backward_packets_per_second', 'forward_iat_mean', 'backward_iat_mean',
       'flow_iat_mean', 'flow_packets_per_seconds', 'flow_bytes_per_seconds',
       'label'],
      dtype='object')
```

Step 6

Next data1=data.dropna(how="any",axis=0), data1.head() was used to do get rid of any missing values.

```
data1=data.dropna(how="any",axis=0)
data1.head()
```

	hash	millisecond	classification	state	usage_count
0	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	0	malware	0	
1	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	1	malware	0	
2	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	2	malware	0	
3	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	3	malware	0	
4	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	4	malware	0	

5 rows × 35 columns

Step 7

As done in the previous model the value_count function was used again to see count the amount of malware and benign files.

```
: data1["classification"].value_counts()
```

```
: malware    50000
  benign     50000
  Name: classification, dtype: int64
```

Step 8

The map function was used to apply a function to an item within an iterable. A new iterable will then be returned, that will then be used in the code. As shown in the picture where malware used to be is now changed to 1. The second dataset did not need to do this as the value was already 1.

```
data1['classification'] = data1.classification.map({'benign':0, 'malware':1})
```

```
data1.head()
```

	hash	millisecond	classification	state	usage_count
0	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	0	1	0	
1	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	1	1	0	
2	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	2	1	0	
3	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	3	1	0	
4	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	4	1	0	

5 rows × 35 columns

Step 9

Next the data1.tail() function was used. The last row will be returned when this function is used depending on its position. It is good for verifying data quickly. W3Resources. (2020).

Dataset 1

```
data1.tail()
```

	hash	millisecond	classification	state	usage_counter	prio	static_prio	normal_prio	policy	vm_p
99995	025c63d266e05d9e3bd57dd9ebd0abe904616f569fe4e2...	995	1	4096	0	3070148608	13988	0	0	
99996	025c63d266e05d9e3bd57dd9ebd0abe904616f569fe4e2...	996	1	4096	0	3070148608	13988	0	0	
99997	025c63d266e05d9e3bd57dd9ebd0abe904616f569fe4e2...	997	1	4096	0	3070148608	13988	0	0	
99998	025c63d266e05d9e3bd57dd9ebd0abe904616f569fe4e2...	998	1	4096	0	3070148608	13988	0	0	
99999	025c63d266e05d9e3bd57dd9ebd0abe904616f569fe4e2...	999	1	4096	0	3070148608	13988	0	0	

5 rows × 35 columns

Dataset 2

	Name	md5	Machine	SizeOfOptionalHeader	Characteristics	MajorLinkerVersion
138042	VirusShare_8e292b418568d6e7b87f2a32aee7074b	8e292b418568d6e7b87f2a32aee7074b	332	224	258	11
138043	VirusShare_260d9e2258aed4c8a3bbd703ec895822	260d9e2258aed4c8a3bbd703ec895822	332	224	33167	2
138044	VirusShare_8d088a51b7d225c9f5d11d239791ec3f	8d088a51b7d225c9f5d11d239791ec3f	332	224	258	10
138045	VirusShare_4286dccf67ca220fe67635388229a9f3	4286dccf67ca220fe67635388229a9f3	332	224	33166	2
138046	VirusShare_d7648eae45f09b3adb75127f43be6d11	d7648eae45f09b3adb75127f43be6d11	332	224	258	11

5 rows × 57 columns

Dataset 3

```
data1.tail()
```

	protocol	flow_duration	total_forward_packets	total_backward_packets	total_forward_packets_length	total_backward_packets_length	forward_packet_lengt
33920	17	236	2	0	2848.0	0.0	
33921	17	47	2	0	274.0	0.0	
33922	17	3	2	0	1158.0	0.0	
33923	17	6	2	0	866.0	0.0	
33924	17	45	2	0	2864.0	0.0	

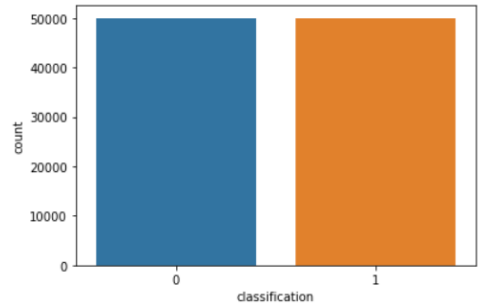
Step 10

The next two function are a seaborn graph and a pie chart to show the visual difference between the amount of benign and malware files.

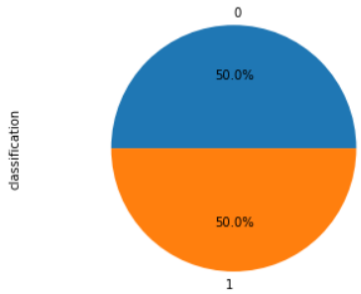
Dataset 1

```
sns.countplot(data1["classification"])
plt.show()
```

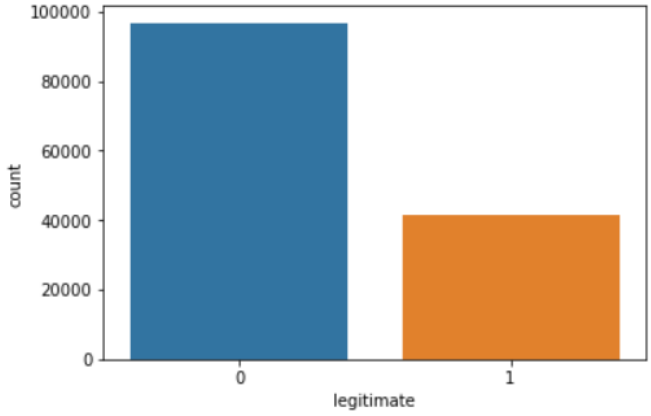
```
C:\Users\mishk\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword a
rg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```

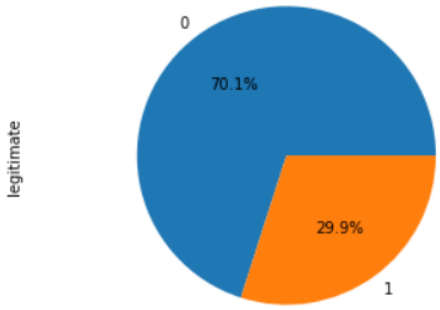



```
data1["classification"].value_counts().plot(kind="pie",autopct="%1.1f%%")
plt.axis("equal")
plt.show()
```

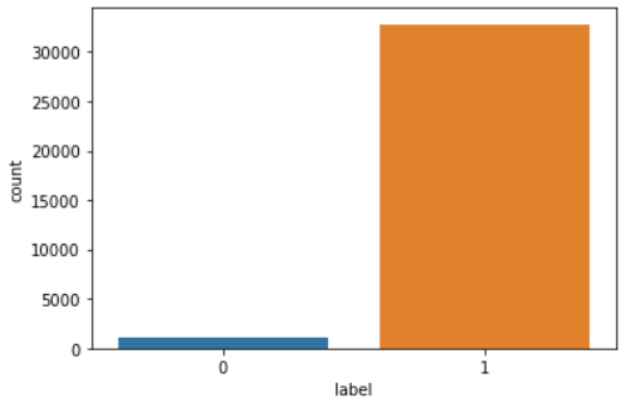


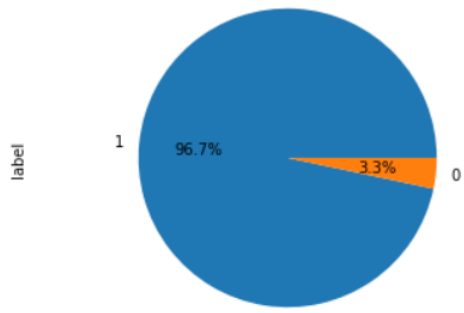
Dataset 2





Dataset 3





Step 11

Next the .loc function was used. This will access row or columns by labels or a boolean array.

The only inputs that are allowed are:

- A single label e.g., 5 or a
- A list or an array of labels e.g., a,b,c
- A slice object with labels e.g., a:f
- A boolean array that is the same length as the axis that is being sliced. e.g., True, False, True.
- A cable function that has one argument and that will return a valid output for indexing. (W3Resources. (2020)).

Dataset 1

```
In [13]: benign1=data.loc[data['classification']=='benign']
         benign1["classification"].head()

Out[13]: 1000    benign
         1001    benign
         1002    benign
         1003    benign
         1004    benign
         Name: classification, dtype: object

In [14]: malware1=data.loc[data['classification']=='malware']
         malware1["classification"].head()

Out[14]: 0    malware
         1    malware
         2    malware
         3    malware
         4    malware
         Name: classification, dtype: object
```

Dataset 2

```
: benign1=df.loc[df['legitimate']=='0']
  benign1["legitimate"].head()

: Series([], Name: legitimate, dtype: int64)

: malware1=df.loc[df['legitimate']=='1']
  malware1["legitimate"].head()

: Series([], Name: legitimate, dtype: int64)
```

Dataset 3

```
benign1=data.loc[data['label']=='BENIGN']
benign1["label"].head()
```

```
8      BENIGN
12     BENIGN
51     BENIGN
146    BENIGN
164    BENIGN
Name: label, dtype: object
```

```
malware1=data.loc[data['label']=='DrDoS_DNS']
malware1["label"].head()
```

```
0    DrDoS_DNS
1    DrDoS_DNS
2    DrDoS_DNS
3    DrDoS_DNS
4    DrDoS_DNS
Name: label, dtype: object
```

Step 12

Next the .corr() function was used. This will search all the columns in the dataframe to find the pairwise correlations. If there are any values, they will be excluded. Non-numerical columns will be ignored. (W3Resources. (2020)).

Dataset 1

```
corr=data1.corr()
corr.nlargest(35,'classification')["classification"]
```

```
classification       1.000000e+00
prio                 1.100359e-01
last_interval        6.952036e-03
min_flt              3.069595e-03
millisecond          5.482134e-15
gtime               -1.441608e-02
stime               -4.203713e-02
free_area_cache     -5.123678e-02
total_vm            -5.929110e-02
state               -6.470178e-02
mm_users            -9.364091e-02
reserved_vm         -1.186078e-01
fs_excl_counter     -1.378830e-01
nivcsw              -1.437912e-01
exec_vm             -2.551234e-01
map_count           -2.712274e-01
static_prio         -3.179406e-01
end_data            -3.249535e-01
maj_flt             -3.249535e-01
shared_vm           -3.249535e-01
vm_truncate_count   -3.548607e-01
utime               -3.699309e-01
nvcsw               -3.868893e-01
Name: classification, dtype: float64
```

Dataset 2

```
legitimate                    1.000000
Machine                       0.548835
SizeOfOptionalHeader          0.547498
Subsystem                     0.514352
MajorSubsystemVersion         0.380393
VersionInformationSize        0.379646
ResourcesMinEntropy           0.299112
Characteristics               0.221956
ExportNb                      0.134408
ImportsNbOrdinal              0.128112
FileAlignment                 0.125169
ImportsNb                     0.116415
ResourcesNb                   0.090405
MajorImageVersion             0.084410
MinorImageVersion             0.083220
SectionsMinRawsize            0.059346
SectionsMinVirtualsize        0.056466
ImportsNbDLL                  0.038395
SizeOfCode                    0.017476
MajorLinkerVersion            0.017320
SizeOfHeaders                 0.010125
ImageBase                     0.008245
MajorOperatingSystemVersion   0.002402
SectionsMeanVirtualsize       0.001734
SectionsMeanRawsize           0.001175
AddressOfEntryPoint          -0.000134
SectionMaxRawsize            -0.000790
BaseOfData                   -0.001136
MinorSubsystemVersion        -0.001213
SectionMaxVirtualsize        -0.001332
MinorOperatingSystemVersion  -0.001702
ResourcesMinSize             -0.001774
SectionAlignment             -0.002429
SizeOfHeapCommit             -0.002506
SizeOfImage                  -0.002603
Name: legitimate, dtype: float64
```

Dataset 3

```
label                          1.000000
forward_packet_length_mean     0.255970
total_forward_packets_length   0.139783
total_forward_packets          0.131577
forward_packets_per_second     0.075135
flow_packets_per_seconds       0.075101
flow_bytes_per_seconds         0.061670
flow_iat_mean                 -0.043882
forward_iat_mean              -0.049511
backward_iat_mean             -0.049928
flow_duration                 -0.055195
total_backward_packets_length -0.876010
backward_packet_length_mean   -0.876524
backward_packets_per_second   -0.953679
total_backward_packets        -0.993116
Name: label, dtype: float64
```

Step 13

Next the .drop function was used to drop certain labels from rows or columns. This can be done by indicating certain label names and corresponding axis or by a direct index or column name. Labels that are different levels are removed when a level is specified by using multi-indexing. (W3Resources. (2020)).

Dataset 1

```
x=data1.drop(["hash","classification",'vm_truncate_count','shared_vm','exec_vm','nvcsw','maj_flt','utime'],axis=1)
x.head()
```

	millisecond	state	usage_counter	prio	static_prio	normal_prio	policy	vm_pgoff	task_size	cached_hole_size	...	end_data	last_interval	nivcsw	r
0	0	0	0	3069378560	14274	0	0	0	0	0	...	120	3473	0	
1	1	0	0	3069378560	14274	0	0	0	0	0	...	120	3473	0	
2	2	0	0	3069378560	14274	0	0	0	0	0	...	120	3473	0	
3	3	0	0	3069378560	14274	0	0	0	0	0	...	120	3473	0	
4	4	0	0	3069378560	14274	0	0	0	0	0	...	120	3473	0	

5 rows × 27 columns

Dataset 2

```
x=data1.drop(["Name","md5",'Machine','MinorLinkerVersion','SizeOfUninitializedData','legitimate'],axis=1)
x.head()
```

	SizeOfOptionalHeader	Characteristics	MajorLinkerVersion	SizeOfCode	SizeOfInitializedData	AddressOfEntryPoint	BaseOfCode	BaseOfData	ImageBase	Sec
0	224	258	9	361984	115712	6135	4096	372736	4194304	
1	224	3330	9	130560	19968	81778	4096	143360	771751936	
2	224	3330	9	517120	621568	350896	4096	811008	771751936	
3	224	258	9	585728	369152	451258	4096	798720	771751936	
4	224	258	9	294912	247296	217381	4096	536576	771751936	

5 rows × 51 columns

Dataset 3

```
x=data1.drop(["protocol","flow_duration",'forward_iat_mean','backward_iat_mean','flow_packets_per_seconds','flow_bytes_per_second
x.head()
```

	total_forward_packets	total_backward_packets	total_forward_packets_length	total_backward_packets_length	forward_packet_length_mean	backward_packet_le
0	4	0	1580.0	0.0	395.0	
1	4	0	5888.0	0.0	1472.0	
2	200	0	88000.0	0.0	440.0	
3	200	0	88000.0	0.0	440.0	
4	2	0	2062.0	0.0	1031.0	

Step 14

Next the data that was changed from a string to an integer and then was printed out.

Dataset 1

```
y=data1["classification"]
y
```

```
0        1
1        1
2        1
3        1
4        1
        ..
99995    1
99996    1
99997    1
99998    1
99999    1
Name: classification, Length: 100000, dtype: int64
```

Dataset 2

```
y=data1["legitimate"]
y
```

```
0         1
1         1
2         1
3         1
4         1
         ..
138042    0
138043    0
138044    0
138045    0
138046    0
Name: legitimate, Length: 138047, dtype: int64
```

Dataset 3

```
y=data1["label"]
y
```

```
0        1
1        1
2        1
3        1
4        1
        ..
33920    1
33921    1
33922    1
33923    1
33924    1
Name: label, Length: 33925, dtype: int64
```

Step 15

Next the train_test_split and GaussianNB libraries were imported

```
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
```

Step 16

The train-test test split function was performed. This is used to evaluate a machine learning algorithms performance. It will take a dataset and separate it into two subnets. The first one will be used to fit the model and is known as the training dataset. For the second it is used to provide the input elements of the dataset to the model. Once this is done predictions will be made and then will be compared with the values that were expected. It is known as the test dataset.

The main aim is to work out the performance of a model on data that is new. Not the data that is being used to train the model. (Brownlee, J. (2020)).

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=1)
```

Step 17

Next GaussianNB was imported from sklearn.naive_bayes, then instantiate it. I then used x_train and y_train to fit the model.

Naïve Bayes is a classification model that comes from the Bayes Theorem. Bayes' theorem describes the probability of an event A taking place given that another event B has already occurred. (Sharma, P. (2021)).

The formula for is it:

$$P(A|B) = \frac{P(B|A).P(A)}{P(B)}$$

(Sharma, P. (2021)).

- This classifier will go on the assumption that the predictor can contribute equally and independently in the selection of the output class.

- The Naïve Bayes will go off on the assumption that all of the predictors are independent, but this is not the case in real-world circumstances. The outcome from these assumptions are satisfactory in a lot of the instances.
- It is used for text categorization on occasion because the dimensionality of the data is larger. (Sharma, P. (2021)).

When predictor values are constant and are expected to follow a Gaussian distributor, this is when Gaussian Naïve Bayes will be employed. (Sharma, P. (2021)).

```
from sklearn.naive_bayes import GaussianNB
model=GaussianNB()
model.fit(x_train,y_train)
```

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

Step 18

Next the model.predict was used. This function will accept only the tested data the is a single argument. The returned labels that are within the data will then be passed as an argument depending on the learned or trained data taken from the model. This function works on top of the trained model and will use the learned labels to map and predict the labels that need to be tested. (AskPython. (2022)).

Model 1

```
pred=model.predict(x_test)
pred
```

```
array([1, 1, 1, ..., 1, 0, 1], dtype=int64)
```

Model 2

```
pred=model.predict(x_test)
pred
```

```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

Model 3

```
pred=model.predict(x_test)
pred
```

```
array([1, 1, 1, ..., 1, 1, 1], dtype=int64)
```

Step 19

Next the model.score function was used to find the accuracy of the model. As shown below dataset 3 is the most accurate.

Model 1

```
model.score(x_test,y_test)
```

```
0.6274
```

Model 2

```
model.score(x_test,y_test)
```

```
0.6971628636967282
```

Model 3

```
model.score(x_test,y_test)
```

```
0.998919237571232
```

Step 20

Lasty result=pd.DataFrame was used to show the actual and predicted values. Data frames are used to store data in a grid that can be easily viewed. Each row of the grids are correspondent to measurements or values of an instance. Each column is a vector that includes data for a certain type of variable. This means the rows within a dataframe do not need to contain the same values, but they can. The values can be numerical character, logical, etc. (Willems, K. (2019)).

Dataset 1

```
result=pd.DataFrame({
    "Actual_Value":y_test,
    "Predict_Value":pred
})
```

```
result
```

	Actual_Value	Predict_Value
43660	0	1
87278	1	1
14317	0	1
81932	1	1
95321	1	1
...	...	...
994	1	1
42287	0	1
4967	0	1
47725	0	0
42348	0	1

30000 rows × 2 columns

Dataset 2

	Actual_Value	Predict_Value
105842	0	0
126796	0	0
109391	0	0
49135	0	0
118758	0	0
...	...	...
17796	1	0
6384	1	0
59758	0	0
61644	0	0
68972	0	0

41415 rows × 2 columns

Dataset 3

	Actual_Value	Predict_Value
3785	1	1
33856	1	1
26780	1	1
2051	1	1
20659	1	1
...	...	...
22591	1	1
21078	1	1
17672	1	1
27557	1	1
2267	1	1

10178 rows × 2 columns

(SARAVANA , N. (2018)).

Model 3

Step 1

The libraries that were needed were imported in this first step.

```
import pandas as pd
```

Step 2

Next the .drop was used to drop specific malware or legitimate labels. The dataset was imported and then the .head() function was used to print out the dataset.

```
malware_csv=pd.read_csv("Malware dataset.csv")
legit = malware_csv[0:41323].drop(['classification'],axis=1)
malware = malware_csv[41323::].drop(['classification'],axis=1)
malware_csv.head()
```

	hash	millisecond	classification	state	usage_counter	prio	static_prio	normal_prio	policy	vm_pgoff	.
0	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	0	malware	0	0	3069378560	14274	0	0	0	.
1	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	1	malware	0	0	3069378560	14274	0	0	0	.
2	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	2	malware	0	0	3069378560	14274	0	0	0	.
3	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	3	malware	0	0	3069378560	14274	0	0	0	.
4	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	4	malware	0	0	3069378560	14274	0	0	0	.

5 rows × 35 columns

Step 3

The print function was used to print the number of samples and features of the malware and legitimate files. The .shape() function was used to return a tuple that signifies the dimensions of any python object whenever it applies. The objects are usually numpy.array or pandas.DataFrame. The amount of elements within the tuple returned by the shape method will be equal to the amount of dimensions within the python object. Each element that is corresponding to the dimensions of the python object is represented by the tuple elements. (AskPython. (2022)).

Dataset 1

```
print("The no of samples are %s and no of features are %s for legitimate part "%(legit.shape[0],legit.shape[1]))
print("The no of samples are %s and no of features are %s for malware part "%(malware.shape[0],malware.shape[1]))
```

```
The no of samples are 41323 and no of features are 34 for legitimate part
The no of samples are 58677 and no of features are 34 for malware part
```

Dataset 2

```
print("The no of samples are %s and no of features are %s for legitimate part "%(legit.shape[0],legit.shape[1]))
print("The no of samples are %s and no of features are %s for malware part "%(malware.shape[0],malware.shape[1]))
```

```
The no of samples are 41323 and no of features are 56 for legitimate part
The no of samples are 96724 and no of features are 56 for malware part
```

Dataset 3

```
print("The no of samples are %s and no of features are %s for legitimate part "%(legit.shape[0],legit.shape[1]))
print("The no of samples are %s and no of features are %s for malware part "%(malware.shape[0],malware.shape[1]))
```

```
The no of samples are 33925 and no of features are 15 for legitimate part
The no of samples are 0 and no of features are 15 for malware part
```

Step 4

Next set_options was used which takes two arguments and sets them to parameter. It can also be used to change the amount of rows or columns displayed. (Point, T. (2022)). As shown below I set it to the max amount of columns.

Dataset 1

```
pd.set_option("display.max_columns",None)
malware
```

	hash	millisecond	state	usage_counter	prio	static_prio	normal_prio	policy	vm_pgoff	vm_trunc
41323	1dec265aeda7b58e4173f47af0641a949937edbf21904f...	323	0	0	3069526016	22281	0	0	0	
41324	1dec265aeda7b58e4173f47af0641a949937edbf21904f...	324	0	0	3069526016	22281	0	0	0	
41325	1dec265aeda7b58e4173f47af0641a949937edbf21904f...	325	0	0	3069526016	22281	0	0	0	
41326	1dec265aeda7b58e4173f47af0641a949937edbf21904f...	326	0	0	3069526016	22281	0	0	0	
41327	1dec265aeda7b58e4173f47af0641a949937edbf21904f...	327	0	0	3069526016	22281	0	0	0	
...	...	...	...	...	...	...	...	...	...	
99995	025c63d266e05d9e3bd57dd9ebd0abe904616f569fe4e2...	995	4096	0	3070148608	13988	0	0	0	
99996	025c63d266e05d9e3bd57dd9ebd0abe904616f569fe4e2...	996	4096	0	3070148608	13988	0	0	0	
99997	025c63d266e05d9e3bd57dd9ebd0abe904616f569fe4e2...	997	4096	0	3070148608	13988	0	0	0	
99998	025c63d266e05d9e3bd57dd9ebd0abe904616f569fe4e2...	998	4096	0	3070148608	13988	0	0	0	
99999	025c63d266e05d9e3bd57dd9ebd0abe904616f569fe4e2...	999	4096	0	3070148608	13988	0	0	0	

58677 rows × 34 columns

Dataset 2

	Name	md5	Machine	SizeOfOptionalHeader	Characteristics	MajorLinkerVersion
41323	VirusShare_4a400b747afe6547e09ce0b02dae7f1c	4a400b747afe6547e09ce0b02dae7f1c	332	224	258	11
41324	VirusShare_9bd57c8252948bd2fa651ad372bd4f13	9bd57c8252948bd2fa651ad372bd4f13	332	224	271	6
41325	VirusShare_d1456165e9358b8f61f93a5f2042f39c	d1456165e9358b8f61f93a5f2042f39c	332	224	258	10
41326	VirusShare_e4214cc73afbba0f52bb72d5db8f8bb1	e4214cc73afbba0f52bb72d5db8f8bb1	332	224	258	10
41327	VirusShare_710890c07b3f93b90635f8bff6c34605	710890c07b3f93b90635f8bff6c34605	332	224	258	9
...	...	...	...	...	...	...
138042	VirusShare_8e292b418568d6e7b87f2a32aee7074b	8e292b418568d6e7b87f2a32aee7074b	332	224	258	11
138043	VirusShare_260d9e2258aed4c8a3bbd703ec895822	260d9e2258aed4c8a3bbd703ec895822	332	224	33167	2
138044	VirusShare_8d088a51b7d225c9f5d11d239791ec3f	8d088a51b7d225c9f5d11d239791ec3f	332	224	258	10
138045	VirusShare_4286dccf67ca220fe67635388229a9f3	4286dccf67ca220fe67635388229a9f3	332	224	33166	2
138046	VirusShare_d7648eae45f09b3adb75127f43be6d11	d7648eae45f09b3adb75127f43be6d11	332	224	258	11

96724 rows × 56 columns

Dataset 3

In this dataset there seems to be no maximum columns, so no values were displayed.

```
pd.set_option("display.max_columns",None)
malware
```

protocol	flow_duration	total_forward_packets	total_backward_packets	total_forward_packets_length	total_backward_packets_length	forward_packet_length_me

Step 5

Next the libraries that are important and that are used for train and test splitting, extra tree classifier, select from model and cross validate were imported.

```
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_validate
```

Step 6

Next extra tree classifier was used. This is a type of technique that will combine multiple de-correlated decision tree results that have been collected in a “forest” to put the results of the classification. It is fairly similar to Random Forest Classifier; the only difference is the manner of construction of the decision tree. Each tree is built from the training samples. A random sample of k features will be provided to each tree at the test nodes. Each tree will then select the best feature to split the data, depending on mathematical criteria. These features will lead to the creation of several de-correlated decision trees. (Gupta A. (2020)).

Select.transfom function was also used. The transform function is used to return a dataframe that is self-produced with transformed value once the specific function parameter has been applied. The dataframe is the same in length as the dataframe that is passed. (Sethi, A. (2020)).

A .drop function was used to drop the columns that were not needed.

```
data_input = malware_csv.drop(['hash','millisecond','classification'],axis = 1).values
labels = malware_csv['classification'].values
extratrees = ExtraTreesClassifier().fit(data_input, labels)
select = SelectFromModel(extratrees, prefit = True)
data_input_new = select.transform(data_input)
```

Step 7

During this next step a feature_importance function was used. This is a class of techniques that are used to assign input feature scores to a predictive model. This will indicate how important each feature is when it is making predictions. The scores can be calculated with regression. What this does is take the problems that involve predicting a numerical value. The problems are called classification.

These scores can be used for a variety of situations in predictive modelling problems:

- Better understanding the data.
- Better understanding a model.
- Reducing the number of input features.

The scores can show which feature are more important to the target and the converse and which are less important. A domain expert can used to interpret this. And can be used to gather more of different data. (Brownlee, J. (2022)).

Dataset 1

```
import numpy as np
features = data_input_new.shape[1]
importances = extratrees.feature_importances_
indices = np.argsort(importances)[::-1]
for i in range (features):
    print("%d"%(i+1),malware_csv.columns[2+indices[i]],importances[indices[i]])
```

```
1 prio 0.15787507610330162
2 last_interval 0.09333851810082468
3 lock 0.0895140493457294
4 shared_vm 0.06832125437554125
5 min_flt 0.0625510173846741
6 vm_pgoff 0.057218167169379
7 total_vm 0.05691091274217622
8 nr_ptes 0.054907549951913125
9 cached_hole_size 0.04600650214397335
10 usage_counter 0.04188719534754644
11 free_area_cache 0.04057343440452844
12 mm_users 0.036130299432872104
13 end_data 0.033097720102113336
14 nvcsw 0.03242174527948294
```

Dataset 2

```
import numpy as np
features = data_input_new.shape[1]
importances = extratrees.feature_importances_
indices = np.argsort(importances)[::-1]
for i in range (features):
    print("%d"%(i+1),malware_csv.columns[2+indices[i]],importances[indices[i]])
```

```
1 DllCharacteristics 0.15005552268849384
2 Characteristics 0.12319505783309875
3 Machine 0.10059803258842327
4 VersionInformationSize 0.07967749195541136
5 Subsystem 0.057761460491253185
6 MajorSubsystemVersion 0.0574471585665799
7 SectionsMaxEntropy 0.05161771152723265
8 ImageBase 0.04820180455388697
9 ResourcesMaxEntropy 0.04265970135844566
10 SizeOfOptionalHeader 0.03753706251275174
11 ResourcesMinEntropy 0.03223518965526969
12 MajorOperatingSystemVersion 0.023146102298726032
13 SectionsMinEntropy 0.019453334960242496
```

Dataset 3

```
1 backward_packets_per_second 0.3948806276336016
2 total_backward_packets 0.24206450205738392
3 total_backward_packets_length 0.17130153384261393
4 backward_packet_length_mean 0.1529861783881282
```

Step 8

Next a test, train split was performed and then a random forest classifier and n_elimiators was used.

N_elimators is the amount of trees that someone is wanting to build before taking maximum votes or average of the predictions. If the amount of trees are higher it will give a better performance, but the code will be a lot slower. Choosing a higher value makes predictions stronger and a lot more stable. (Tavish. (2015)).

A .fit method was also used. This is used to estimators some of the parameters in the model. It takes the training data ad arguments. If its unsupervised learning it can be one array and two arrays for supervised learning. (Learn, S. (2022)).

It is fitted using x and y (or in my model legit and mal) and does not hold any reference to x and y in the object. In the case of precomputed kernels, it will make an exception for this and will store the data by using the prediction method. X.shape[0] and y.shape[0] must be the same. A value error will pop up if the requisite is not met. In the case of unsupervised learning y may be ignored. When supervised and unsupervised learning are mixed together this makes it possible for estimators to be part of a pipeline. In the second position, even unsupervised estimators must accept y=None, otherwise the estimator will ignore it. It also should return the object (self). It is good for implementing one liners in an IPython session. (Learn, S. (2022)).

```
from sklearn.ensemble import RandomForestClassifier
legit_train,legit_test,mal_train,mal_test = train_test_split(data_input_new,labels,test_size=0.2)

classifier = RandomForestClassifier(n_estimators=50)
classifier.fit(legit_train,mal_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=50,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

Step 9

During this step the .score() algorithm was used to calculate the score of the algorithm.

Dataset 1

```
print("The score of algorithm is " + str(classifier.score(legit_test,mal_test)*100))
```

```
The score of algorithm is 100.0
```

Dataset 2

```
print("The score of algorithm is " + str(classifier.score(legit_test,mal_test)*100))
```

```
The score of algorithm is 99.35530604853314
```

Dataset 3

```
print("The score of algorithm is " + str(classifier.score(legit_test,mal_test)*100))
```

```
The score of algorithm is 99.9852616064849
```

Step 10

Next classifier.predict was used to predict the confusion matrix. A confusion matrix is a performance measurement for classification problems. Outputs can be two or more classes. It can be very helpful when measuring Recall, Precision, Specificity, Accuracy ad AUC_ROC curves. (Narkhede, S. (2018)).

Next the confusion matrix labels were printed out.

Dataset 1

```
from sklearn.metrics import confusion_matrix
result = classifier.predict(legit_test)
conf_matrix = confusion_matrix(mal_test,result)
```

```
conf_matrix
```

```
array([[ 9915,     0],
       [    0, 10085]], dtype=int64)
```

Dataset 2

```
conf_matrix
```

```
array([[19186,   106],
       [   72,  8246]], dtype=int64)
```

Dataset 3

```
from sklearn.metrics import confusion_matrix
result = classifier.predict(legit_test)
conf_matrix = confusion_matrix(mal_test,result)
```

```
conf_matrix
```

```
array([[ 221,    1],
       [   0, 6563]], dtype=int64)
```

Step 11

Next conf_matrix was used to calculate the false positive and false negative, the same as one of the previous models.

Dataset 1

```
print("False Positives:",conf_matrix[0][1]*100/sum(conf_matrix[0]))
print("False Negatives:",conf_matrix[1][0]*100/sum(conf_matrix[1]))
```

```
False Positives: 0.0
False Negatives: 0.0
```

Dataset 2

```
False Positives: 0.5494505494505495
False Negatives: 0.8655926905506132
```

Dataset 3

```
False Positives: 0.45045045045045046
False Negatives: 0.0
```

Step 12

During this step a gradient booster was used. This is mainly used for structured predictive modelling problems, for example classification and regression that are on tabular data. It is the primary algorithm used in winning solutions to machine learning competitors. Decision tree models will construct ensembles. The tress will be added one at a time to these ensembles and are fitted to correct any prediction errors that are made by the previous model. This is known as boosting.

Arbitrary differentiable loss function and gradient descent optimization algorithm will be fitted to the models. This is how the technique got its name *gradient boosting.* Just like a neural network, the loss gradient is minimized as the model is fit. (Brownlee, J. (2020)).

It provides hyperparameters that will be tuned for a specific dataset. The most important ones are:

- The amount of trees or estimators that are in a model.
- The rate at which the model learns.
- The sampling rate for rows and columns for stochastic models.
- The maximum depth of the tree
- The minimum weight of the tress.
- Regularization terms alpha and lambda. (Brownlee, J. (2020)).

Dataset 1

```
from sklearn.ensemble import GradientBoostingClassifier
grad_boost = GradientBoostingClassifier(n_estimators=50)
grad_boost.fit(legit_train,mal_train)
```

```
GradientBoostingClassifier(n_estimators=50)
```

Dataset 2

```
from sklearn.ensemble import GradientBoostingClassifier
grad_boost = GradientBoostingClassifier(n_estimators=50)
grad_boost.fit(legit_train,mal_train)
```

```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=50,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

Dataset 3

```
: from sklearn.ensemble import GradientBoostingClassifier
  grad_boost = GradientBoostingClassifier(n_estimators=50)
  grad_boost.fit(legit_train,mal_train)
```

```
: GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                             learning_rate=0.1, loss='deviance', max_depth=3,
                             max_features=None, max_leaf_nodes=None,
                             min_impurity_decrease=0.0, min_impurity_split=None,
                             min_samples_leaf=1, min_samples_split=2,
                             min_weight_fraction_leaf=0.0, n_estimators=50,
                             n_iter_no_change=None, presort='deprecated',
                             random_state=None, subsample=1.0, tol=0.0001,
                             validation_fraction=0.1, verbose=0,
                             warm_start=False)
```

Step 13

Next the gradient boost score was printed. As shown below the accuracy of all three datasets are similar.

Dataset 1

```
print("Score:", grad_boost.score(legit_test,mal_test)*100)
```

```
Score: 99.615
```

Dataset 2

```
print("Score:", grad_boost.score(legit_test,mal_test)*100)
```

```
Score: 98.7721839913075
```

Dataset 3

```
print("Score:", grad_boost.score(legit_test,mal_test)*100)
```

```
Score: 99.9852616064849
```

(Codes7, L. (2021)).

Tools Used

Juptyter notebook was used during this project. This is an open-source application that is used to create and share documents. These consist of live code, equations, visualisation, and text. (Driscoll, M. (2019)).

Anaconda 3 was downloaded because this includes jupyter notebook which means it does not need to be installed through the command prompt. Once installed, there was a section to open jupyter notebook in the browser and all that was needed to do was locate the file that the code was in.

Self-Appraisal / Self Reflection

During this project I decided to compare malware machine learning models against three detection dataset, two are malware attacks and one is a dos attack. I searched for weeks through GitHub to find machine learning models that would work with all of my datasets. One of the main problems I kept having was not being able to convert string to float. After searching for a while to see if there was any solution and trying different things, nothing worked. I tried to drop the columns that would not work but the error message kept saying cannot find in the array. After several weeks of trying to fix this problem I decided to try and find other models that would work. After some more research I found models that worked with all of my datasets. What I have learnt from this is that not all problem can be solved, and some datasets are just not meant to work for certain models.

Results

The outcome of this project shows the accuracy of each dataset. From this I then decided which model is the best and most accurate for malware detection. After reviewing all of the models, it has been decided that model one is the most accurate because it has the highest accuracy score when I run all three datasets in it whether it is a large difference or a small difference. Below I will show all of the accuracy results for three models and the three datasets. The second-best model is model three as it has the second highest accuracy score and the least accurate is model two.

Model 1:

Dataset 1

```
In [20]: print(forest.score(X_test, y_test))
         1.0
```

Dataset 2

```
In [25]: print(forest.score(X_test, y_test))
         0.9949293734154292
```

Dataset 3

```
In [16]: print(forest.score(X_test, y_test))
         0.999852616064849
```

Model 2:

Dataset 1

```
model.score(x_test,y_test)
0.6274
```

Dataset 2

```
model.score(x_test,y_test)
0.6971628636967282
```

Dataset 3

```
model.score(x_test,y_test)
```

```
0.998919237571232
```

Model 3:

Dataset 1

```
print("Score:", grad_boost.score(legit_test,mal_test)*100)
```

```
Score: 99.615
```

Dataset 2

```
print("Score:", grad_boost.score(legit_test,mal_test)*100)
```

```
Score: 98.7721839913075
```

Dataset 3

```
print("Score:", grad_boost.score(legit_test,mal_test)*100)
```

```
Score: 99.9852616064849
```

Research Hypothesis

Several datasets exist for use in the detection of malware, and it is commonly assumed that all work equally well. Using three machine learning models the research project intends to prove that there are differences in performance between three of these sets in relation to malware detection.

Reference

ALO, U - NWEKE, H - ELE, S. (2021). *MACHINE LEARNING-BASED FRAMEWORK FOR AUTOMATIC MALWARE DETECTION USING ANDROID TRAFFIC DATA.* Available: http://www.jatit.org/volumes/Vol99No15/10Vol99No15.pdf. Last accessed 18/03/2022.

AskPython. (2022). *Python predict() function – All you need to know!.* Available: https://www.askpython.com/python/examples/python-predict-function. Last accessed 11/03/2022.

AskPython. (2022). *Python shape method: Identify the dimensions of a Python object.* Available: https://www.askpython.com/python-modules/pandas/shape-methodc. Last accessed 11/03/2022.

Brownlee, J. (2016). *What is a Confusion Matrix in Machine Learning.* Available: https://machinelearningmastery.com/confusion-matrix-machine-learning/#:~:text=A%20confusion%20matrix%20is%20a%20summary%20of%20prediction%20results%20on,key%20to%20the%20confusion%20matrix.. Last accessed 04/03/2022.

Brownlee, J. (2020). *Gradient Boosting with Scikit-Learn, XGBoost, LightGBM, and CatBoost.* Available: https://machinelearningmastery.com/gradient-boosting-with-scikit-learn-xgboost-lightgbm-and-catboost/. Last accessed 17/03/2022.

Brownlee, J. (2020). *Train-Test Split for Evaluating Machine Learning Algorithms.* Available: https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/. Last accessed 09/03/2022.

Brownlee, J. (2022). *How to Calculate Feature Importance With Python.* Available: https://machinelearningmastery.com/calculate-feature-importance-with-python/. Last accessed 15/03/2022.

Camp, D. (2018). *Pickle in Python Tutorial: Object Serialization.* Available: https://www.datacamp.com/community/tutorials/pickle-python-tutorial. Last accessed 06/03/2022.

Christopher, A. (2021). *Random Forest.* Available: https://medium.com/analytics-vidhya/random-forest-4a2981aab4f7. Last accessed 04/03/2022.

Codes7, L. (2021). *Malware-Detection using scikit-learn.* Available: https://github.com/Sameer411/Malware-Detection-Using-Machine-Learning/blob/main/Rishab%20Mudliar%20Code/Malware-Detection.ipynb. Last accessed 01/04/2022.

Cyware. (2021). *The Evolution of Threat Intelligence.* Available: https://cyware.com/educational-guides/cyber-threat-intelligence/the-evolution-of-threat-intelligence-fdba. Last accessed 27/03/2022.

Driscoll, M. (2019). *Jupyter Notebook: An Introduction.* Available: https://realpython.com/jupyter-notebook-introduction/. Last accessed 24/03/2022.

Ebner, J. (2021). *Pandas isnull, Explained.* Available: https://www.sharpsightlabs.com/blog/pandas-isnull/. Last accessed 07/03/2022.

Fish, D. (2020). *Example of Random Forest in Python.* Available: https://datatofish.com/random-forest-python/. Last accessed 04/03/2022.

Gupta A. (2020). *ML | Extra Tree Classifier for Feature Selection.* Available: https://www.geeksforgeeks.org/ml-extra-tree-classifier-for-feature-selection/. Last accessed 12/03/2022.

Hjorth-Jensen, M. (2021). *Decisions Trees, Random Forests, Bagging and Boosting.* Available: https://compphysics.github.io/MachineLearning/doc/pub/week45/html/week45.html. Last accessed 04/03/2022.

Learn, S. (2022). *Developing scikit-learn estimators.* Available: https://scikit-learn.org/stable/developers/develop.html. Last accessed 15/03/2022.

Macrae, A. (2020). *I, CyBOK – An Introduction to the Cyber Security Body of Knowledge Project.* Available: https://www.tripwire.com/state-of-security/security-data-protection/icybok-introduction-cybersecurity-body-knowledge-project/. Last accessed 31/03/2022.

Malik, U. (2019). *Advantages and Disadvantages of the Python Programming Language.* Available: https://learnpython.com/blog/python-programming-advantages-disadvantages/. Last accessed 31/03/2022.

Narkhede, S. (2018). *Understanding Confusion Matrix.* Available: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62. Last accessed 15/03/2022.

Nixus. (2022). *Advantages of Machine Learning | Disadvantages of Machine Learning.* Available: https://nixustechnologies.com/advantages-disadvantages-of-machine-learning/. Last accessed 22/03/2022.

Pandio. (2021). *A Brief History of Machine Learning.* Available: https://pandio.com/blog/when-was-machine-learning-invented/. Last accessed 26/03/2022.

Point, T. (2022). *Python Pandas - Options and Customization.* Available: https://www.tutorialspoint.com/python_pandas/python_pandas_options_and_customization.htm. Last accessed 12/03/2022.

Pramanick, S. (2022). *History of Python.* Available: https://www.geeksforgeeks.org/history-of-python/. Last accessed 26/03/2022.

premnagdeo . (2020). *Malware Detection.* Available: https://github.com/premnagdeo/Malware-Detection/blob/master/Malware-Detection.ipynb. Last accessed 01/04/2022.

Pythontic. (2020). *Pickle.dumps() Method In Python.* Available: https://pythontic.com/modules/pickle/dumps#:~:text=The%20dumps()%20method%20of,object%20of%20the%20serialized%20object.. Last accessed 06/03/2022.

Pythontic. (2020). *Pickle.load() Method In Python.* Available: https://pythontic.com/modules/pickle/load. Last accessed 06/03/2022.

Roohparvar, R. (2021). *THE BENEFITS OF CYBER THREAT INTELLIGENCE FOR YOUR ORGANIZATION.* Available: https://www.infoguardsecurity.com/the-benefits-of-cyber-threat-intelligence-for-your-organization/. Last accessed 31/03/2022.

Rose, S. (2019). *Advantages of Using Python for Machine Learning.* Available: https://medium.com/@scarlett8285/advantages-of-using-python-for-machine-learning-8d2093697e8f. Last accessed 31/03/2022.

SARAVANA , N. (2018). *Malware Detection Using Naive Bayes.* Available: https://www.kaggle.com/code/nsaravana/malware-detection-using-naive-bayes/notebook. Last accessed 01/04/2022.

Scarfone, K. (2020). *AI threat intelligence is the future, and the future is now.* Available: https://www.techtarget.com/searchsecurity/tip/AI-threat-intelligence-is-the-future-and-the-future-is-now. Last accessed 21/03/2022.

Sethi, A. (2020). *Learn How to use the Transform Function in Pandas.* Available: https://www.analyticsvidhya.com/blog/2020/03/understanding-transform-function-python/. Last accessed 12/03/2022.

Sharma, P. (2021). *Implementation of Gaussian Naive Bayes in Python Sklearn.* Available: https://www.analyticsvidhya.com/blog/2021/11/implementation-of-gaussian-naive-bayes-in-python-sklearn/. Last accessed 11/03/2022.

Souri, A & Hosseini, R. (2018). *A state-of-the-art survey of malware detection approaches using data mining techniques.* Available: https://hcis-journal.springeropen.com/articles/10.1186/s13673-018-0125-x. Last accessed 17/03/2022.

Stojiljković, M. (2021). *Split Your Dataset With scikit-learn's train_test_split().* Available: https://realpython.com/train-test-split-python-data/. Last accessed 04/03/2022.

Tavish. (2015). *Tuning the parameters of your Random Forest model.* Available: https://www.analyticsvidhya.com/blog/2015/06/tuning-random-forest-model/. Last accessed 15/03/2022.

W3Resources. (2020). *Pandas DataFrame property: loc.* Available: https://www.w3resource.com/pandas/dataframe/dataframe-loc.php. Last accessed 08/03/2022.

W3Resources. (2020). *Pandas DataFrame: drop() function.* Available: https://www.w3resource.com/pandas/dataframe/dataframe-drop.php. Last accessed 09/03/2022.

W3Resources. (2020). *Pandas Series: tail() function.* Available: https://www.w3resource.com/pandas/series/series-tail.php#:~:text=The%20tail()%20function%20is,after%20sorting%20or%20appending%20rows.. Last accessed 07/03/2022.

Willems, K. (2019). *Pandas Tutorial: DataFrames in Python.* Available: https://www.datacamp.com/community/tutorials/pandas-tutorial-dataframe-python. Last accessed 11/03/2022.

Zhang, J. (2018). *5 Reasons Why Threat Intelligence Doesn't Work.* Available: https://www.darkreading.com/vulnerabilities-threats/5-reasons-why-threat-intelligence-doesn-t-work. Last accessed 31/03/2022.