CI601 Project Report

Creation of a browser extension to detect phishing URLs using machine learning

Ethan Floate

Student No. 20825578

# 

[Abstract 1](#_Toc1571959417)

[1 Introduction: 2](#_Toc801585750)

[1.1 Aims: 3](#_Toc2060241376)

[1.2 Requirements: 3](#_Toc2077014293)

[2 Methodology: 4](#_Toc1099223664)

[2.1 Chosen Approach: 4](#_Toc127217650)

[2.2 AGILE Project Management: 5](#_Toc1300433649)

[Sprint 1 5](#_Toc800794527)

[Sprint 2 6](#_Toc1080068238)

[Sprint 3 6](#_Toc913531528)

[Sprint 4 7](#_Toc743978889)

[Sprint 5 8](#_Toc1737428631)

[Sprint 6 8](#_Toc1945851262)

[Sprint 7 9](#_Toc1694489856)

[Sprint Backlog: 10](#_Toc681314172)

[2.3 Dev Tools: 10](#_Toc1705930776)

[2.4 Risk Analysis: 10](#_Toc1373121041)

[3 Research: 11](#_Toc1119728214)

[3.1 Round 1 (Sprint 1) 11](#_Toc1030508688)

[3.2 Round 2 (Sprint 4/6) 12](#_Toc933386772)

[3.3 Round 3 (Sprint 9) 13](#_Toc990031681)

[4 Product Description: 14](#_Toc2116046796)

[4.1 Data Processing: 14](#_Toc1728170278)

[4.1.1 Data Collection 14](#_Toc2044897228)

[Phishing URL Datasets 14](#_Toc822109739)

[Hybrid URL Dataset (Both legit and phish) 14](#_Toc1017590306)

[Legitimate URL Dataset 14](#_Toc1225099587)

[4.1.2 Data Pre-processing 14](#_Toc1231021433)

[Clean Legitimate Data 14](#_Toc715179017)

[Clean Hybrid/Phishing data 16](#_Toc2025039835)

[4.1.3 Feature Creation 18](#_Toc1850308853)

[Extract features from URLs (Legitimate Data): 18](#_Toc1868082436)

[Extract Features from URL (Hybrid/Phishing Data): 20](#_Toc1774330866)

[Extract Hyperlink Features (2nd round model): 20](#_Toc1916307019)

[Appending Hyperlink Data (Legit URLs) 23](#_Toc1082302181)

[Appending Suspicious Words Data (Legit URLs) 24](#_Toc1495590723)

[Combine and Clean Data 24](#_Toc1468453515)

[4.2 AI Model 26](#_Toc613917087)

[4.2.1 1st Round of Training 26](#_Toc1992869826)

[4.2.2 2nd Round of Training 30](#_Toc1040854234)

[4.2.3 Evaluation of Models 34](#_Toc955000000)

[1st Round Model 34](#_Toc1163437679)

[2nd Round Model 36](#_Toc1105227602)

[4.3 Flask App: 39](#_Toc1017052653)

[4.4 JavaScript App: 40](#_Toc804203487)

[4.5 Pop-Up: 42](#_Toc1559572050)

[4.6 Google Cloud Implementation: 43](#_Toc1411906958)

[4.7 Firefox Add-On: 43](#_Toc1140670241)

[4.8 Bug Fixes: 45](#_Toc856851617)

[4.9 Demo: 45](#_Toc2004508572)

[5 Critical Review: 45](#_Toc1004708394)

[5.1 Functional Requirements: 46](#_Toc456286520)

[Must (Priority 0): 46](#_Toc1886091895)

[High Priority (Priority 1): 46](#_Toc701789761)

[Medium Priority (Priority 2): 46](#_Toc1004087574)

[Low Priority (Priority 3): 47](#_Toc1419728124)

[5.2 Non-Functional Requirements: 47](#_Toc2121373557)

[Must (Priority 0): 47](#_Toc705944925)

[High Priority (Priority 1): 47](#_Toc1919886662)

[5.3 Reflection and Evaluation: 47](#_Toc1134922294)

[What I did well: 47](#_Toc464883774)

[How I could improve: 48](#_Toc1332958249)

[6 Conclusion 49](#_Toc137294553)

[Research & Literature Review: 49](#_Toc1666021227)

# Abstract

Phishing attacks pose a significant threat to cyber-security. In the following paper, the development of a pythonic AI model to detect phishing links is presented. We trained the browser extension using approximately 13 500 URLs and created a model with an accuracy of 80%. The extension is free and availableas a Firefox add-on here: <https://addons.mozilla.org/en-GB/firefox/addon/phisherman-protect/> . The add-on is still a work in progress, and there are areas where it could be improved. For instance, the number of false positives could be reduced, the accuracy of the model could be improved, and the pop-up warning system could be made more user-friendly and compliant with best practices. The results of this study suggest that AI models can be used to effectively detect phishing links and we believe that this extension can help make the internet a safer place by protecting users from phishing attacks.

# 1 Introduction:

## 1.1 Aims:

* Build a web extension to help detect phishing.
* Train an AI model to identify malicious URLs
* Interrupt web request when a malicious URL is detected by the extension, alert the user.
* URLs flagged as malicious will be added to a database, these can be manually checked by me.
* Use the database of malicious URLs to help improve the AI model, can also be used for signature-based detection

The aim of this project is to create a web service in the form of a web extension that uses a custom-trained AI model to aide in the detection of phishing attacks. Specifically, the application will detect malicious URLs, using a mixture of its AI model and signature-based detection. When the user makes a request to a URL that the application has flagged as potentially malicious, the extension interrupts the request and alerts the user to the possible malicious link.

Another aspect to the application is that it will store the URLs it detects as malicious and stores them in a database. These can then be manually analysed and then used for signature-based detection or be used as training data to help improve the accuracy of the model.

The web extension should also be aesthetically pleasant, and not interrupt the regular user-experience of surfing the web any more than it needs to. With a small pop-up notifying the user to the malicious link and explaining why the app flagged this link. Most importantly, it should be unconfusing. As a large target audience for this app is people who are not that computer literate, as they do not naturally know red flags to look for when navigating to a URL.

This web extension, when completed, will be available to download from Add-Ons for Firefox and the Chrome Web Store (ideally)

## 1.2 Requirements:

|  |  |  |
| --- | --- | --- |
| Requirement # | Requirement Detail | Priority (0 High – 3 Low) |
| FR1 | Train a Pythonic AI model to detect malicious URLs with at least 50% accuracy | 0 |
| Fr1.5 | Build a web extension that interrupts web requests and alerts the user to a malicious URL with at least 50% accuracy | 0 |
| FR1.75 | Build a web extension alerts the user to a malicious URL with at least 75% accuracy | 1 |
| FR2 | Build a database of malicious URLs | 1 |
| Fr2.5 | Use the database for signature-based detection | 2 |
| Fr2.75 | Use the database as training data to improve the accuracy (decrease Mean Absolute Error) of the AI model | 3 |
| FR3 | Return a true positive malicious URL and warn the user in under 4 seconds | 1 |
| Fr3.5 | Return a true positive malicious URL, add the URL to the database, and warn the user in under 4 seconds | 2 |
| Fr3.75 | Return a true positive malicious URL, add the URL to the database, and warn the user in under 3 seconds | 3 |
| NFR1 | Create a minimal and unconfusing UI | 1 |
| NFR2 | The web extension will be available as a Firefox Add-on | 0 |
| NFR2.5 | The web extension will be available to download as a Chrome Extension | 1 |

# 2 Methodology:

## 2.1 Chosen Approach:

* Inspired by advancements in AV, moved past exclusively signature-based defence, to now include AI
* Create my own AI model to detect cyber-attacks
* Phishing is the most common type of cyber-crime
* You can train an AI model to detect phish-y URLs
* Reasearch suggests SVM to be the most effective AI model for detecting phishing URLs
* Use python to create the AI model
* Use Kaggle + GitHub to source training/validation data
* Connect this AI model to a web service, serve it as a web extension
* AGILE planning, as I am experienced with it plus it has great flexibility

My goal is to create a web extension to help prevent phishing attacks. The are many ways to go about creating this type of web-extension (Browser extensions - mozilla: MDN, 2022). I took my inspiration from advancements in AV (Anti-Virus) software. Historically, AV software has detected malicious software, URLs, and devices via a technique known as signature-detection. This involves using an identifying piece of data, such as a hash sum, to label certain pieces of software etc. as malicious. Recently, improvements in AI research have led to AV using trained AI models to detect ransomware (Microsoft 365 Defender Threat Intelligence Team, 2021). I believe a similar technique can be used to detect malicious URLs.

I decided to use Python to create the AI model for several reasons. Firstly, I would be able to present my model, research, and testing together in a Jupyter Notebook. Additionally, Python is a well-established language for creating AI models, and contains the appropriate libraries to do so such as *SKLearn* (Pedregosa, 2011). As well as this, I can use another Python library *Flask* to connect the model to a web service that can receive JSON style requests (Valley, 2019).

There are many different types of AI model, each suited for a different purpose. So, to maximise the effectiveness of my model, I would need to pick the correct type of AI. Thankfully, there are multiple papers that have been published around the subject of detecting malicious emails using an AI model. They concluded that an SVM (Support Vector Machine) model was the best suited to detect phishy emails, and I can conclude that they will also be effective against malicious URLs (Rawal et al., 2017). As the features used by the model to detect whether an email is malicious are very similar to the features, I will use to detect whether a URL is malicious.

In my research to find training and validation data, I looked at both GitHub and Kaggle. Kaggle is a very useful resource for finding datasets. In the end, I found a dataset that contained over 50,000 URLs, including possible feature parameters such as URL length or number of equals characters. After I cleaned the data set there were approximately 58,000 URLs to train and validate the model with, as well as a few data sets from Mendeley Data (*Mendeley Data*).

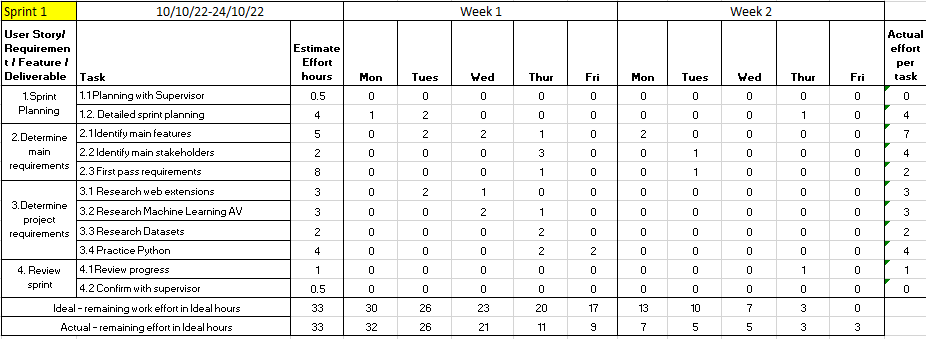
The functionality of the finished application should follow as such: a web extension acts as a man-in-the-middle inside the user’s browser, every time a web request is made the URL is passed into a parsing function which extracts the URL length and number of parameters. These numbers are then put into a REST style POST request to a Flask application that is connected to an AI model. After these features are passed to the model, a Boolean response is then given. 0 indicating the URL is benign and 1 indicating the possibility of the URL being malicious. In the case of a 0 response, the extension continues with the web request as normal, in the case of a 1 response, the man-in-the-middle (MITM) interrupts the request, presents the user with a pop-up warning them, and the URL is added to a database of other suspicious URLs.

In terms of project management, I decide to use the classic AGILE method (Noori, 2023). My reasoning behind this is that I have now used this project management technique with multiple projects, and each time I find it to be an effective way of managing time and effort into the top priority requirements of (in this case) the web application. I decided on 2-week sprints, again as this is a sprint length I have used before personally and found success with.

The final point in my approach is the decision to make this type of application in the first place, why focus on phishing attacks? Well, simply, because they are the most common type of cybercrime today, by a very large margin (Statista Research Department, 2022). By focussing on this type of attack, I hope to create a piece of software that is effective at preventing the most common type of cyber-crime.

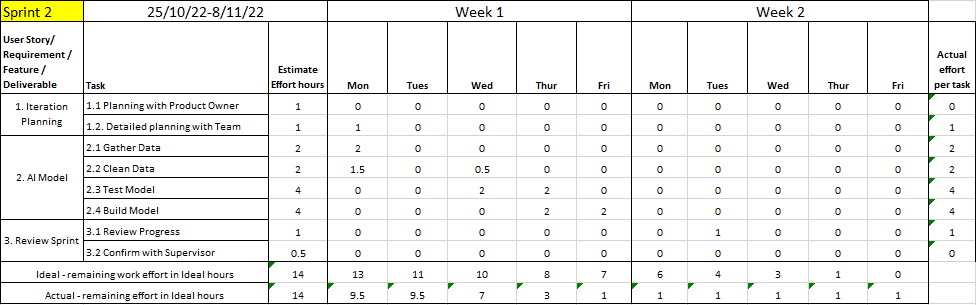
## 2.2 AGILE Project Management:

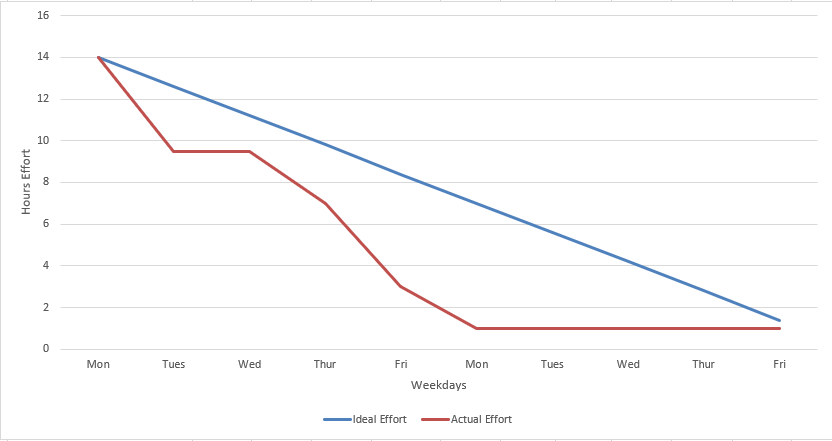
### Sprint 1



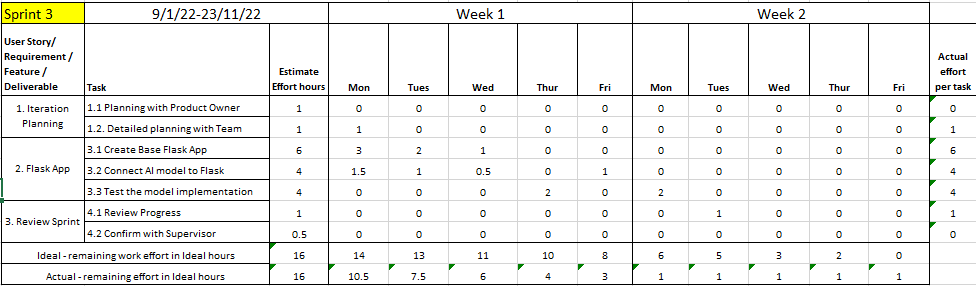
# 

### Sprint 2



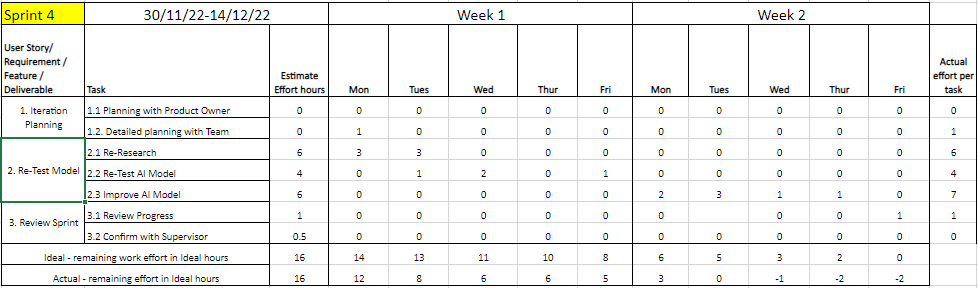


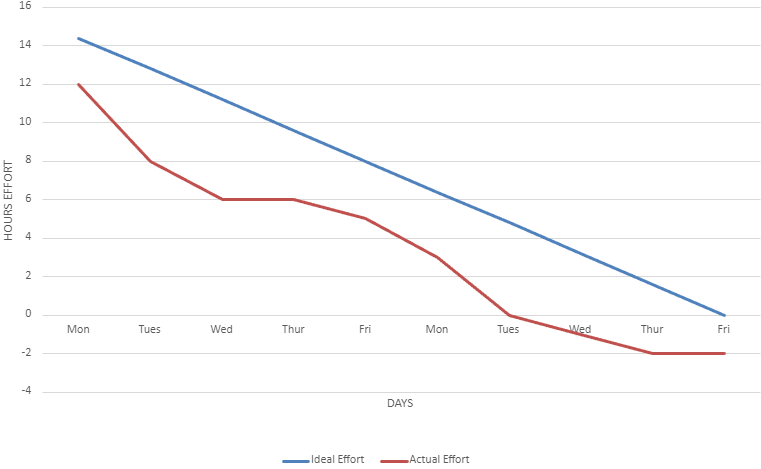
### Sprint 3



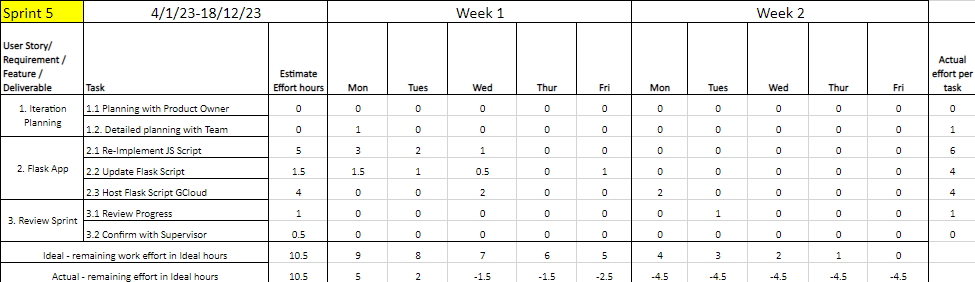
### 

### Sprint 4



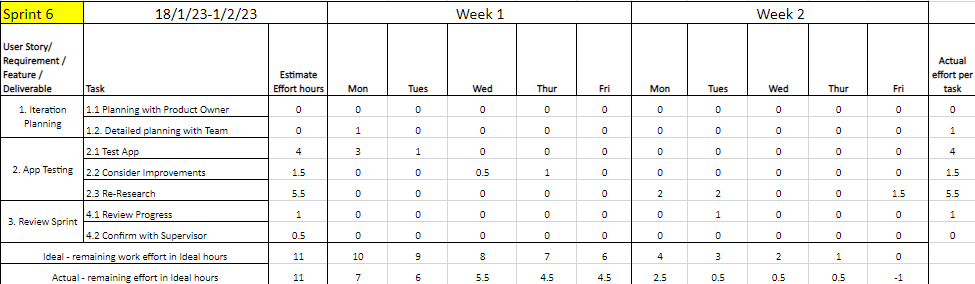


### Sprint 5



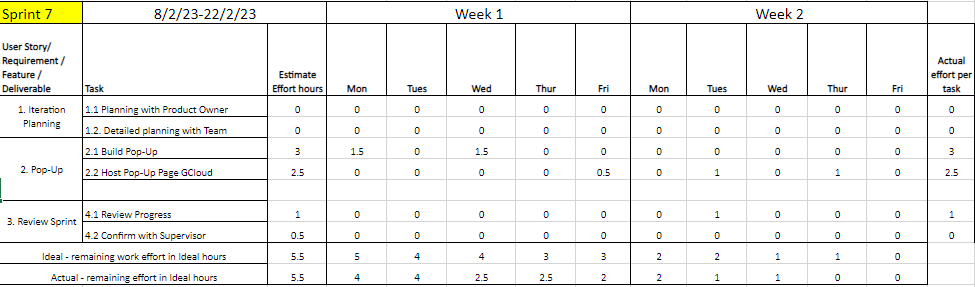


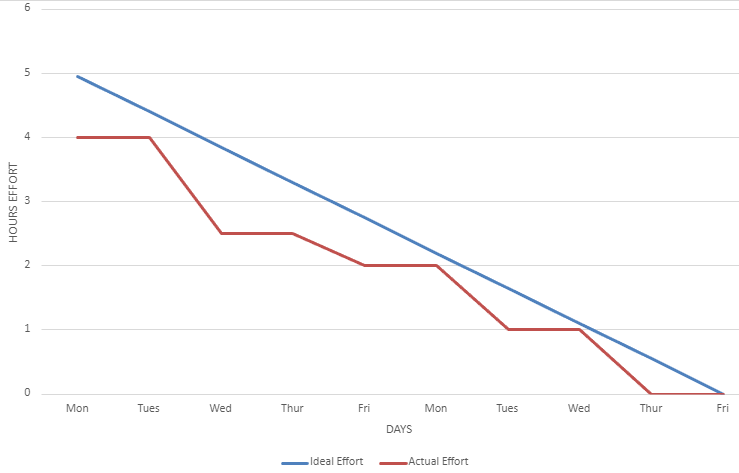
### Sprint 6



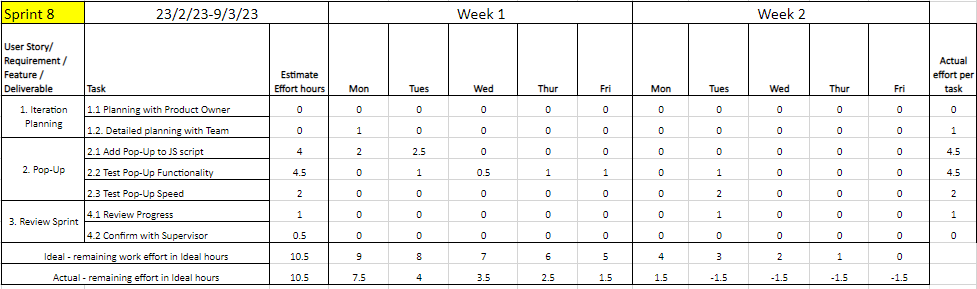


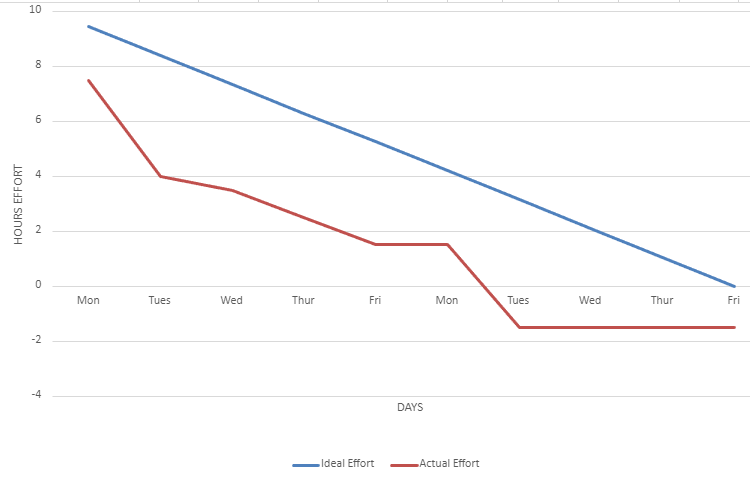
### Sprint 7



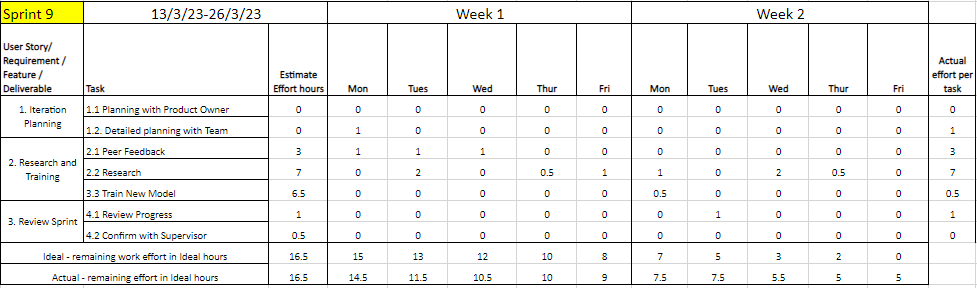


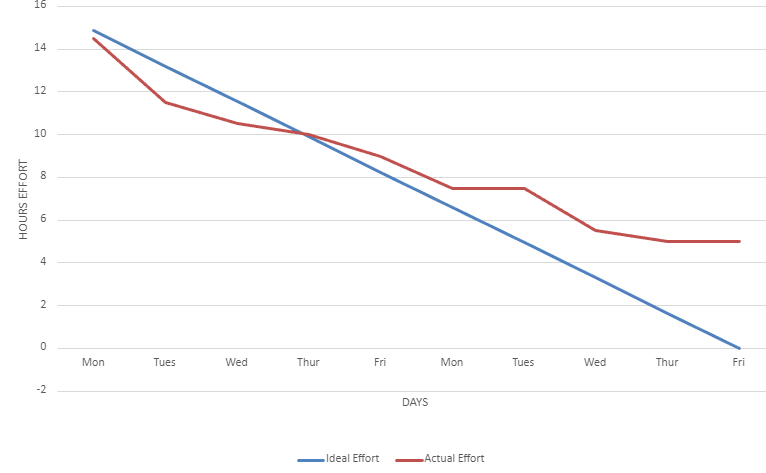
### Sprint 8



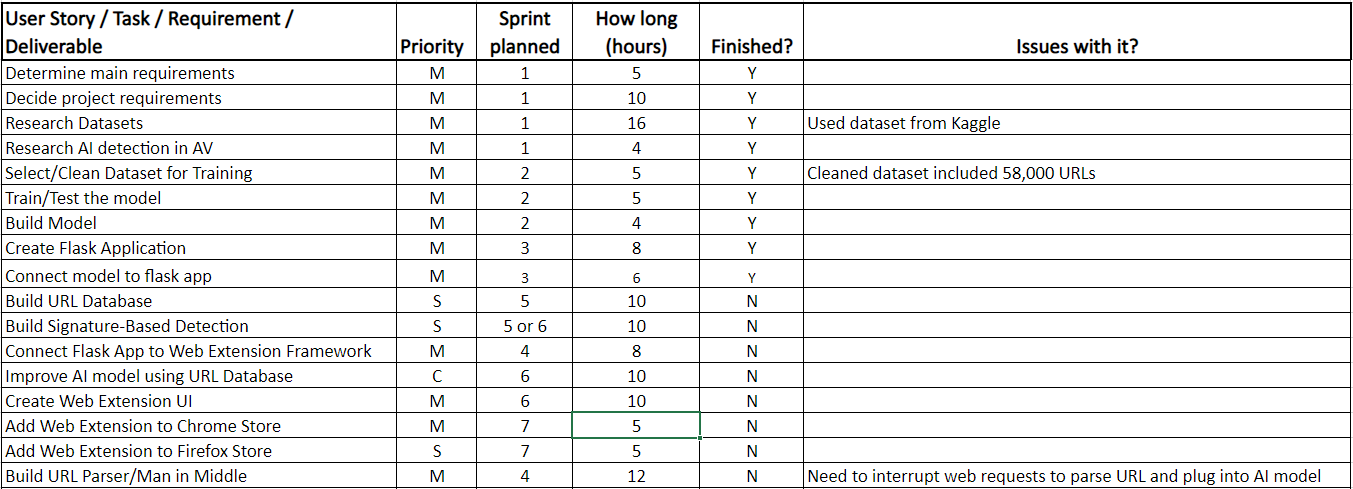


### Sprint 9





### Sprint Backlog:



## 2.3 Development Tools:

* VSCode (*Visual studio code – code editor: Microsoft Azure*)
  + Open-source light-weight code editor
  + Has a large selection of extensions to improve workflow
* Jupyter Notebook (*Jupyter Project Documentation*)
  + IDE with many use cases, so it’s ideal for training AI models as well as cleaning data and presenting findings
* OneDrive
  + Storing backups, version control
* Google Cloud (*Google Cloud Documentation*)
  + Hosting the flask app and pop-up page
* Flask (*Flask Documentation*)
  + Runs the app that contains the AI model
* Beautiful Soup
  + Data cleaning, used when automating retrieving webpage data from URLs

## 2.4 Risk Analysis:

I performed a risk analysis of the most crucial SPRINT backlog items:

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement/Deliverable/Task | Priority | Risk Level (0 Low – 3 Very High) | Risk Detail |
| Build URL Database | Should | 0 | Need to connect the database to the main Flask application somehow |
| Build Signature-Based Detection | Should | 1 | Need to create a separate function of MITM to search URL database for a match |
| Connect Flask App to Web Extension Framework | Must | 1 | Need to integrate all functionality (parser, AI model, MITM, database) into web extension framework |
| Improve AI model using URL database | Could | 2 | Requires a large enough database |
| Create Web Extension UI | Must | 0 | Create minimal UI, keep the look simple, I’m not an artist... |
| Build URL Parser/MITM | Must | 2 | Find a way to interrupt web requests, parse every URL and feed them to the model without slowing down the user |
| Add Extension to Chrome Web Store | Should | 0 | - |
| Add Extension to Firefox Web Add-Ons | Must | 0 | - |

# 3 Research:

## 3.1 Round 1 (Sprint 1)

Relevant literature includes:

* A comprehensive survey of AI-enabled phishing attacks detection techniques (Basit,Liu,Zafar)
* Heuristic nonlinear regression strategy for detecting phishing websites (Mehdi Babagoli, Mohammad Pourmahmood Aghababa & Vahid Solouk)
* Machine learning based phishing detection from URLs (Ozgur Koray Sahingoz, Ebubekir Buber, Onder Demir, Banu Diri)

To successfully fulfil the requirements, and to ensure that they are reasonable requirements in the first place, research would be key. I started by looking for scholarly articles relating to phishing detection using AI. From there I found three particularly useful papers.

The first (Basit et al., 2020), provided a clear answer on whether machine learning can be used effectively in this field, when referring to phishing URLs, ‘A portion of the ML procedures can identify TP up to 99%’. This figure is more than a good enough goal for our application. Figure 2 also show a consistent rising in phishing websites. As well as this, it describes ‘Classifications models [as] also depict[-ing] good performance.’ From this I decided to go ahead with a machine learning classification model.

The second, (Babagoli et al., 2018) used ‘11055 phishing and legitimate webpages, and select 20 features to be extracted from the mentioned websites’ and achieved an ‘accuracy rate as high as 96.32%’. In this paper, they chose ‘modified harmony search based nonlinear regression and SVM (support vector machine)’ detection methods. I decided to use SVM as a classification model, classifying a URL as either a phish or not a phish. It also showed it was more accurate when it used 20 features instead of the full 30 (Fig 8). I would also aim to use less features to improve the speed and memory efficiency of the Add-On.

The final paper (Sahingoz et al., 2019) mentioned some of the logistical issues when trying to train a model such as this one, ‘Due to the absence of a worldwide acceptable test set for phishing systems, we needed to construct our own dataset’. In this paper they also ‘[use] different classification algorithms and natural language processing (NLP) based features’. Another example of someone using a classification model for this job. As well as the mention of NLP based features such as ‘raw word count, shortest word length, subdomain count.’ Use of these features would keep the app very fast, as they are simply features that can be obtained by parsing the URL as as a string, a very computationally fast operation when compared to making web requests to access features on the webpage itself (I.e. number of hyperlinks).

Using this research, I concluded that I could achieve my goals with this project. I would use a Classification based machine learning model, that uses no more than 20 NLP features obtained from the URL. We are to build our own dataset with a mix of legitimate and phishing URLs (around 5000 URLs).

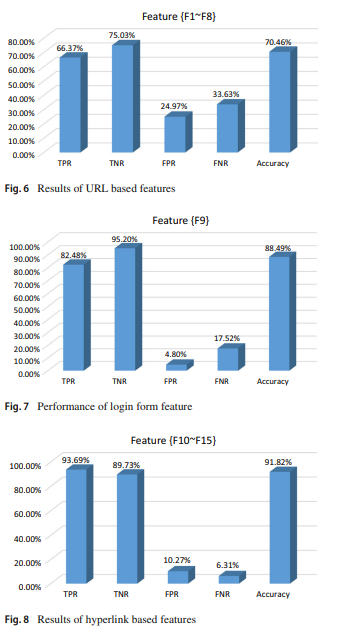
## 3.2 Round 2 (Sprint 4/6)

Relevant literature includes:

* Towards detection of phishing websites on client-side using machine learning based approach (Ankit Kumar Jain, B. B. Gupta)

The second round of research begun when improvements needed to be made to the AI model. More specifically, trying to reduce False Positives produced by the model, as well as increasing the overall accuracy above 90%.

Figures 6 & 8 (Jain & Gupta, 2017) show the difference in accuracy between ‘URL-based features’, what I’ve been using for my first AI Model, and ‘hyperlink-based features’, which are features surrounding the amount of and types of hyperlinks present on the webpage. The difference in accuracy is staggering, over 20%! As well as this the false positive rate is almost 15% lower. When they combine these features together, the model has ‘99.09% of overall detection accuracy.’



This round of research made it clear to me that to improve my model, I would need to use ‘hyperlink-based’ features. This would require a lot more data processing, as I now needed the source code for these webpages, as well as the feature extracted from it, and I would need to automate this whole process as it would take far too long to do manually. As well as this I’d have to train a new model with the added feature.

## 3.3 Round 3 (Sprint 9)

Relevant literature includes:

* PhishStorm: Detecting Phishing with Streaming Analysis
* Samuel Marchal, Jerome Francois, Radu State, and Thomas Engel

On the final round of research (Marchal et al., 2014), an older paper was discovered from 2014. In it, once again, supervised machine-learning is used to detect phishing URLs. However, the feature selection is based on Intra-URL relatedness. This is due to the concept that phishing websites tend to have little relation between the registered domain (the part that must be paid for), and the rest of the URL (lower-level domains, path, query). So, features are extracted that represent the relatedness of those URL parts. They achieved a model with an accuracy of 94% testing over 90 000 URLs.

By using a new set of features, based on the concept of Intra-URL relatedness, and combining them with the URL-based features and the hyperlink-based features of the previous models, an improved model could be created.

# 4 Product Description:

## 4.1 Data Processing:

### 4.1.1 Data Collection

#### Phishing URL Datasets

<https://www.openphish.com>

<https://www.phishtank.com>

#### Hybrid URL Dataset (Both legit and phish)

<https://data.mendeley.com/datasets/n96ncsr5g4/1>

<https://data.mendeley.com/datasets/c2gw7fy2j4/3>

<https://github.com/ebubekirbbr/pdd/tree/master/input>

#### Legitimate URL Dataset

<https://www.kaggle.com/datasets/cheedcheed/top1m>

Above is a collection of datasets, containing both legitimate and suspected phishing sites. The reason both legitimate and phishing URLs are needed is because the AI model is a classification model, where the feature being predicted is whether the URL is a phishing URL. One issue of using data from a lot of different sources, is they all need to be cleaned and re-formatted, so all the datasets are compatible (I.e. using the same columns).

Thankfully, all the sources allowed the download of their data in CSV format, following this I separated the data sets into mixed datasets and purely phishing datasets. Before removing NaN values and unneeded columns.

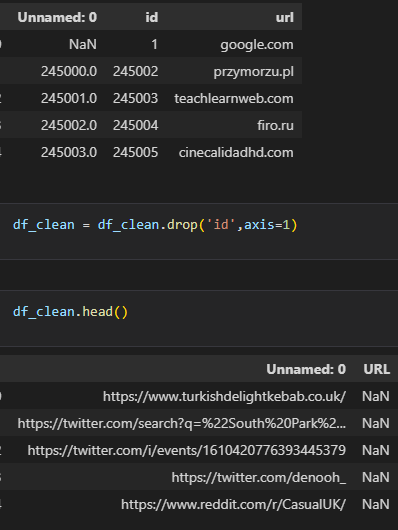
I think I have sourced a good collection of varied data for training my model, with multiple sources.

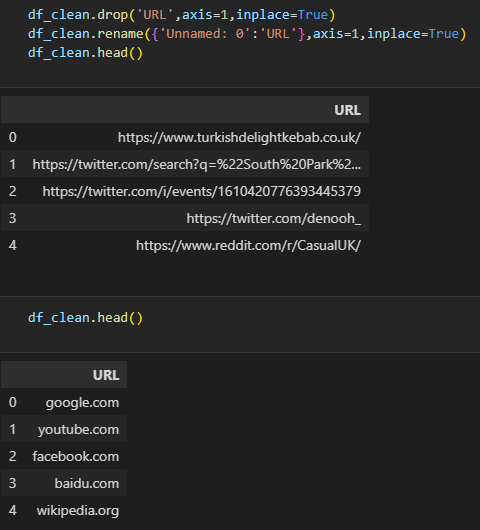
### 4.1.2 Data Pre-processing

The purpose of this step it to clean data into a format in which it can help us. This involves dropping Null values and statistical outliers

#### Clean Legitimate Data

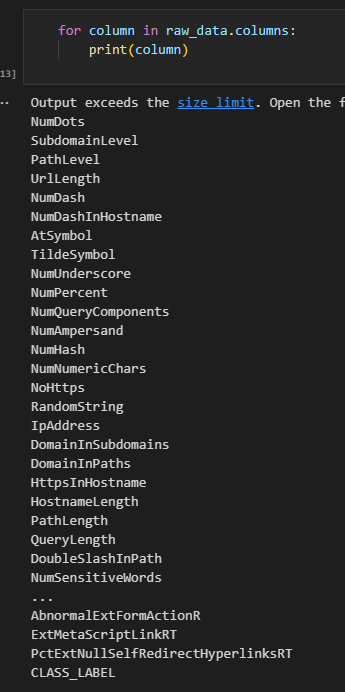
As mentioned previously to properly clean the data it must be uniform (contain the same columns) and must not hold any invalid data (such as NaN values). So, the cleaning started with these two goals (Agarwal, 2023).



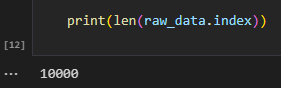


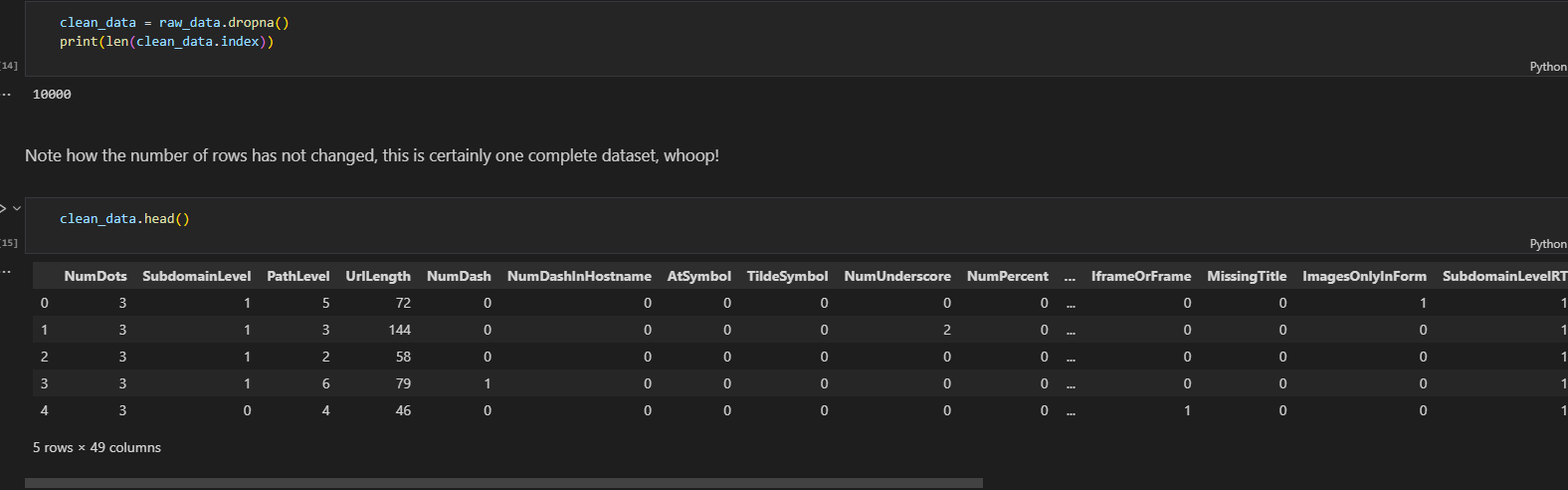
Next, unneeded columns, such as ‘Id’, are dropped and the column containing URLs is renamed to match that. At the bottom we see our data in a clean format by using the *head()* function to output the first 5 rows.

#### Clean Hybrid/Phishing data



Iterating through the column names and printing them. This dataset has features already built into the columns, so much less feature extraction is required here.





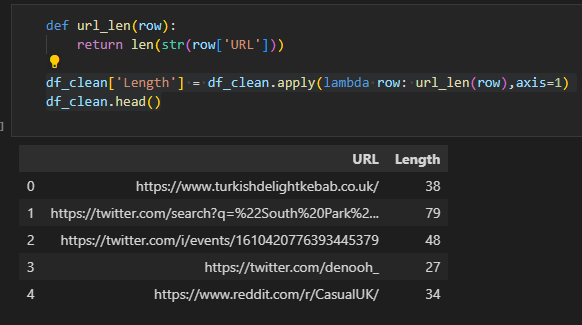
Lastly, clean the data by using the *dropna* function. As is mentioned in the notebook, *dropna* in this case did not drop any rows, implying the dataset has already been cleaned.

### 4.1.3 Feature Creation

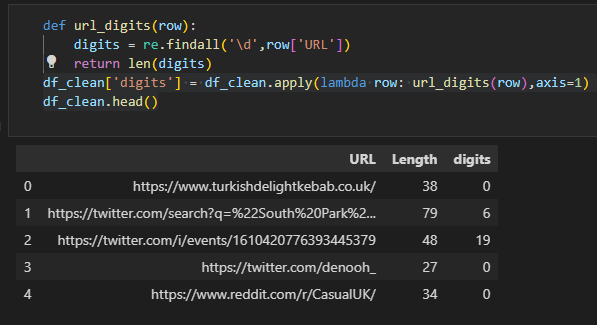
Next, following the first round of research, the goal was to create an AI model that used NLP based features from the URL string (Sahingoz et al., 2019). As seen in the previous section, the phishing/hybrid dataset already has columns representing different features of the URL such as *NumDash* and *AtSymbol*. So, I wouldn’t need to add those columns to that dataset, but I will need to for the legitimate URL data.

#### Extract features from URLs (Legitimate Data):

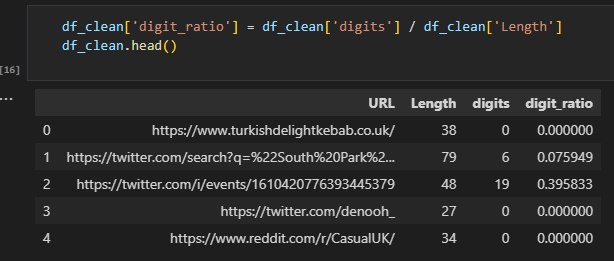
All the URL feature extraction functions take advantage of the regular expression library in Python *(re).* A particularly useful function being *re.findall(),* which returns all instances of the argument given in a string.



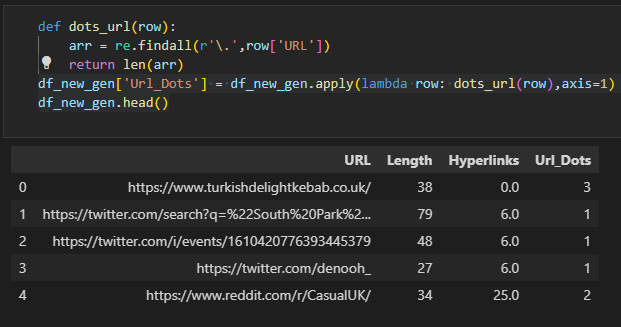
Function that returns the length of the string in the URL column.



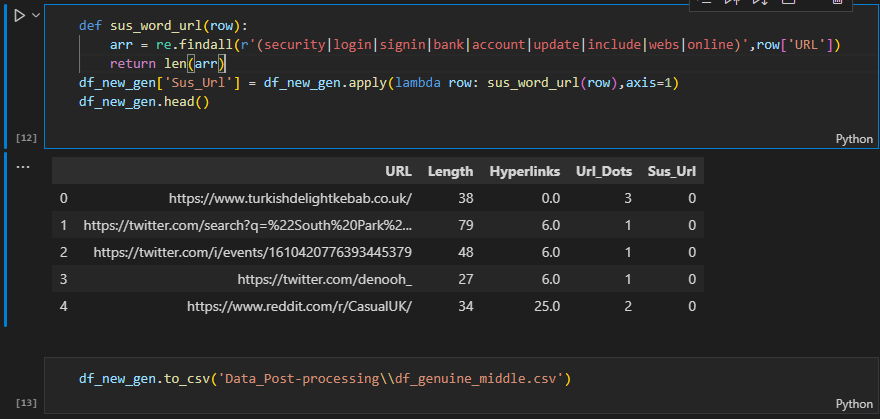
Function that returns the number of digits in the string in the URL column.



Adding a *digit\_ratio* column representing the relationship between the number of digits and the URL’s length.

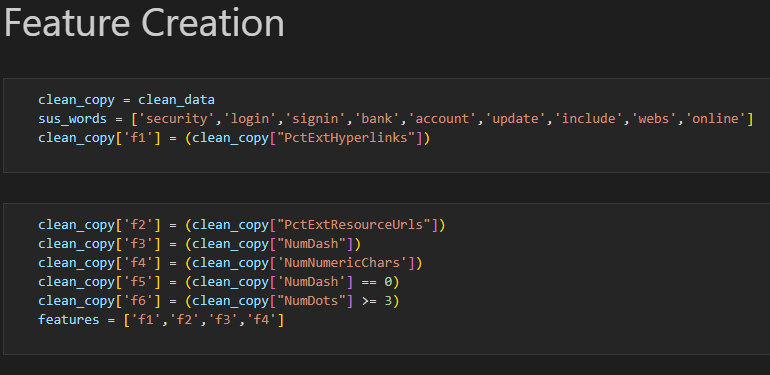


Function that returns amount of ‘.’ in the URL string.



Function that returns the number of times a ‘suspicious’ word appears in the URL. These include security, login, bank, update, and others.

#### Extract Features from URL (Hybrid/Phishing Data):



#### Extract Hyperlink Features (2nd round model):

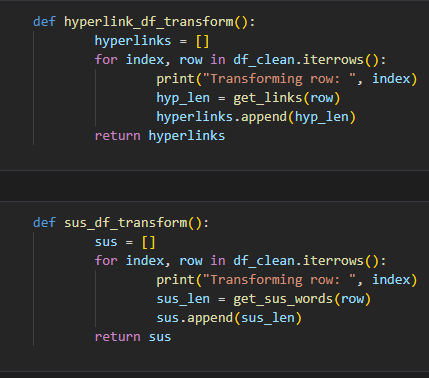
After the second round of research (Jain & Gupta, 2017), the new goal was to add a hyperlink-based feature to the model. This would require iterating through the data, retrieving the URL, making a web request to the URL, retrieving the source code of the page, and finally parsing through the page to find hyperlinks, adding them to an array before returning the length of that array. This whole process would obviously have to be automated, as the dataset is just too large for manual entry.

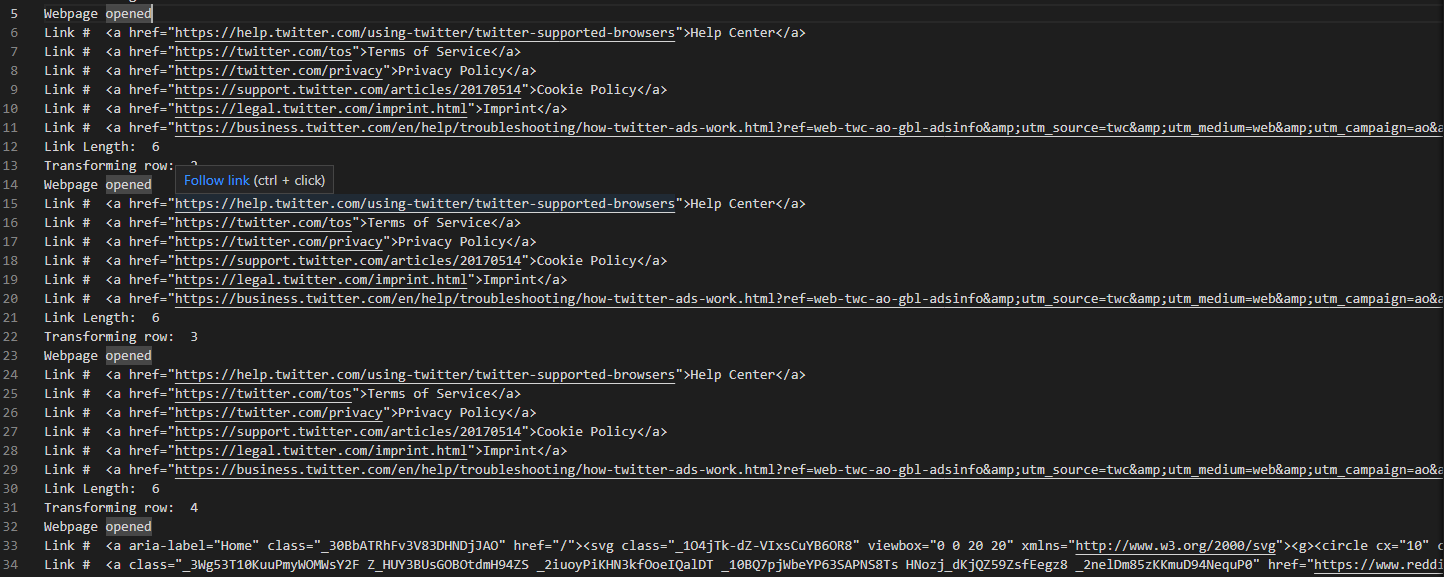


The *get\_links()* function opens a URL, parses it with BeautifulSoup, and returns a list of links found on the page. It also checks if the website is automatically blocked by my ISP.

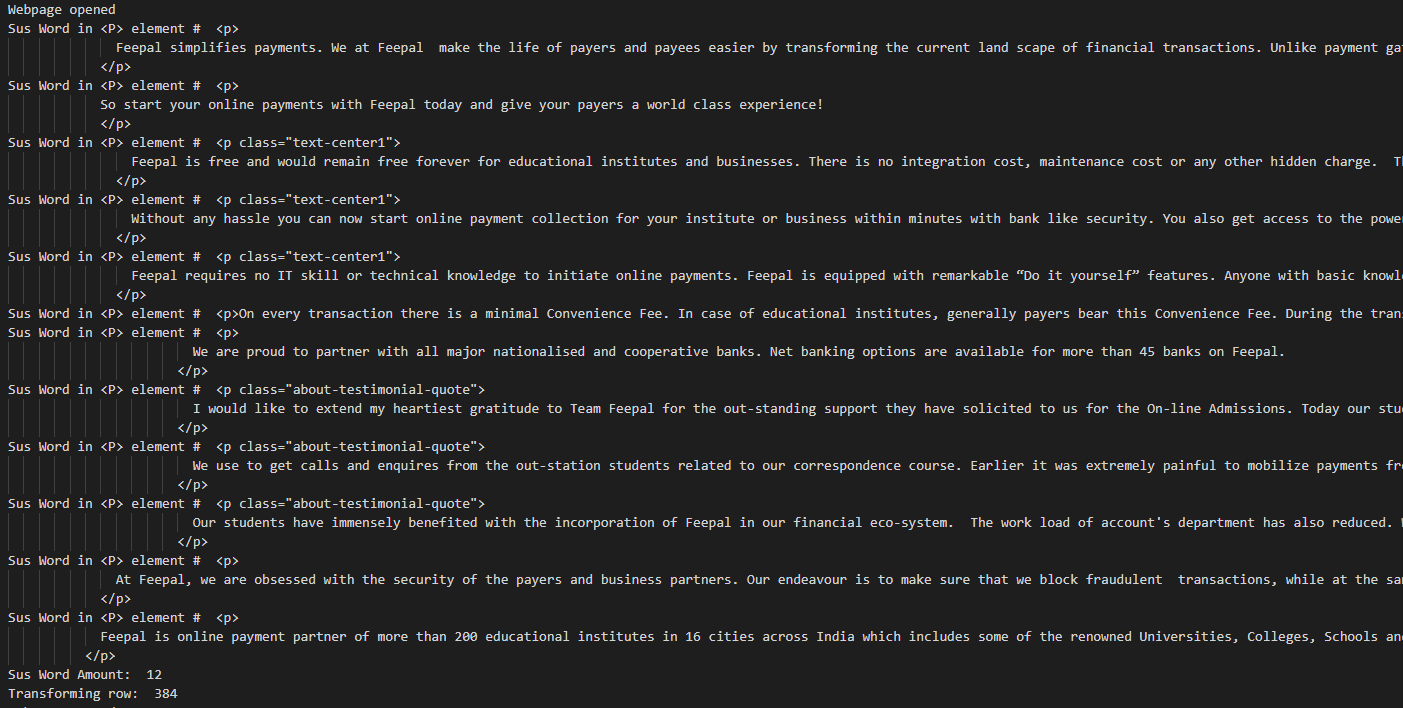


Here is the same function but instead of returning the number of hyperlinks in a webpage; it returns when a word contained in the suspicious word list is found in the webpage. The only difference between the two functions is the first line of the for loop, which iterates through different webpage tags. Regular expressions are used to compile the list of suspicious words.



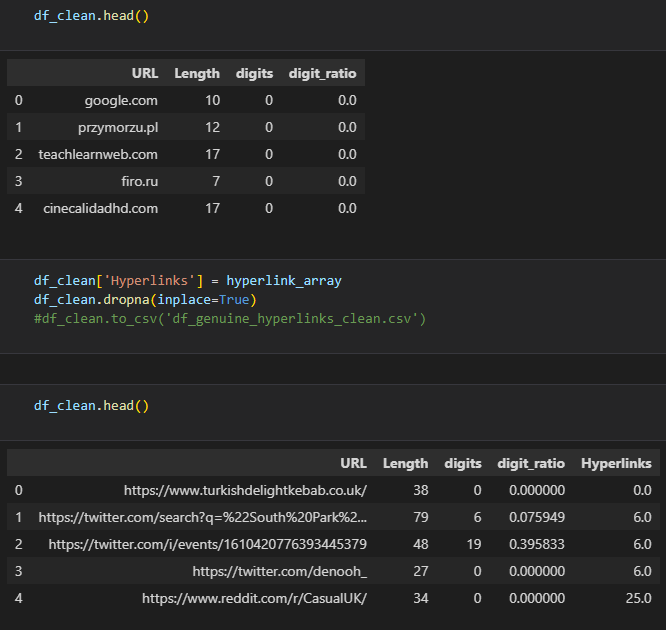


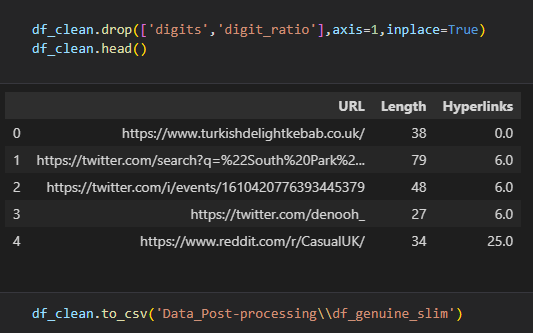
(Output of *hyperlink\_df\_transform()*)



(Output of *sus\_df\_transform()*)

#### Appending Hyperlink Data (Legit URLs)

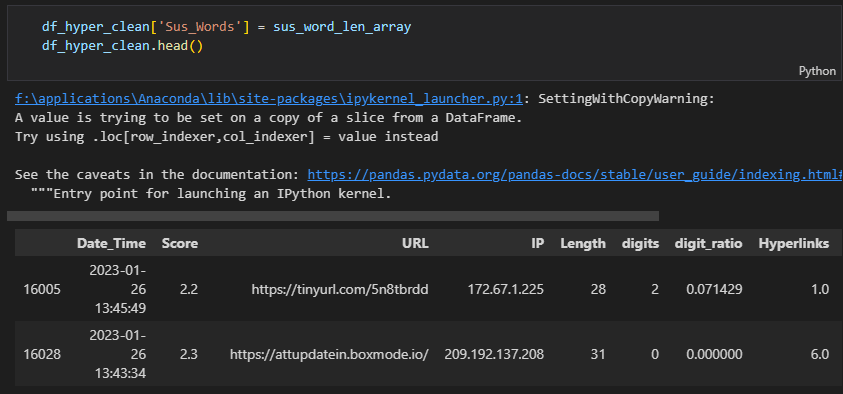




Here we add the hyperlink column by appending the ‘hyperlink\_array’ array and giving it the column name ‘Hyperlinks’

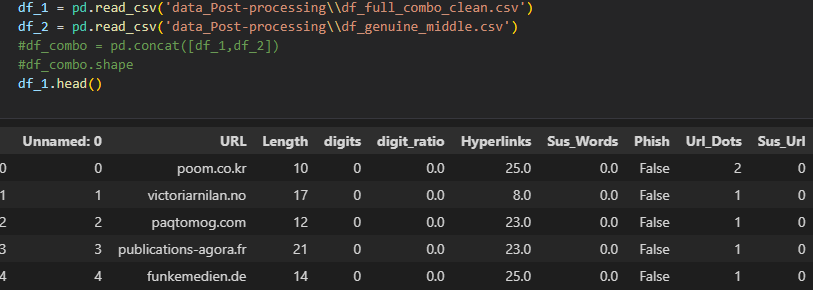
#### Appending Suspicious Words Data (Legit URLs)

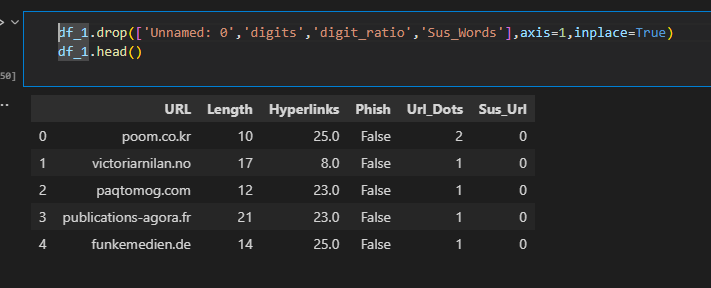




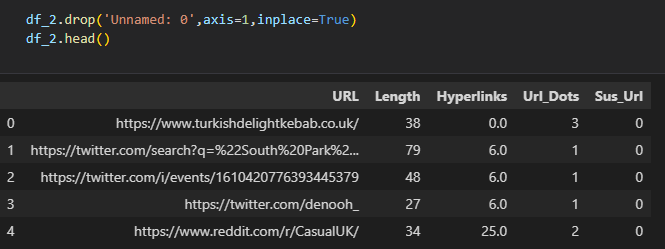
Here we do the same thing but with the *sus\_df\_tranform* function.

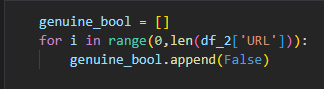
#### Combine and Clean Data

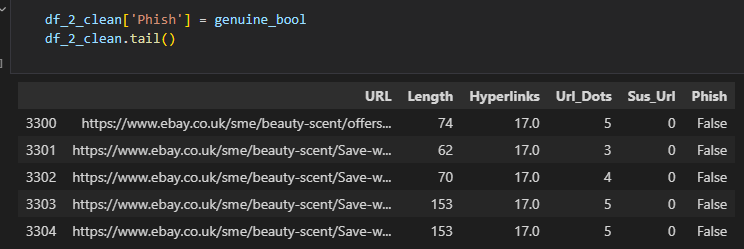




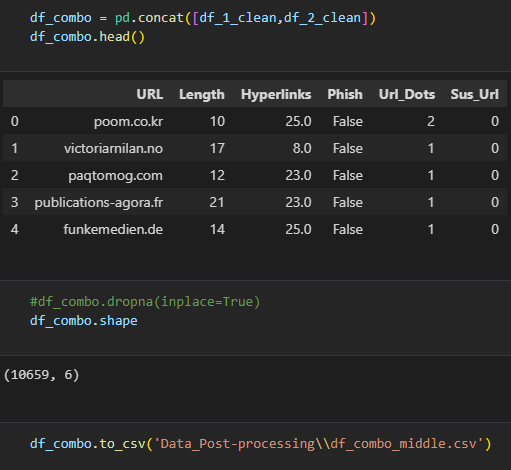
Note that here I also drop the ‘Sus\_Words’ column representing the suspicious word count, as I decided to leave it out as a feature of my final model, as well as the ‘digits’ and ‘digit ratio’ feature







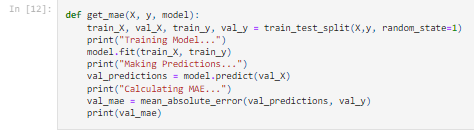
Iterate through the genuine URL dataframe and add a phish column with the value ‘False’.

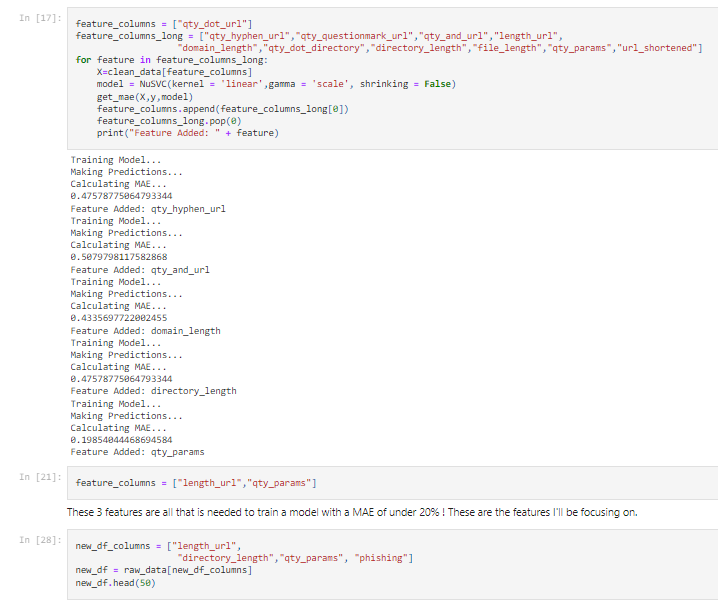


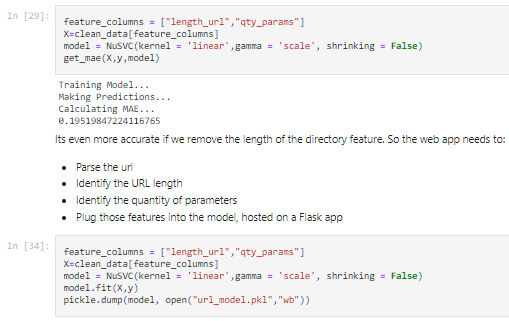
Combine the data using the *pandas* *concat* function and export to new CSV.

## 4.2 AI Model

### 4.2.1 1st Round of Training





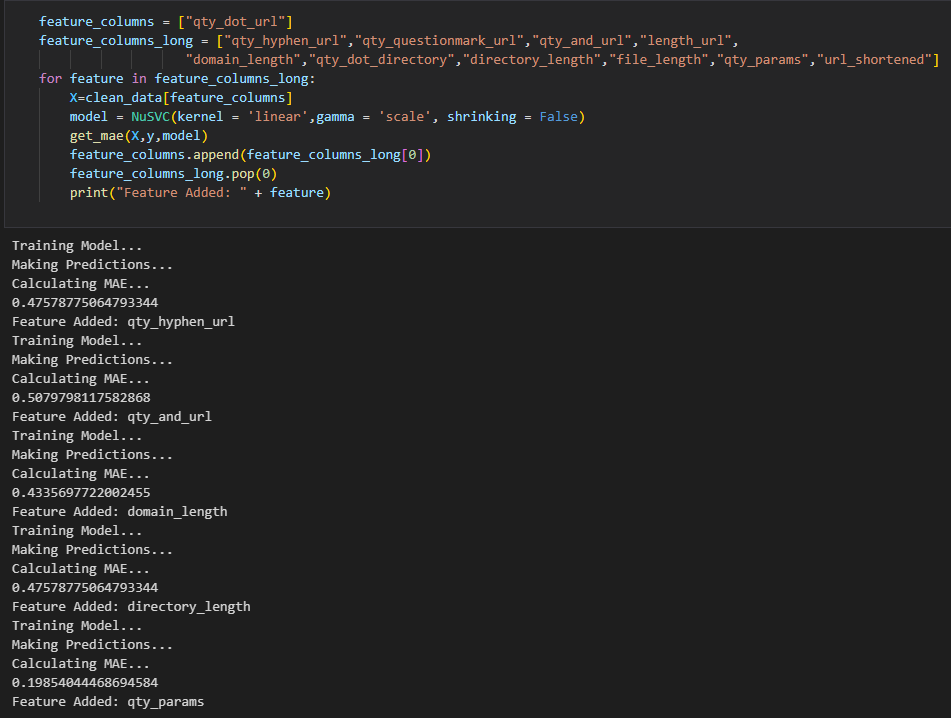


For the first round of training, I was working with the knowledge I had gained from the first round of research (Sahingoz et al., 2019). And so, this model uses URL-based features to identify phishing URLs.

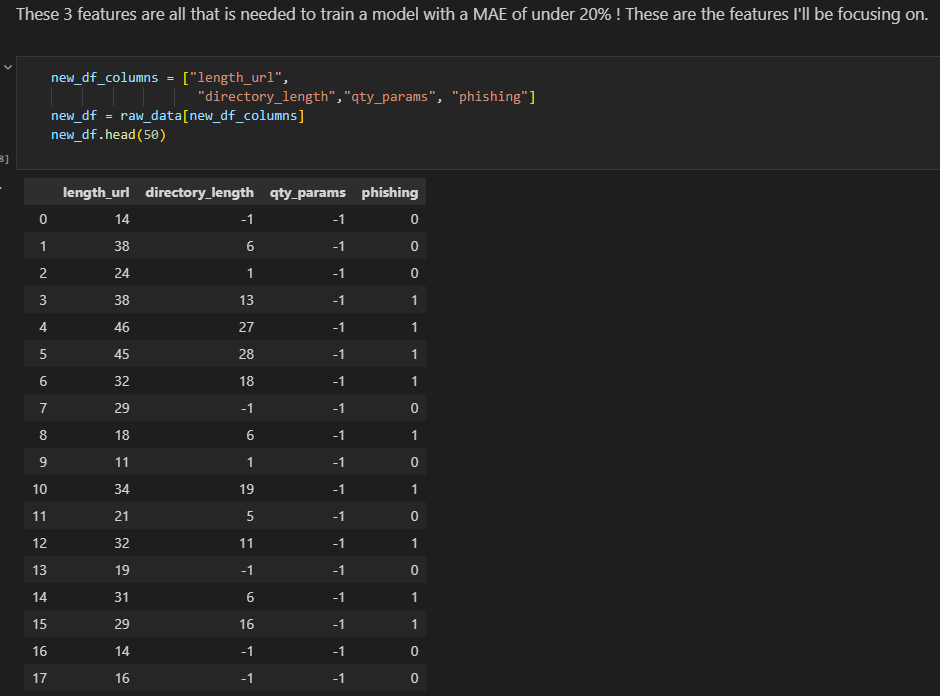
On top of this I made a mistake in my MAE function, to understand this let's look at the documentation for the *mean\_absolute\_error function* (Pedregosa, 2011)*.*

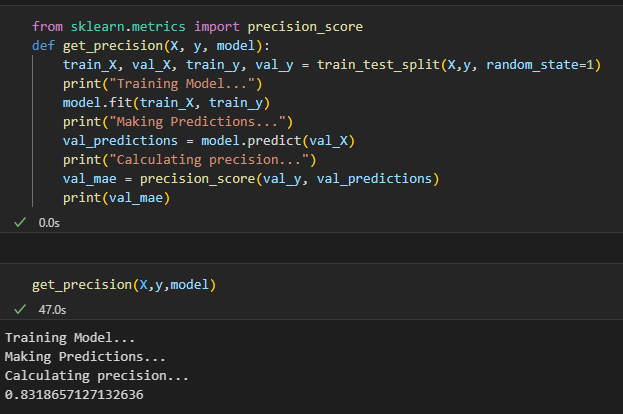


If we look back to my function, you may notice that I provide the arguments the wrong way around, thus (somewhat ironically) producing an inaccurate test of the model’s accuracy.

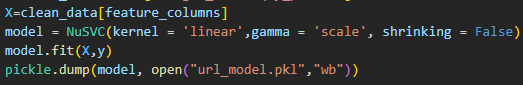


Here is where the features are tested to see how much they improve the model's accuracy, by testing it and then adding features and testing again automatically. This would have given me useful information about which features are most useful, however, as mentioned before, in this round of training I used the *mean\_absolute\_error* function incorrectly. I have made sure to fix this error moving forward.





Despite this mistake, the *precision\_score()* function was correctly implemented and returned a precision of about 83%. This means that when the model identifies a URL it believes to be suspicious, it is right about 83% of the time, this is measured by dividing the true positives by true positives and false positives. And it is a useful metric in determining how well a model is performing (C3 AI, 2022).



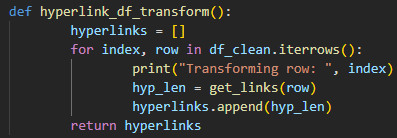
Dump the model into a binary using *pickle* (*Kerr*).

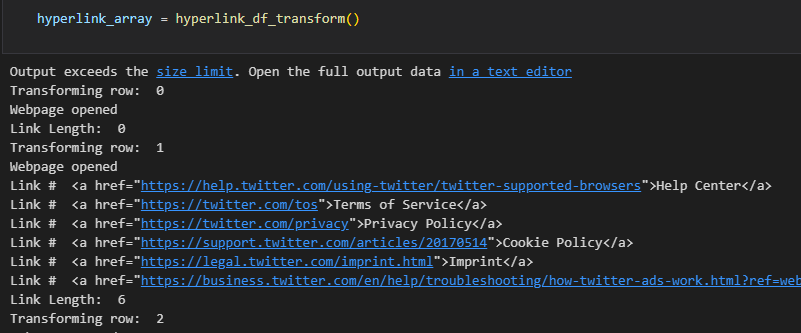
And just like that, the first version of the AI model has been created!

### 4.2.2 2nd Round of Training

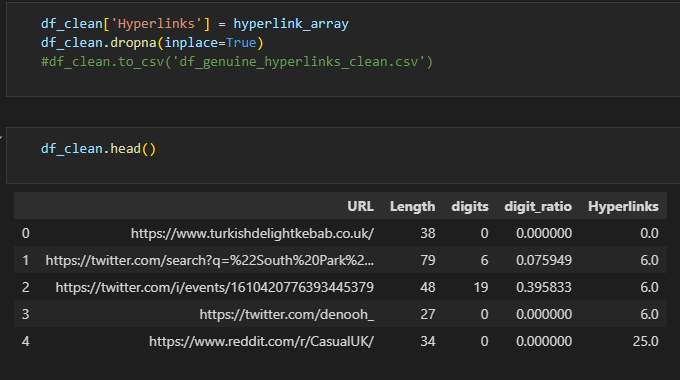
* 2nd Round Training (Sprint 4/6)

For the second round of training, I had a new goal in mind, inspired by my second round of research. I needed to create the previously seen *get\_links()* and *get\_sus\_words()* functions. As mentioned before, these functions automate the process of retrieving hyperlink-based features from the source code of the webpages. According to sources found on this second round of research (Jain & Gupta, 2017), adding hyperlink features could increase the accuracy of the model above 90%!

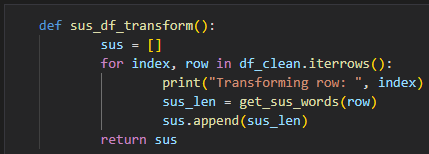


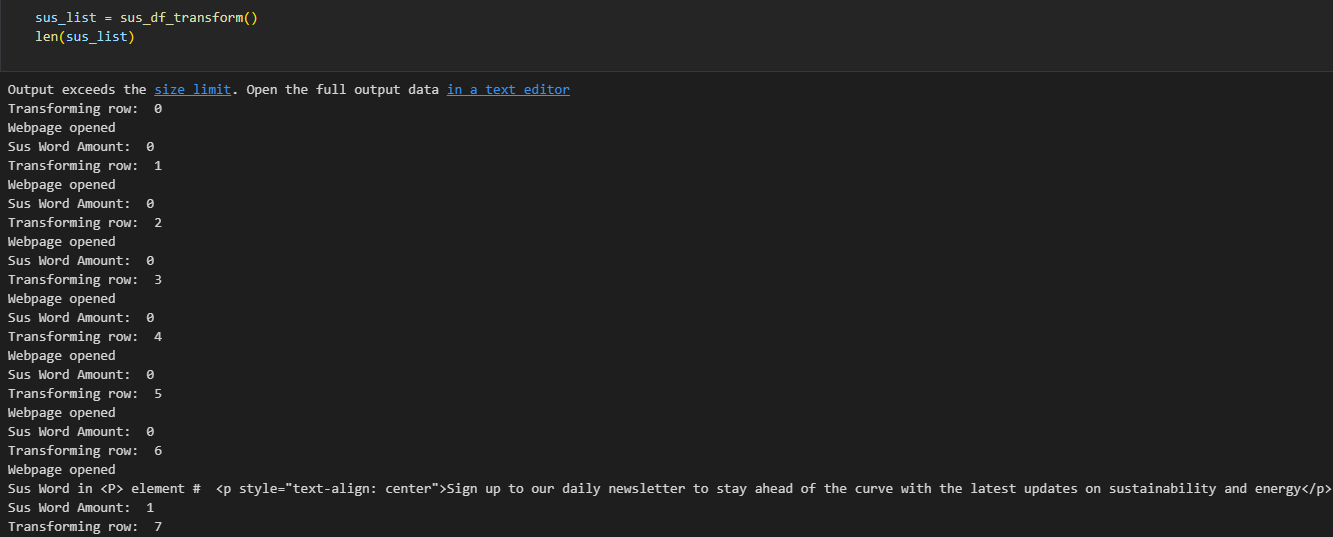


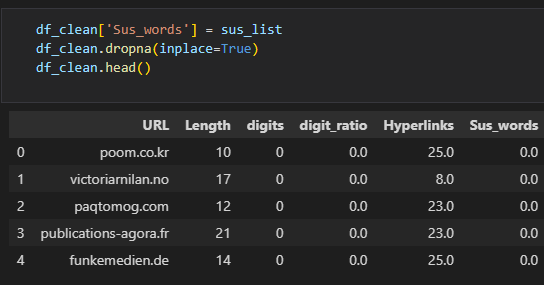
Shown above is the function that iterates through the table and creates an array representing the number of hyperlinks of each webpage.



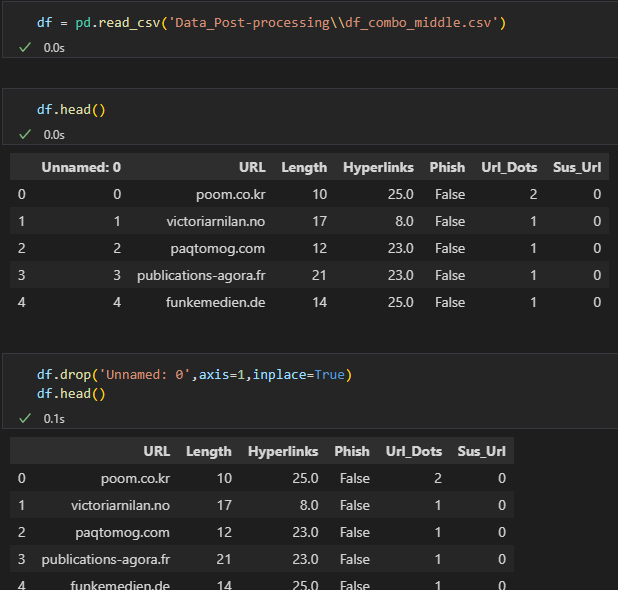
Then we add the array as a new column to our table and clean it up. We then do the exact same process to retrieve suspicious words found in the webpage using the previously mentioned *get\_sus\_words()* function.

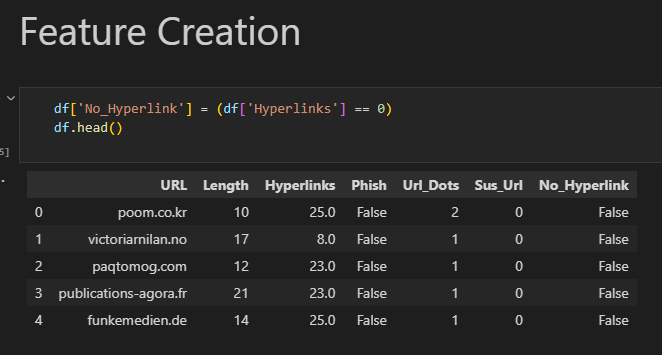




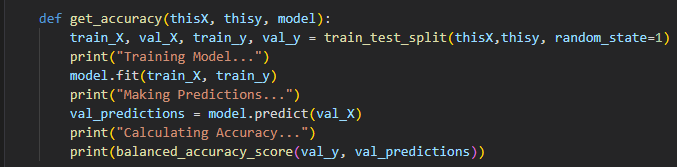


Now the creation of hyperlink-based features for our model is a simple process.

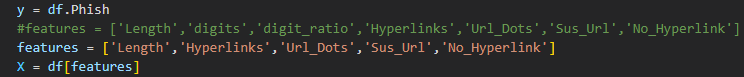


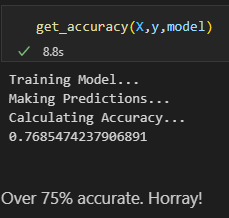


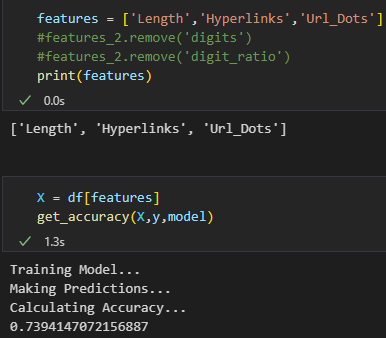
Create a function to get Mean Absolute Error / Accuracy as they are both good metrics for determining the quality of an AI model (Pedregosa, 2011)



Here you can see the arguments are the correct way round now. Next, we test out the accuracy of the model using different features





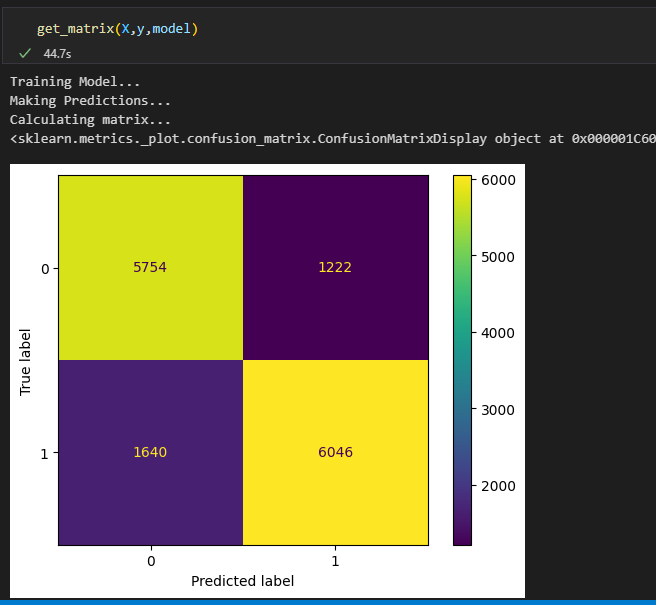




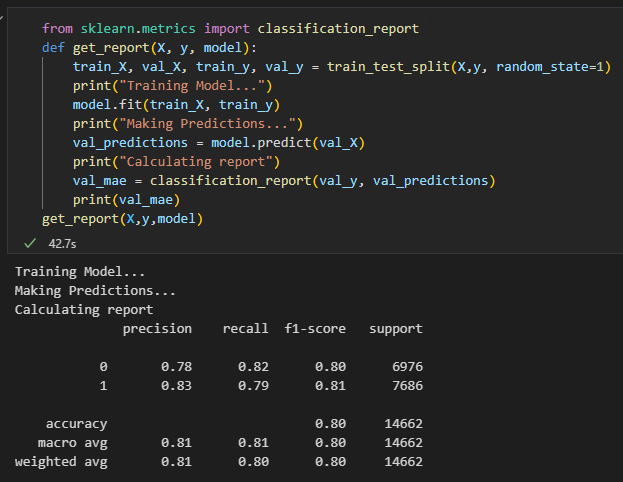
The accuracy hasn’t improved like I’d hoped, in fact it seems to have worsened by about 6%, more metrics will be needed to properly evaluate both models

### 4.2.3 Evaluation of Models

#### 1st Round Model

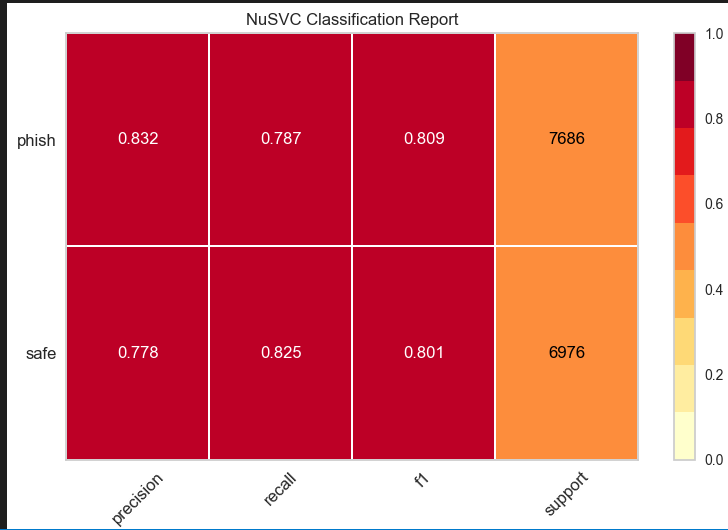


Confusion matrix for the first model, a confusion matrix plots the predicted and true values in a 2x2 matrix. (Pedregosa, 2011)

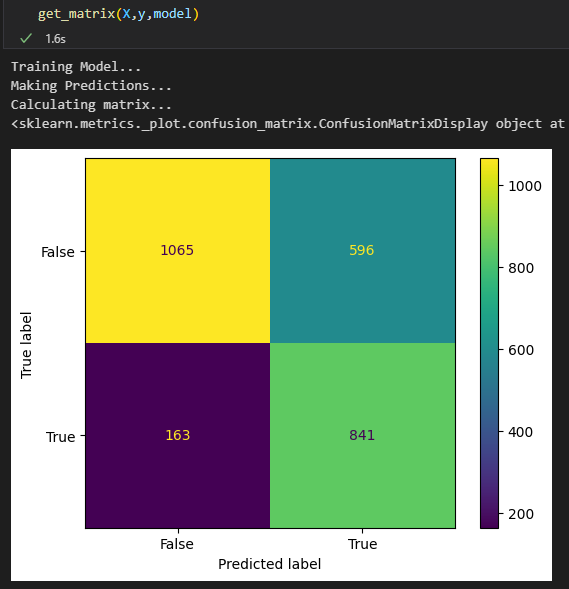


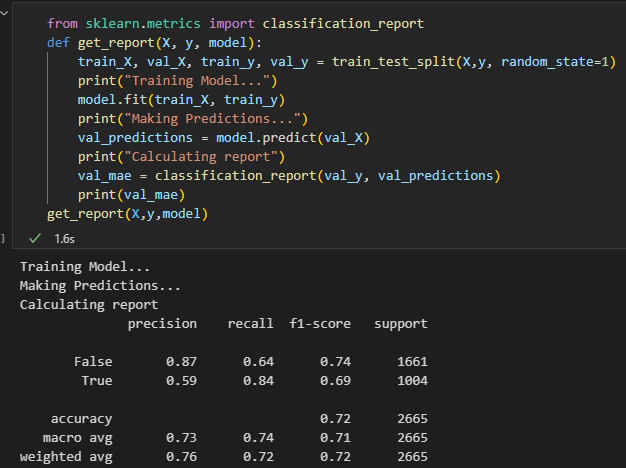
Classification report displaying key metrics for the AI model: precision (explained in section 4.2.1), recall, f1-score, accuracy, and support (explanation ahead).

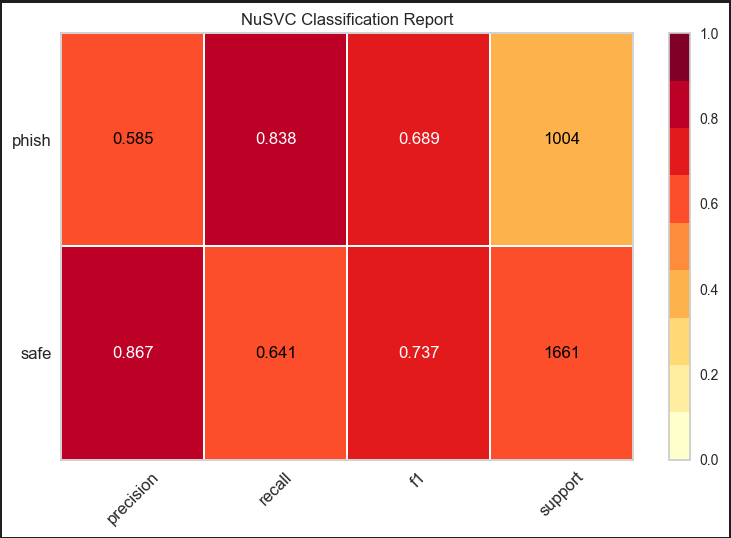
* Recall (when describing a positive) is tp (true positive)/ tp + fn (false negative), as google describes it ‘Recall attempts to answer the following question: What proportion of actual positives was identified correctly?’.
* Accuracy is tp + tn (true negative)/ tp + tn + fp (false positive) + fn, according to google ‘accuracy is the fraction of predictions our model got right.’
* Finally, f1-score is a measure of accuracy using the precision and recall scores. (Pedregosa, 2011)



#### 2nd Round Model







Adding the new hyperlink features has worsened the model. Lowering the accuracy and f1 score (reference), while dramatically increasing the number of false positives. I don’t know why this is, as it seems at odds with the 99% accuracy of the paper by Jain and Gupta.

One explanation could be that only using one hyperlink feature isn’t enough, and is in fact detrimental to the model, however if I added enough, it would eventually improve the accuracy of the model past 80%.

Another reason could be that they had a third set of features aside from URL-based and hyperlink-based, to detect fake login pages. Functionality like this may be needed to improve the model so it surpasses the first iteration.

As well as this, the difference in the support metric between True and False (or 1 and 0) seems relatively larger on the 2nd model. The ratio of the difference between the support (occurrences of the class in training data) to the total occurrences is larger in the 2nd model, suggesting that it may be imbalanced. (Pedregosa, 2011)

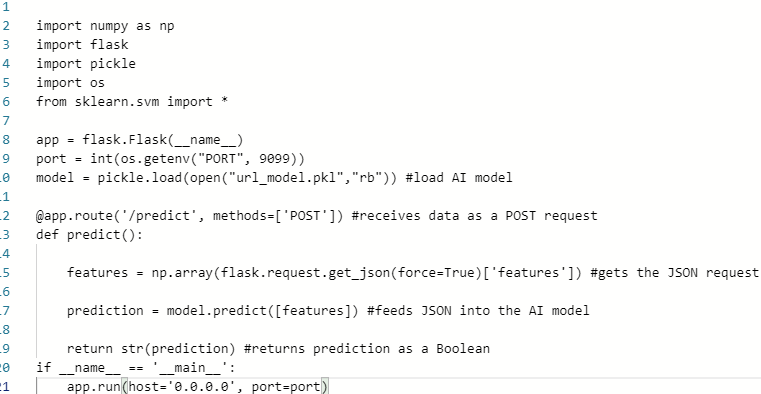
* 2nd model - (1661-1004)/(1661+1004) = .25
* 1st model - (7686-6976)/(7686+6976) = .05

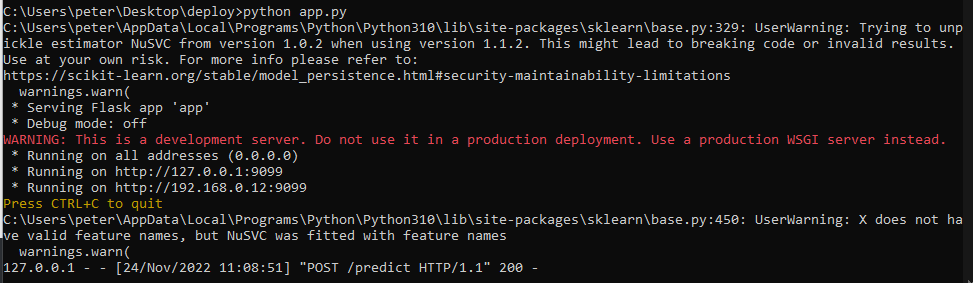
The 2nd model has improved in some areas, both precision for safe classifications and the recall for phish classifications have both improved. This may be due to a model imbalance, or just not enough data. Interestingly, the increase in the recall metric for phishes suggests that the new model is more effective at identifying a positive phish when it is given one, but the dramatic fall in precision for a phish also means that of all the times it predicts a phish, it’s only right about 58% of the time. This would mean a dramatic increase in fp.

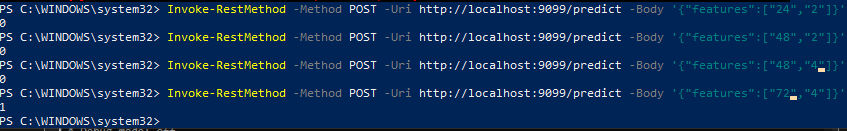
For now, the final implementation of the app still uses the first model, as it performs the best, however, I planned for a third round of research and a new model that hopefully surpasses the first.

## 4.3 Flask App:

* Test on localhost





 Above is the Flask web service running (Breuss, 2023), and successfully receiving REST style requests, and returning a Boolean result. This is the core functionality of the Web Extension.

The model constant can be changed at the top to update the app with a new AI model. Although one must also update the corresponding JS, so it provides the correct number of features to the model. The job of the python app is to:

* Contain the AI model
* Receives Features from JS script
* Send JSON containing the model's decision back to JS script

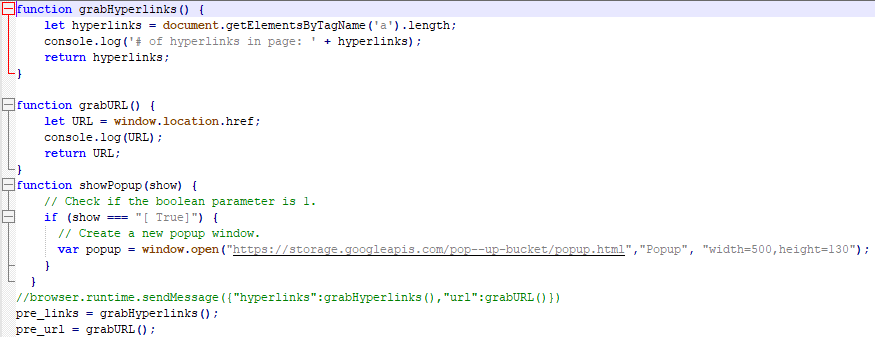
## 4.4 JavaScript App:

The JavaScript portion of our app is where we implement most of the functionality. The job of the JavaScript portion is to:

* Extract features from URL/web page
* Log the features in the browser console
* Send a JSON request containing features to the flask application
* Receive the model’s decision through JSON request
* Make a web request to the pop-up



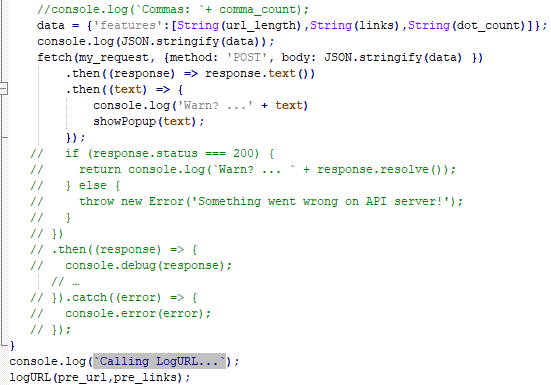
To submit an add-on, a manifest file must be provided with it, this contains meta-data about the application, as well as fields to specify which files to use, and which URLs to run the JavaScript files on. In this case the matches field is “\*://\*/” which is the equivalent of all URLs (MDN Contributors, 2022).



Above are functions to grab all hyperlinks from the current webpage, grab all URLs from the current webpage using the document object model *(document*.), and finally to open the warning pop-up when the provided argument is *“[ True]”* (this is because the flask app sends this value back when it detects a malicious URL.)



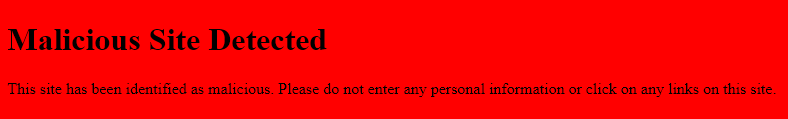
Above is the *LogURL* function that contains most of the application’s logic. Commented out are features that a previous version of the AI model used such as: URL digit count, ratio of digits to all characters in URL, a count of suspicious words in the URL and so on. There's also an if statement near the top that checks for web requests made to the flask application, and returns before the main functionality, as obviously we don’t need to verify that this site is safe from phishing as we made it. This also cuts down on a lot of unnecessary traffic. *My\_request* is also initialised to the URL of the google cloud engine hosting the Python Flask app.



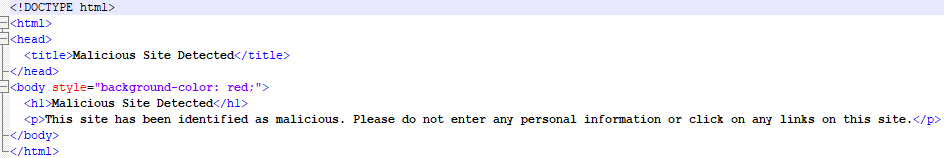
Above is the most important part of our JS, the program creates a dictionary containing all relevant features to send to the Flask app. It’s then converted into a JSON format and sent to the Flask app as a post request to the */predict* endpoint. The program then waits for a response when it is received the console logs the result and the *showPopup* method is called with the Flask result provided as an argument.

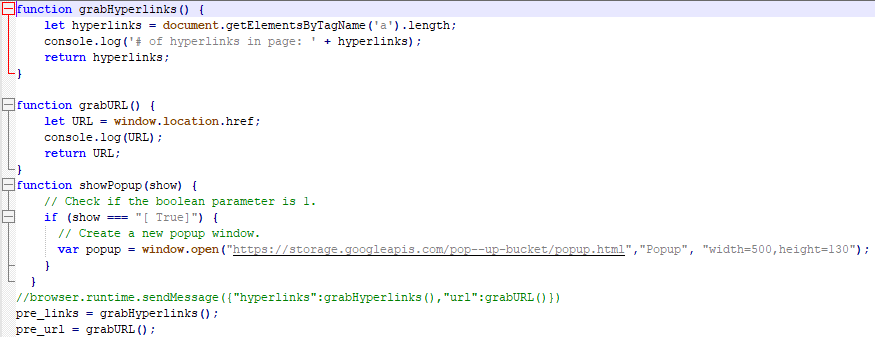
## 4.5 Pop-Up:

* New window opened with pop-up when malicious link is detected

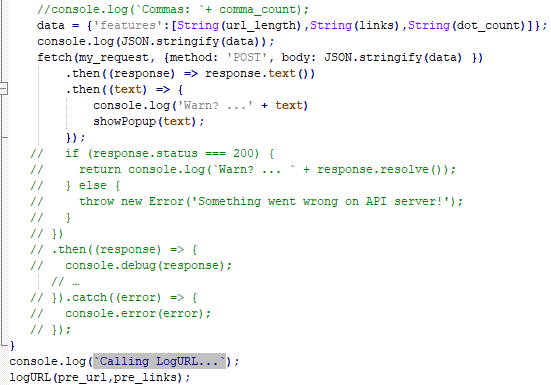


Above is the Pop-up

HTML Code for the Popup



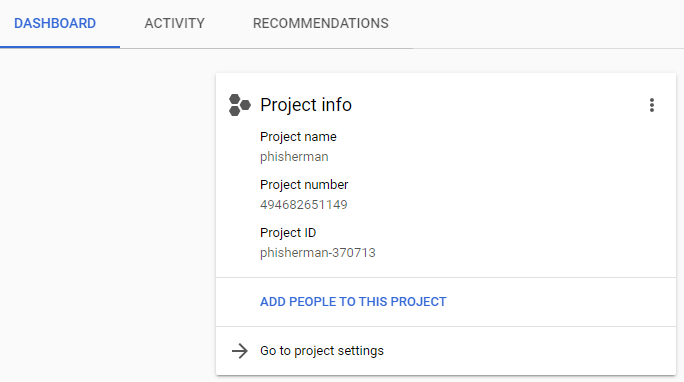
Above is the JS *showPopup* function. This uses the *window.open* function to make a web request to the google bucket that contains the pop-up

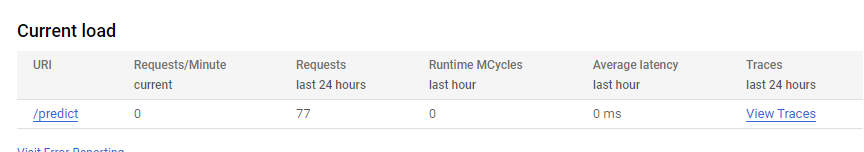


...And the function call

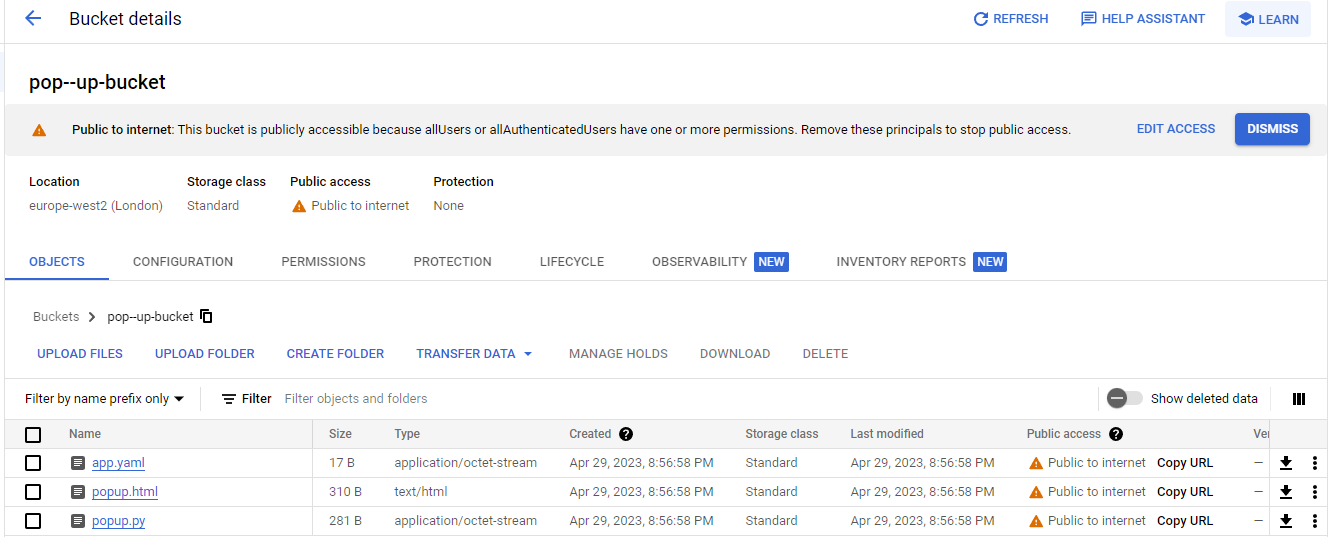
## 4.6 Google Cloud Implementation:

* Host flask app



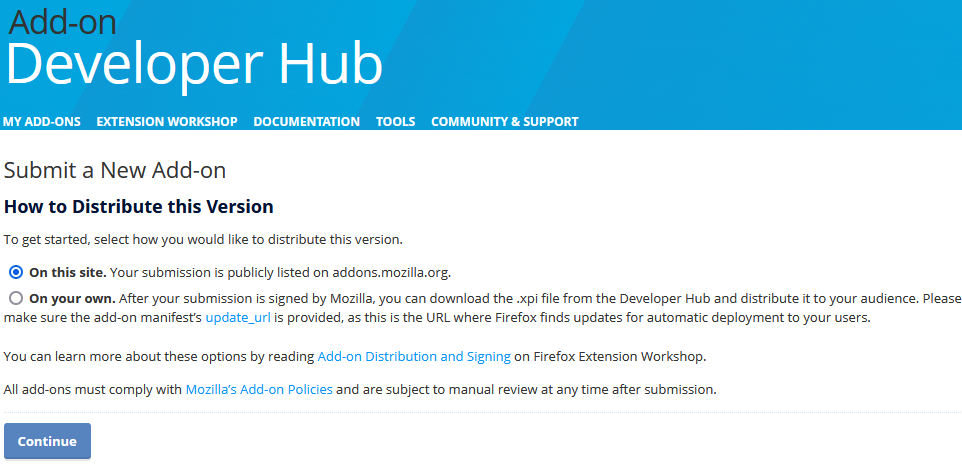


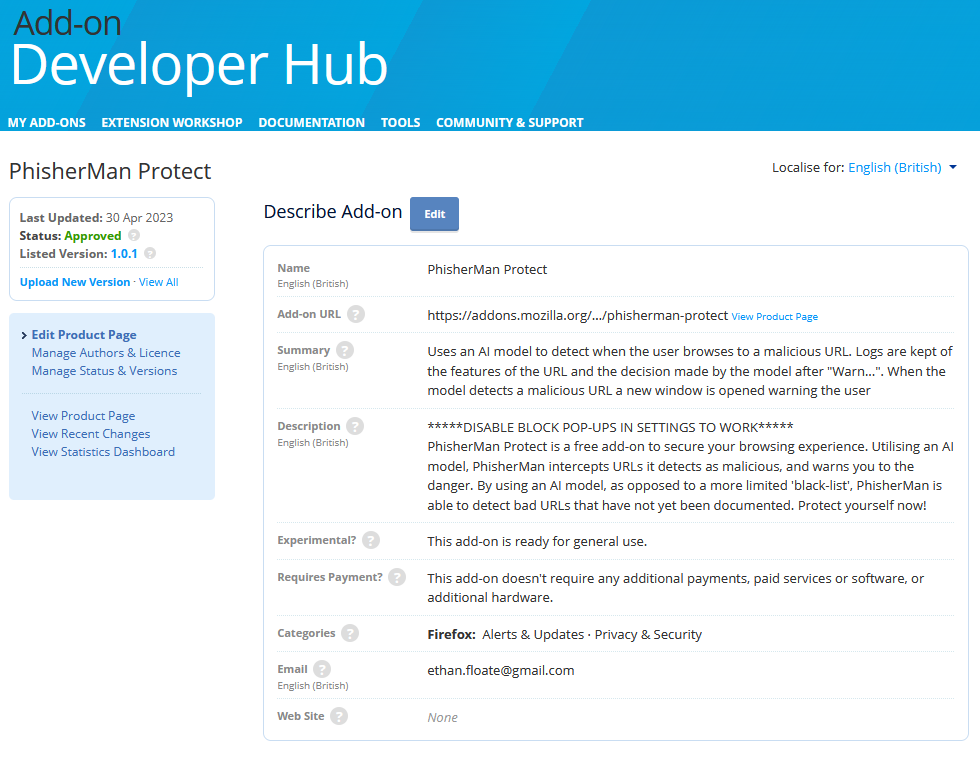
* Host pop-up HTML page
* (*Google Cloud Documentation*)

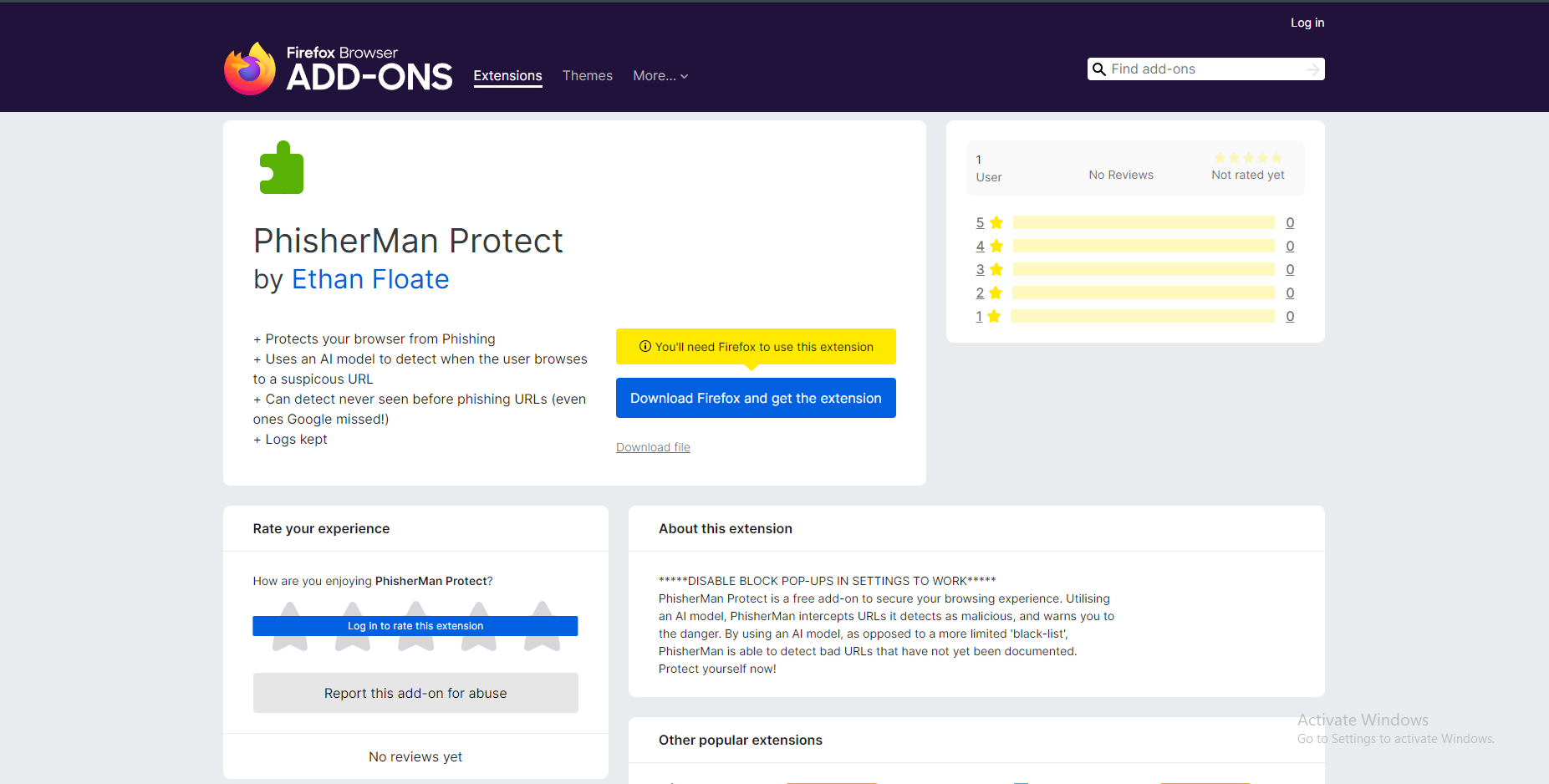


## 4.7 Firefox Add-On:

To submit an add-on, it must go through Mozilla's review process, once the add-on is approved it will be available for public download on the add-on store, and its status will be changed to ‘Approved’, this also applies when updating the add-on (MDN Contributors, 2022).





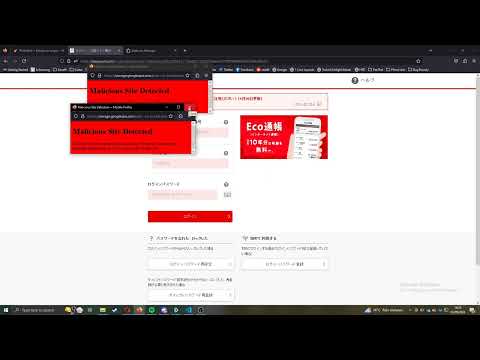


Here we see the add-on page for my browser extension! It can be found at: <https://addons.mozilla.org/en-GB/firefox/addon/phisherman-protect/>

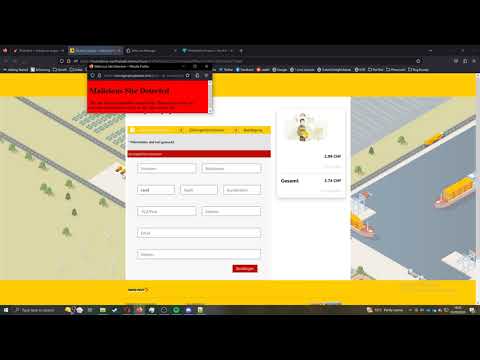
## 4.8 Demo:

These demonstrations show the app being used to test verified ‘phish’ submissions from the website *PhishTank*.

Demo 1 - [CI601 phishman demo1](https://www.youtube.com/watch?v=L0jUK-mJ1Ck)

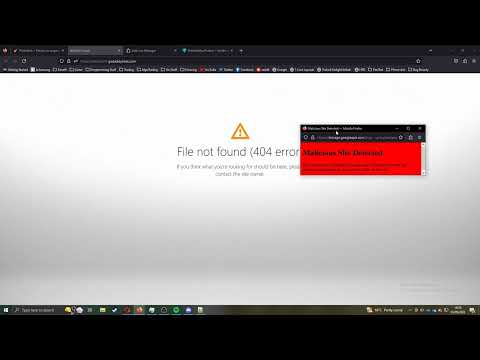
[](https://www.youtube.com/watch?v=L0jUK-mJ1Ck)

Demo 2 - [CI601 phishman demo2](https://www.youtube.com/watch?v=6cN3qOHJVq0)

[](https://www.youtube.com/watch?v=6cN3qOHJVq0)

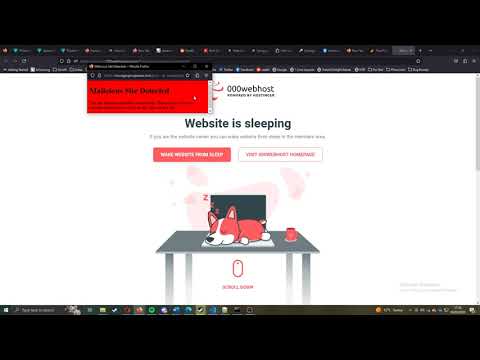
Demo 3 -

[CI601 phishman demo3](https://www.youtube.com/watch?v=dkSmgCOkW00)

[](https://www.youtube.com/watch?v=dkSmgCOkW00)

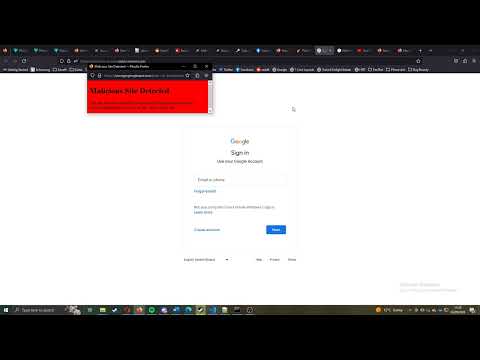
Demo 4 -

[CI601 phishman demo4](https://youtu.be/9MfOcqlSqLU)

[](https://youtu.be/9MfOcqlSqLU)

Demo 5 -

[CI601 phisman demo5](https://youtu.be/WF1dDLMWxT0)

[](https://youtu.be/WF1dDLMWxT0)

# 5 Critical Review:

## 5.1 Functional Requirements:

### Must (Priority 0):

* ACHIEVED:
  + Train a Pythonic AI model to detect malicious URLs with at least 50% accuracy
* NOT ACHIEVED:
  + Build a web extension that interrupts web requests and alerts the user to a malicious URL with at least 50% accuracy

The first requirement ‘The application must use an AI model that I have trained, to detect a web request to a malicious URL with at least 50% accuracy’, I fulfilled without too much problem, as it turns out that 50% accuracy isn’t that high according to my research (papers often featured models with accuracies of over 90%). I decided on 50% accuracy as minimum as I felt this would be a good metric to prove that the application has use and can identify a malicious URL with greater accuracy than a random coin-flip.

The second requirement specifies that the web extension *interrupts* the web request. This proved very difficult to implement. The reason for this being that the model's detection relied on features extracted from the webpage, which would require the web request to NOT be interrupted. As you may see, this creates a catch-22 of sorts. So, I decided to instead just warn the user with a pop-up.

### High Priority (Priority 1):

* ACHIEVED:
  + Build a web extension that alerts the user to a malicious URL with at least 75% accuracy
  + Return a true positive malicious URL and warn the user in under 4 seconds
* NOT ACHIEVED:
  + Build a database of malicious URLs

One of the AI models I created had an accuracy of 76%, just clearing this first requirement. However, I did not achieve my goal of creating a database of malicious URLs. With the benefit of hindsight, a better goal may have been to build a database of both malicious and safe URLs. As false positives have been one of the biggest weaknesses of my app. So, using signature-based detection as a kind of contingency when facing false negatives or false positives would be a valid improvement.

One requirement I smashed out of the park, so to say, is warning the user in under 4 seconds, as you can see from the demonstration videos in section 4.8. My app works significantly faster than 4 seconds, often working in under a second. This may be a result of making the application an add-on that runs in the browser, as opposed to a piece of desktop software.

### Medium Priority (Priority 2):

* NOT ACHIEVED:
  + Use the database for signature-based detection
  + Return a true positive malicious URL, add the URL to the database, and warn the user in under 4 seconds

Unfortunately, I was unable to complete any of the medium level function requirements, I did not have the time to build the functionality to add URLs to a separate database, and therefore could not use it for signature-based detection or add URLs in under 4 seconds.

### Low Priority (Priority 3):

* NOT ACHIEVED:
  + Use the database as training data to improve the accuracy (decrease Mean Absolute Error) of the AI model
  + Return a true positive malicious URL, add the URL to the database, and warn the user in under 3 seconds

Once again, as I was unable to build the database, I did not achieve either of the low priority functional requirements.

## 5.2 Non-Functional Requirements:

### Must (Priority 0):

* ACHIEVED:
  + The web extension will be available as a Firefox Add-on

As mentioned in 4.7, the Add-On is available to download for Firefox.

### High Priority (Priority 1):

* ACHIEVED:
  + Create a minimal and unconfusing UI
* NOT ACHIEVED:
  + The web extension will be available to download as a Chrome Extension

The UI in my app is almost non-existent. Apart from downloading it, the only user interaction required is regular browsing, and closing the warning pop-up when it shows. Apart from that there is no UI to be confusing or maximalist.

As for my second high-priority non-functional requirement, I did not create a version for chrome in time. Although this is one of first goals moving forward with the application.

## 5.3 Reflection and Evaluation:

#### What I did well:

I successfully fulfilled (almost) all the highest priority requirements. The only one I missed ‘Build a web extension that interrupts web requests and alerts the user to a malicious URL with at least 50% accuracy’ wasn’t fulfilled due to the wording of the requirement as explained in 5.1. I have a working Firefox add-on to detect phishing URLs using an AI model that I trained, using data that I sourced and pre-processed.

My first (best performing) model has an accuracy and f1 score of 80% (4.2.3). This is means that the model gets 80% of all its prediction right, which is over the 75% figure, but under that 90% benchmark that I set myself after the second round of research. This being said, 80% accuracy and f1 score is a well performing model for the purpose of a free browser add-on.

The AI model gives the user an answer to the security of the webpage quickly, and in conjunction with that, the pop-up warning appears swiftly too. As can be seen in the demos for my application (4.8), the warning pop-up often appears less than a second after the user first clicks the suspicious link.

I underwent three separate rounds of research through the creation of this application. Through this process I learned a lot about different techniques regarding phishing detection with machine learning. I concluded from my research that the best performing model will use features that are: URL-based (URL length, number of parameters in URL etc.), hyperlink-based (number of hyperlinks in web page, ratio of hyperlinks with different high-level domains to current site to total hyperlinks in web page), and Intra-URL related (relation between the high-level domain and other parts of the URL such as the path, queries, or lower-level domains.) This knowledge I gained was crucial to building a model that was effective a detecting phishes.

#### How I could improve:

One of the biggest improvements I could make to my application is to reduce the number of false positives it returns, looking at the visualised classification report of the first model (4.2.3), we see that the recall rate for safe URLs is about 82%, meaning that when the model is given a safe URL, it predicts it correctly as safe 82% of the time, and about 18% of the time, the model will predict a safe URL as a phish. 18% doesn’t sound that high on paper, but with the frequency the user will be visiting safe sites (let's assume, for every 1000 safe sites a user visits, they visit 1 phishing site), it is crucial to lower this figure dramatically to improve the user experience. This need for a low false positive rate is the main reason I didn’t implement the 2nd model I had trained. Although the model identified phishes more accurately in some contexts, the recall rate for safe URLs sat at about 60%. That means, when provided with a safe URL, the model incorrectly identifies it as phishing 40% of the time, which is huge for our context and simply not acceptable.

An effective way to achieve this, without making any changes to the model, would be to add a URL (or even better, a hostname) whitelist that the app checks when the model detects a phish before serving the user a warning. By also adding the functionality for a user to whitelist domains, that would dramatically improve the experience of the user, as they would deal with false positives at a much scarcer rate. And when they do, they can use it to improve the performance of the model even more. This was a goal when I set out to create the application, but during its development, the training of the model itself became my focus. In hindsight, I think the application would have benefitted from implementing this feature earlier on instead of training a second model.

Another important improvement would be to improve the accuracy of the model. This was my main goal when finishing the second round of research (3.2). I thought that by adding a feature that represents the number of hyperlinks in the webpage, that my model's performance would improve, it did not. One way I could have improved its accuracy was by adding more features to the model.

Another way I could improve the model itself is by increasing the amount of training data, in the paper ‘Machine learning based phishing detection from URLs’ (Sahingoz,Buber,Demir,Diri), they used a dataset of over 70 000 entries. This may be an ideal figure to aim for, as a larger amount of training data makes a more refined model.

Additionally, my model did need some more testing if I wanted to improve it, I should continue comparing classification reports between models with different feature sets. Constantly improving its accuracy, precision, and recall.

To improve my application further, I would also find a way to warn the user without using a pop-up warning. This is because pop-ups are generally not considered good practise. I should consider any browser alert/notification functions I could utilise for this.

Leading from this, I would like to either remove pop-ups (as previously stated), or at least find a way to not have to make the user turn off ‘Block Pop-ups’ in browser settings to let my apps pop-up warning functionality work correctly. As disabling this browser feature may decrease your online security, which is the direct opposite of my applications goal.

Lastly, I should have corresponded with my project supervisor more regularly for guidance. I thought that since my idea for what the app should be, and my method for achieving this was so clear to me, that I could just self-guide myself through the process. Along with the fact that I have been working part-time during the weeks, which reduces the time I can meet with them to a very few instances. And I did successfully reflect, evaluate, and plan changes based on the then current version of my application. An example of this was the three separate rounds of research I conducted in an attempt to improve my model. This being said, it is clear that I may have created an even better app had I rendezvous with my supervisor more frequently. On top of this, my overall project management could be improved. As I spent a lot of time focussing on tasks like creating a new AI model, when if I focussed more on my original aims, I may have implemented the domain whitelist and improved my application.

# 6 Conclusion

Overall, I believe that I successfully accomplished most of the goals I had set out for myself in this project. I have a working prototype, that is available to download for free from the Firefox Add-on store. The browser extension uses a Pythonic AI model to predict phishing URLs, using features extracted from the URL string and the actual webpage. The current model has an accuracy of 80%, and the recall and precision sit at about the same figure. Like my demos (4.8) show, the application can successfully warn the user to websites that have been confirmed to be phishing, and in some cases, warning the user even when googles in built deceptive page warning didn’t, I’m very happy with this result, especially considering the disparity between the resources a computer-science student has versus the huge technology empire of Google. Improvements can be made, however, specifically to reduce the number of false positives the model returns, the accuracy of the model, and altering the pop-up warning system to be more user-friendly and compliant with best practice.

# References & Literature Review:

Agarwal, M. (2023) *Pythonic data cleaning with pandas and NumPy*, *Real Python*. Real Python. Available at: <https://realpython.com/python-data-cleaning-numpy-pandas/>

Babagoli, M., Aghababa, M.P. and Solouk, V. (2018) “Heuristic nonlinear regression strategy for detecting phishing websites,” *Soft Computing*, 23(12), pp. 4315–4327. Available at: <https://doi.org/10.1007/s00500-018-3084-2>.

Basit, A. *et al.* (2020) “A comprehensive survey of AI-enabled phishing attacks detection techniques,” *Telecommunication Systems*, 76(1), pp. 139–154. Available at: <https://doi.org/10.1007/s11235-020-00733-2>.

A survey into phishing detection using AI; provides a literature review of Artificial Intelligence (AI) techniques: Machine Learning, Deep Learning, Hybrid Learning, and Scenario-based techniques for phishing attack detection. Findings include that a portion of machine learning procedures can identify true positive phishing attacks with an accuracy of up to 99%.

Breuss, M. (2023) *Python web applications: Deploy your script as a flask app*, *Real Python*. Real Python. Available at: <https://realpython.com/python-web-applications/> (Accessed: May 4, 2023).

C3 AI (2022) *Precision*, *C3 AI*. Available at: <https://c3.ai/glossary/machine-learning/precision/#:~:text=What%20is%20Precision%3F,the%20number%20of%20false%20positives> (Accessed: May 4, 2023).

*Flask Documentation* (no date) *Welcome to Flask - Flask Documentation (2.3.x)*. Available at: <https://flask.palletsprojects.com/en/2.3.x/> (Accessed: May 4, 2023).

Google Cloud Documentation (no date) Google cloud documentation. Google. Available at: <https://cloud.google.com/docs>

Jupyter Project Documentation (no date) Jupyter Project Documentation - Jupyter Documentation 4.1.1 alpha. Available at: <https://docs.jupyter.org/en/latest/>

Marchal, S. *et al.* (2014) “PhishStorm: Detecting phishing with streaming analytics,” *IEEE Transactions on Network and Service Management*, 11(4), pp. 458–471. Available at: <https://doi.org/10.1109/tnsm.2014.2377295>.

MDN Contributors (2022) *Browser extensions - mozilla: MDN*, *Mozilla*. Available at: <https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions>

Documentation relating to the use of the WebExtensions API cross-browser technology. Contains guides on starting out, to intercepting HTTP requests, to API references.

Kerr, J. (no date) *Pickle - python object serialization*, *Python documentation*. Available at: <https://docs.python.org/3/library/pickle.html>

Mendeley Data. Available at: <https://data.mendeley.com/>

Microsoft 365 Defender Threat Intelligence Team (2021) *AI-driven adaptive protection against human-operated ransomware*, *Microsoft Security Blog*. Available at: <https://www.microsoft.com/en-us/security/blog/2021/11/15/ai-driven-adaptive-protection-against-human-operated-ransomware/>

Blog post outlining Microsoft's use of AI in the fight against ransomware. Windows defender uses an adaptive AI that adjusts its aggressiveness depending on factors like suspicious system calls or task injections.

Noori, R. (2023) What is Agile Project Management: When and how to use it, monday.com Blog. Available at: <https://monday.com/blog/project-management/agile-project-management/>

Pedregosa, F. (2011) Scikit Learn Documentation, scikit. Available at: <https://scikit-learn.org/stable/user_guide.html>

Rawal, S. *et al.* (2017) “Phishing detection in e-mails using machine learning,” *International Journal of Applied Information Systems*, 12(7), pp. 21–24. Available at: <https://doi.org/10.5120/ijais2017451713>.

Describes the problem of identifying phishy emails using a classification problem, and thus tests different types of AI models’ effectiveness. The most effective were Support Vector Machine and Random Forrest classifier.

Salloum, S. *et al.* (2021) “Phishing email detection using Natural Language Processing Techniques: A literature survey,” *Procedia Computer Science*, 189, pp. 19–28. Available at: <https://doi.org/10.1016/j.procs.2021.05.077>.

A journal focusing on the use of natural language processing to detect phishing attacks in email. Survey concludes that more research is needed into the topic, but that NLP could provide a good alternative to just source code classification as it tends to be faster.

Statista Research Department (2022) *Most frequently reported types of cyber crime 2021*, *Statista*. Available at: <https://www.statista.com/statistics/184083/commonly-reported-types-of-cyber-crime/> (Accessed: November 24, 2022).

Statistic showing Phishing to be the most reported type of cybercrime as of 2021.

Sahingoz, O.K. *et al.* (2019) “Machine learning based phishing detection from urls,” *Expert Systems with Applications*, 117, pp. 345–357. Available at: <https://doi.org/10.1016/j.eswa.2018.09.029>.

Valley, N. (2019) *Deploy a Python machine learning model as a web service*, *IBM developer*. Available at: <https://developer.ibm.com/tutorials/deploy-a-python-machine-learning-model-as-a-web-service/> (Accessed: November 24, 2022).

IBM tutorial outlining the process of connecting a Python machine learning model to a web service.

Visual studio code – code editor: Microsoft Azure (no date) – Code Editor | Microsoft Azure. Available at: <https://azure.microsoft.com/en-us/products/visual-studio-code/>