Configuration Manual

MSc Research Project
MSc. Cybersecurity

Yamah Hanson Shonibare
Student ID: 21106941

School of Computing
National College of Ireland

Supervisor: Jawad Salahuddin

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Yamah Hanson Shonibare		
Student ID:	21106941		
Programme:	MSc. Cybersecurity	**Year:**	2022
Module:	Research Project		
Lecturer:	Jawad Salahuddin		
Submission Due Date:	December 15, 2022		
Project Title:	Detecting Spear-phishing Attacks using Machine Learning		
Word Count:	………1337…..……… **Page Count:** …………20……………..		

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Yamah Hanson Shonibare ……………………………………………

Date: December 15, 2022……………………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

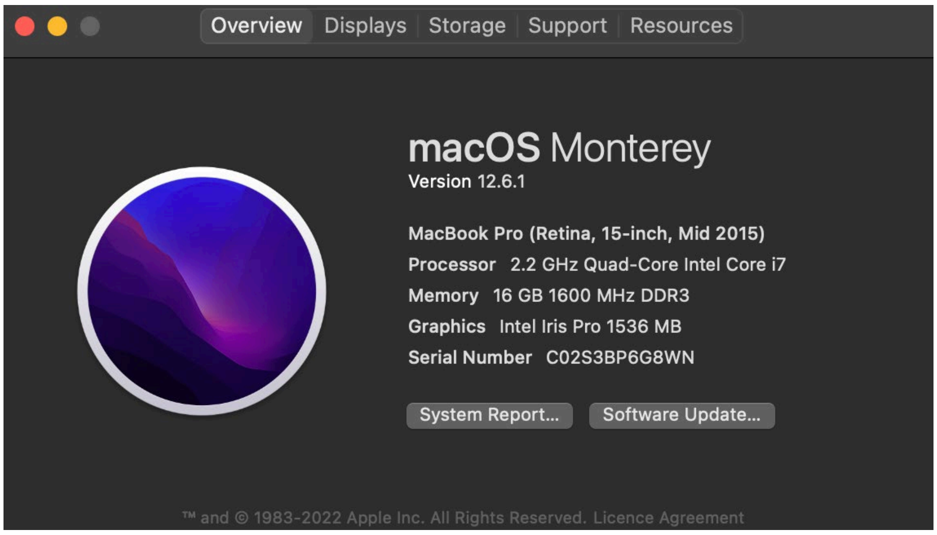
Yamah Hanson Shonibare
21106941

1 Introduction

The hardware requirements and software setups, procedures for gathering, organizing, and pre-processing data, as well as the whole project implementation, are all covered in detail in this configuration. The project's goal was to use machine learning to detect spear-phishing attacks.

The technical specifications and procedures listed below lead to the project's outcomes.

System Configuration



macOS Monterey
Version 12.6.1

MacBook Pro (Retina, 15-inch, Mid 2015)
Processor 2.2 GHz Quad-Core Intel Core i7
Memory 16 GB 1600 MHz DDR3
Graphics Intel Iris Pro 1536 MB
Serial Number C02S3BP6G8WN

™ and © 1983-2022 Apple Inc. All Rights Reserved. Licence Agreement

Figure 1: System Configuration

This project was carried out using a PC with a 2.2GHz quad-core Intel Core i7 processor, 16GB of RAM, and a 1TB hard drive running macOS Monterey.

Environment Setup

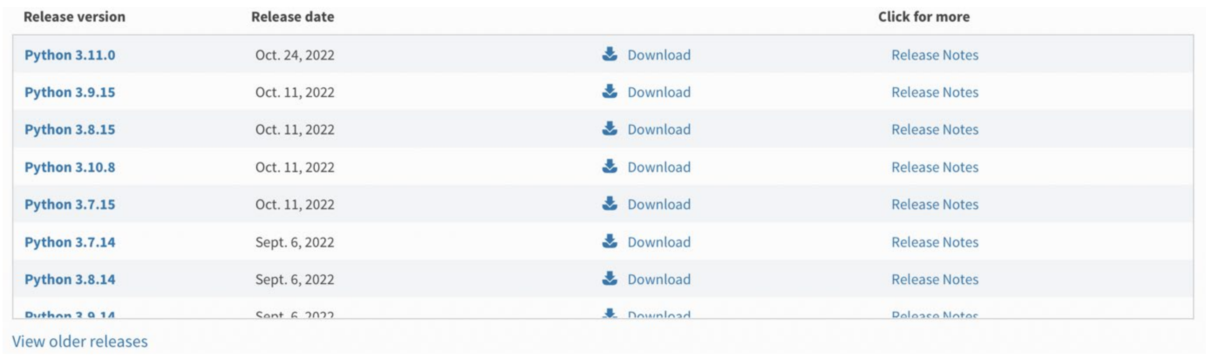
The software setup needed to run the project includes:

1. Anaconda IDE
2. Jupyter Notebook
3. Python

For the implementation of this project, Python was chosen as the programming language. And Jupyter Notebook in Anaconda was also used in all phases of this project including data pre-processing, model training, testing, and assessment.

Python

The latest version of python was downloaded and installed from the official website[1]



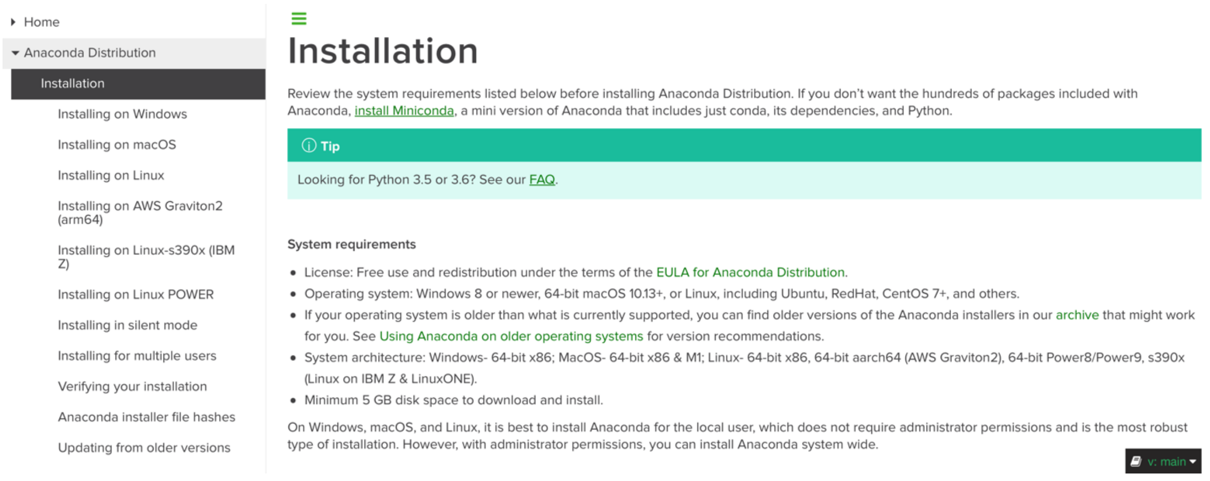
Release version	Release date		Click for more
Python 3.11.0	Oct. 24, 2022	Download	Release Notes
Python 3.9.15	Oct. 11, 2022	Download	Release Notes
Python 3.8.15	Oct. 11, 2022	Download	Release Notes
Python 3.10.8	Oct. 11, 2022	Download	Release Notes
Python 3.7.15	Oct. 11, 2022	Download	Release Notes
Python 3.7.14	Sept. 6, 2022	Download	Release Notes
Python 3.8.14	Sept. 6, 2022	Download	Release Notes
Python 3.9.14	Sept. 6, 2022	Download	Release Notes

View older releases

Figure 2: Python download

1.1 Anaconda Individual Edition

As Anaconda already has the Jupyter Notebook pre-installed, the next step is to download Anaconda from its official website. The minimal system requirements and instructions for downloading and installing Anaconda can be found on the anaconda distribution and installation page[2].



- Home
- Anaconda Distribution
  - Installation
    - Installing on Windows
    - Installing on macOS
    - Installing on Linux
    - Installing on AWS Graviton2 (arm64)
    - Installing on Linux-s390x (IBM Z)
    - Installing on Linux POWER
    - Installing in silent mode
    - Installing for multiple users
    - Verifying your installation
    - Anaconda installer file hashes
    - Updating from older versions

Installation

Review the system requirements listed below before installing Anaconda Distribution. If you don't want the hundreds of packages included with Anaconda, install Miniconda, a mini version of Anaconda that includes just conda, its dependencies, and Python.

Tip

Looking for Python 3.5 or 3.6? See our FAQ.

System requirements

- License: Free use and redistribution under the terms of the EULA for Anaconda Distribution.
- Operating system: Windows 8 or newer, 64-bit macOS 10.13+, or Linux, including Ubuntu, RedHat, CentOS 7+, and others.
- If your operating system is older than what is currently supported, you can find older versions of the Anaconda installers in our archive that might work for you. See Using Anaconda on older operating systems for version recommendations.
- System architecture: Windows- 64-bit x86; MacOS- 64-bit x86 & M1; Linux- 64-bit x86, 64-bit aarch64 (AWS Graviton2), 64-bit Power8/Power9, s390x (Linux on IBM Z & LinuxONE).
- Minimum 5 GB disk space to download and install.

On Windows, macOS, and Linux, it is best to install Anaconda for the local user, which does not require administrator permissions and is the most robust type of installation. However, with administrator permissions, you can install Anaconda system wide.

Figure 3: Anaconda Documentation

Upon the completion of installation, Jupyter Notebook can be launched from within Anaconda by clicking the Jupyter Notebook icon and the Anaconda Navigator symbol, respectively. Below is a snapshot of the process.

[1] https://www.python.org/downloads/
[2] https://docs.anaconda.com/anaconda/navigator/install/

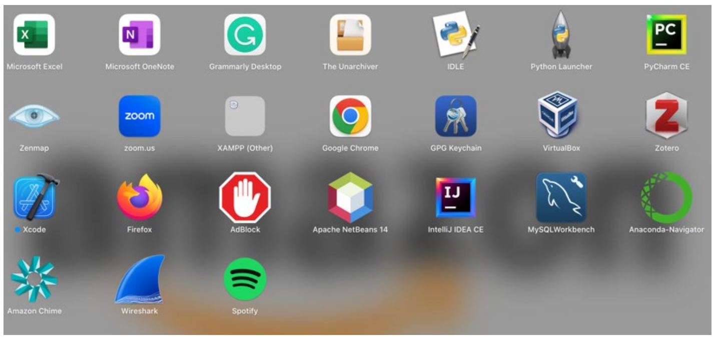
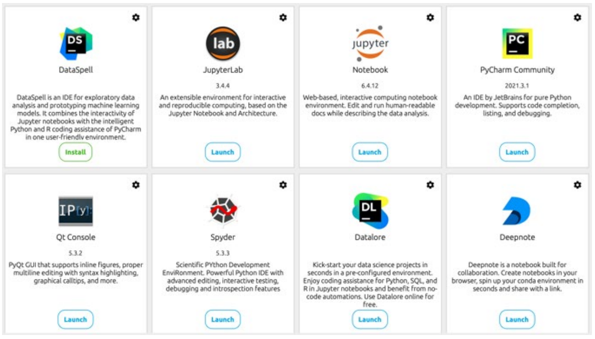


Figure 4: Launch Anaconda Navigator



DataSpell

DataSpell is an IDE for exploratory data analysis and prototyping machine learning models. It combines the interactivity of Jupyter notebooks with the intelligent Python and R coding assistance of PyCharm in one user-friendly environment.

JupyterLab

3.4.4

An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Notebook

6.4.12

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

PyCharm Community

2021.3.1

An IDE by JetBrains for pure Python development. Supports code completion, listing, and debugging.

Qt Console

5.3.2

PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Spyder

5.3.3

Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

Datalore

Kick-start your data science projects in seconds in a pre-configured environment. Enjoy coding assistance for Python, SQL, and R in Jupyter notebooks and benefit from no-code automations. Use Datalore online for free.

Deepnote

Deepnote is a notebook built for collaboration. Create notebooks in your browser, spin up your conda environment in seconds and share with a link.

Figure 5: Launch Jupyter Notebook

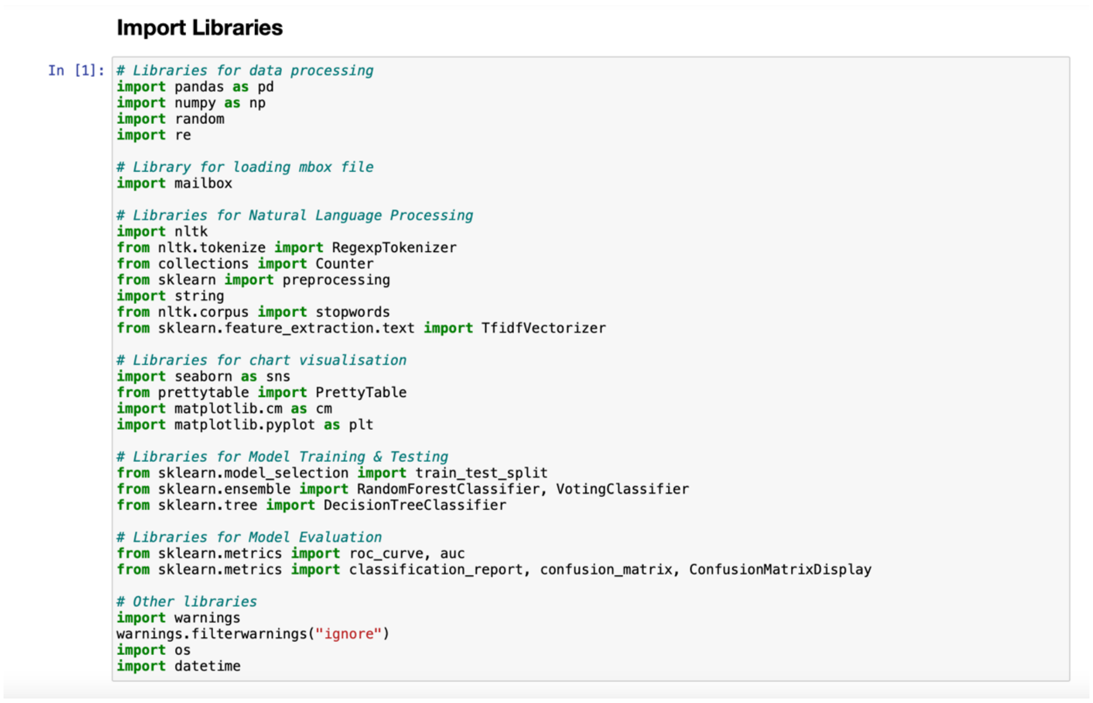
2 Data Collection

The dataset utilized in this study was put together and created by the CALO (Cognitive Assistant that Learns and Organizes) Project which contains emails that the Federal Energy Regulatory Commission (FERC) initially made publicly accessible online as part of an investigation[3].

3 Pre-processing

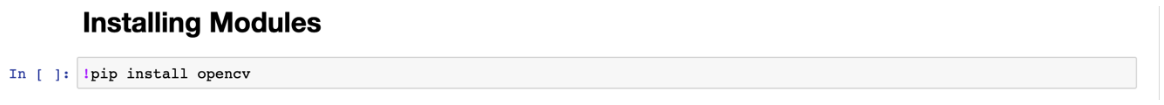
To prepare the data for modeling, it is crucial to pre-process it after downloading the dataset. As a result, three key pre-processing procedures were completed. These actions were all carried out in the "spearphishing.ipynb" file of the same Jupyter notebook. The first step would be to import the required packages as depicted in Figure 6 in order to execute the code. As seen in Figure 7 below, several packages that have not yet been installed on the Anaconda environment can be installed using the "!pip install module name>" command from within Jupyter Notebook:

[3] https://www.cs.cmu.edu/~enron/


Import Libraries

In [1]:
```
# Libraries for data processing
import pandas as pd
import numpy as np
import random
import re

# Library for loading mbox file
import mailbox

# Libraries for Natural Language Processing
import nltk
from nltk.tokenize import RegexpTokenizer
from collections import Counter
from sklearn import preprocessing
import string
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer

# Libraries for chart visualisation
import seaborn as sns
from prettytable import PrettyTable
import matplotlib.cm as cm
import matplotlib.pyplot as plt

# Libraries for Model Training & Testing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.tree import DecisionTreeClassifier

# Libraries for Model Evaluation
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay

# Other libraries
import warnings
warnings.filterwarnings("ignore")
import os
import datetime
```

Figure 6: Import Modules/ Dependencies

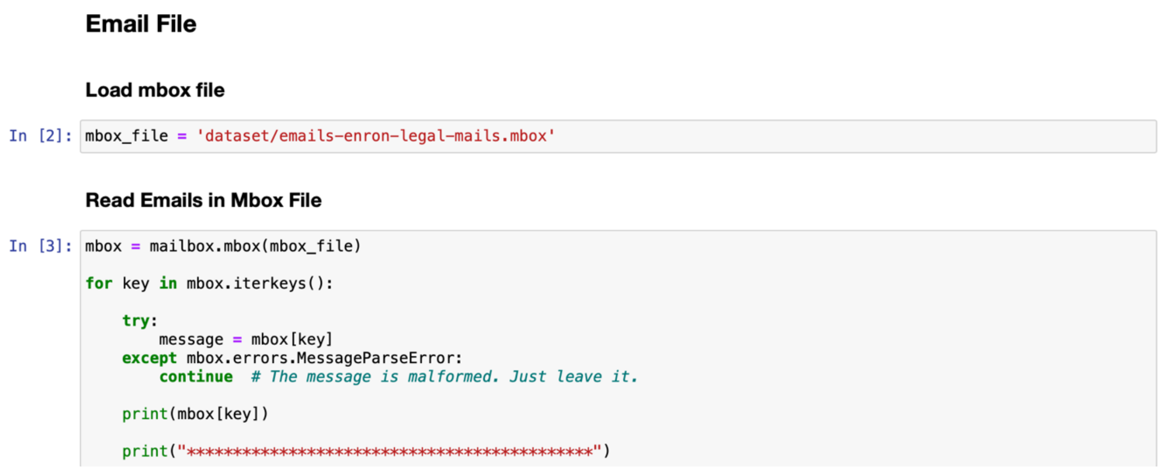

Installing Modules

In [ ]:
```
!pip install opencv
```

Figure 7: Pip Install Modules

Data Organization and Exploration

The dataset is programmatically loaded from the mbox file saved in the PC and subsequently, the emails saved in a CSV file are loaded into the environment (and the first five (5) emails are displayed) for testing, training, and validation as shown in figures 8 and 9 below.


Email File

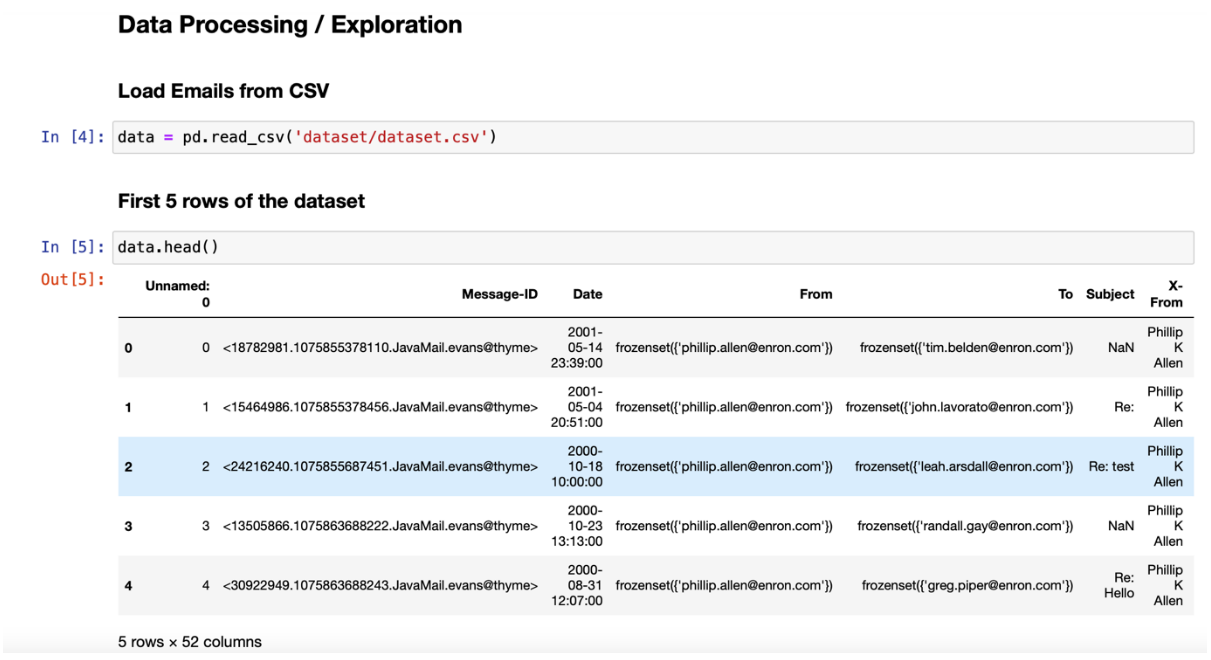
Load mbox file

In [2]:
```
mbox_file = 'dataset/emails-enron-legal-mails.mbox'
```

Read Emails in Mbox File

In [3]:
```
mbox = mailbox.mbox(mbox_file)

for key in mbox.iterkeys():

    try:
        message = mbox[key]
    except mbox.errors.MessageParseError:
        continue  # The message is malformed. Just leave it.

    print(mbox[key])

    print("************************************************")
```

Figure 8: Load Email Files



Data Processing / Exploration

Load Emails from CSV

```
In [4]: data = pd.read_csv('dataset/dataset.csv')
```

First 5 rows of the dataset

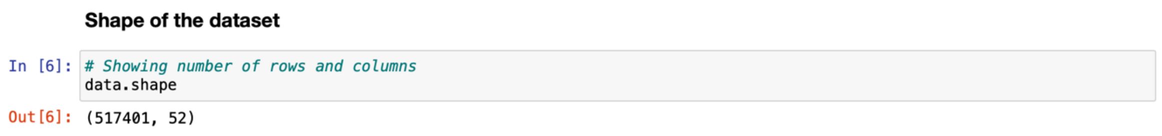
```
In [5]: data.head()
Out[5]:
```

	Unnamed: 0	Message-ID	Date	From	To	Subject	X-From
0	0	<18782981.1075855378110.JavaMail.evans@thyme>	2001-05-14 23:39:00	frozenset({'phillip.allen@enron.com'})	frozenset({'tim.belden@enron.com'})	NaN	Phillip K Allen
1	1	<15464986.1075855378456.JavaMail.evans@thyme>	2001-05-04 20:51:00	frozenset({'phillip.allen@enron.com'})	frozenset({'john.lavorato@enron.com'})	Re:	Phillip K Allen
2	2	<24216240.1075855687451.JavaMail.evans@thyme>	2000-10-18 10:00:00	frozenset({'phillip.allen@enron.com'})	frozenset({'leah.arsdall@enron.com'})	Re: test	Phillip K Allen
3	3	<13505866.1075863688222.JavaMail.evans@thyme>	2000-10-23 13:13:00	frozenset({'phillip.allen@enron.com'})	frozenset({'randall.gay@enron.com'})	NaN	Phillip K Allen
4	4	<30922949.1075863688243.JavaMail.evans@thyme>	2000-08-31 12:07:00	frozenset({'phillip.allen@enron.com'})	frozenset({'greg.piper@enron.com'})	Re: Hello	Phillip K Allen

5 rows × 52 columns

Figure 9: Data Processing/Exploration

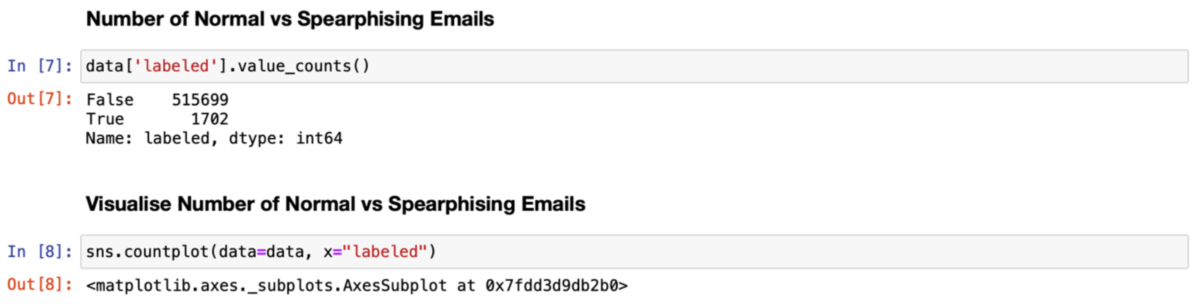
The next step is to check the shape of the dataset to ascertain the total number of emails within the dataset. The number of spear-phishing emails and the number of normal emails was also determined as shown in figures 10 and 11 below.



Shape of the dataset

```
In [6]: # Showing number of rows and columns
        data.shape
Out[6]: (517401, 52)
```

Figure 10: Shape of the Dataset



Number of Normal vs Spearphising Emails

```
In [7]: data['labeled'].value_counts()
Out[7]: False    515699
        True       1702
        Name: labeled, dtype: int64
```

Visualise Number of Normal vs Spearphising Emails

```
In [8]: sns.countplot(data=data, x="labeled")
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x7fdd3d9db2b0>
```

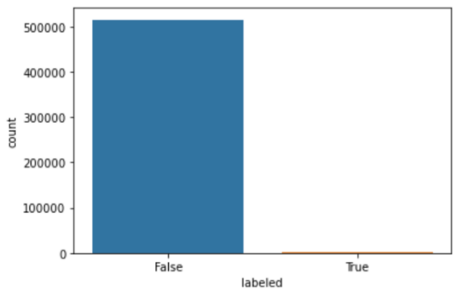
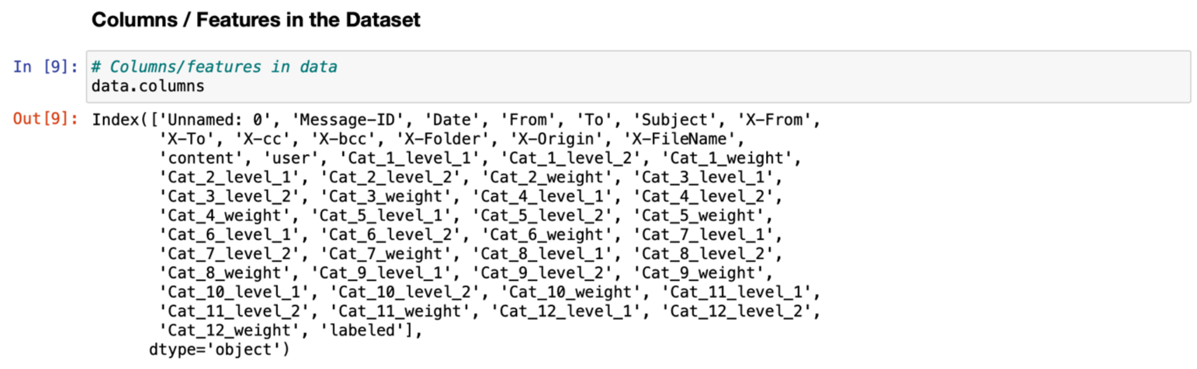


Figure 11: Visualization of Normal and Spear-phishing emails

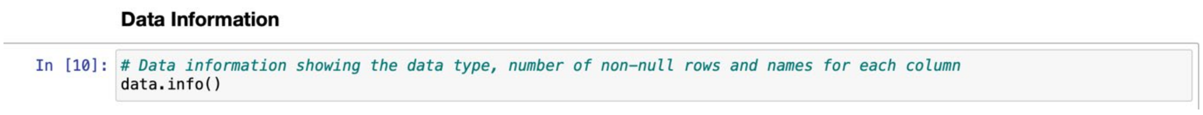
Subsequently, the columns representing all of the features within the dataset are selected (Figure 12) and the dataset is selected for its varying type of data types as well as the number of null rows and the number of non-null rows as shown in Figures 13 and 14 below. Figure 15 showed columns that were empty were dropped.



Columns / Features in the Dataset

```
In [9]:  # Columns/features in data
         data.columns

Out[9]:  Index(['Unnamed: 0', 'Message-ID', 'Date', 'From', 'To', 'Subject', 'X-From',
                'X-To', 'X-cc', 'X-bcc', 'X-Folder', 'X-Origin', 'X-FileName',
                'content', 'user', 'Cat_1_level_1', 'Cat_1_level_2', 'Cat_1_weight',
                'Cat_2_level_1', 'Cat_2_level_2', 'Cat_2_weight', 'Cat_3_level_1',
                'Cat_3_level_2', 'Cat_3_weight', 'Cat_4_level_1', 'Cat_4_level_2',
                'Cat_4_weight', 'Cat_5_level_1', 'Cat_5_level_2', 'Cat_5_weight',
                'Cat_6_level_1', 'Cat_6_level_2', 'Cat_6_weight', 'Cat_7_level_1',
                'Cat_7_level_2', 'Cat_7_weight', 'Cat_8_level_1', 'Cat_8_level_2',
                'Cat_8_weight', 'Cat_9_level_1', 'Cat_9_level_2', 'Cat_9_weight',
                'Cat_10_level_1', 'Cat_10_level_2', 'Cat_10_weight', 'Cat_11_level_1',
                'Cat_11_level_2', 'Cat_11_weight', 'Cat_12_level_1', 'Cat_12_level_2',
                'Cat_12_weight', 'labeled'],
               dtype='object')
```

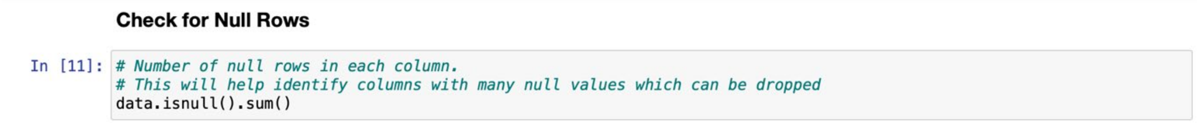
Figure 12: Columns/Features in the Dataset



Data Information

```
In [10]:  # Data information showing the data type, number of non-null rows and names for each column
          data.info()
```

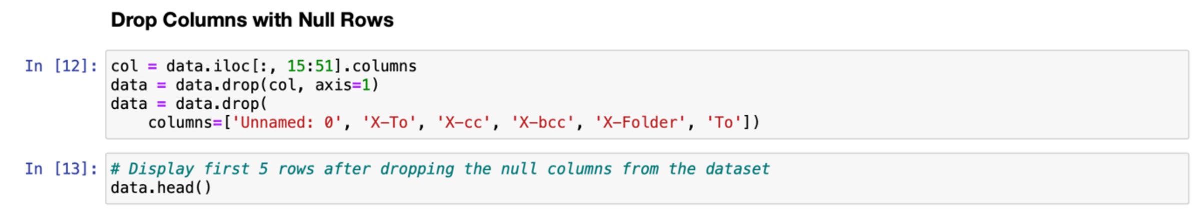
Figure 13: Data Information



Check for Null Rows

```
In [11]:  # Number of null rows in each column.
          # This will help identify columns with many null values which can be dropped
          data.isnull().sum()
```

Figure 14: Checking for Null and Non-Null Row



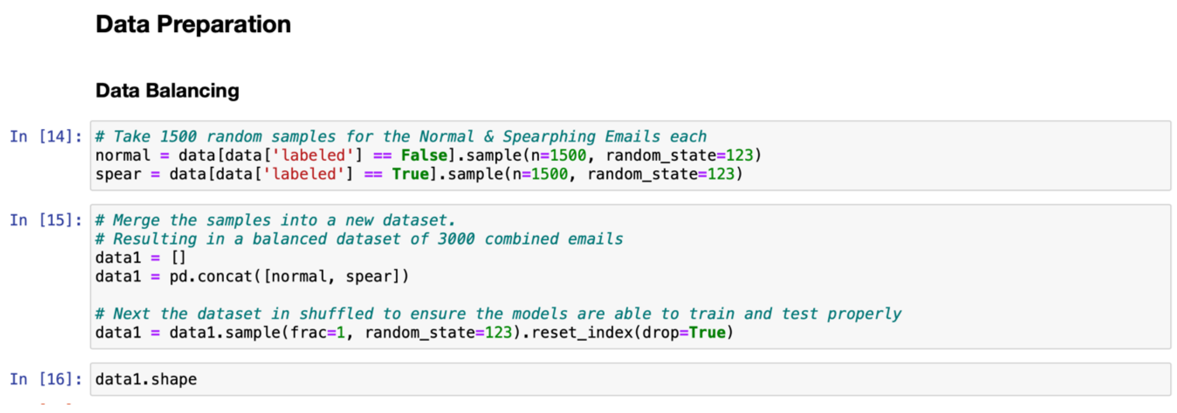
Drop Columns with Null Rows

```
In [12]:  col = data.iloc[:, 15:51].columns
          data = data.drop(col, axis=1)
          data = data.drop(
              columns=['Unnamed: 0', 'X-To', 'X-cc', 'X-bcc', 'X-Folder', 'To'])

In [13]:  # Display first 5 rows after dropping the null columns from the dataset
          data.head()
```

Figure 15: Dropping of Null Rows

Data Preparation

Data for this project was prepared by taking 1500 random samples of spear-phishing emails and 1500 normal emails resulting in a balanced dataset of 3000 emails to form a new dataset (data1). The dataset is shuffled to ensure randomness when picked for training and testing (Figure 16). Figure 17 shows a visual representation of the balanced data set and Figure 18 showed the removal columns that would not be used.

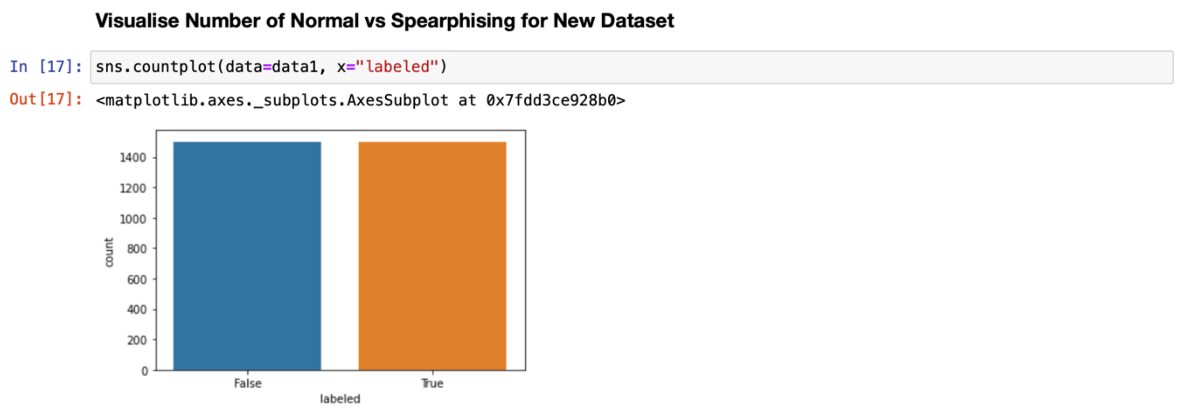


Data Preparation

Data Balancing

```
In [14]: # Take 1500 random samples for the Normal & Spearphing Emails each
         normal = data[data['labeled'] == False].sample(n=1500, random_state=123)
         spear = data[data['labeled'] == True].sample(n=1500, random_state=123)

In [15]: # Merge the samples into a new dataset.
         # Resulting in a balanced dataset of 3000 combined emails
         data1 = []
         data1 = pd.concat([normal, spear])

         # Next the dataset in shuffled to ensure the models are able to train and test properly
         data1 = data1.sample(frac=1, random_state=123).reset_index(drop=True)

In [16]: data1.shape
```

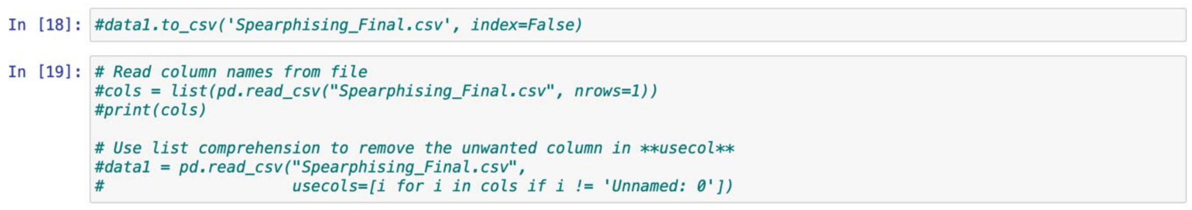
Figure 16: Balancing of Data



Visualise Number of Normal vs Spearphising for New Dataset

```
In [17]: sns.countplot(data=data1, x="labeled")

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x7fdd3ce928b0>
```

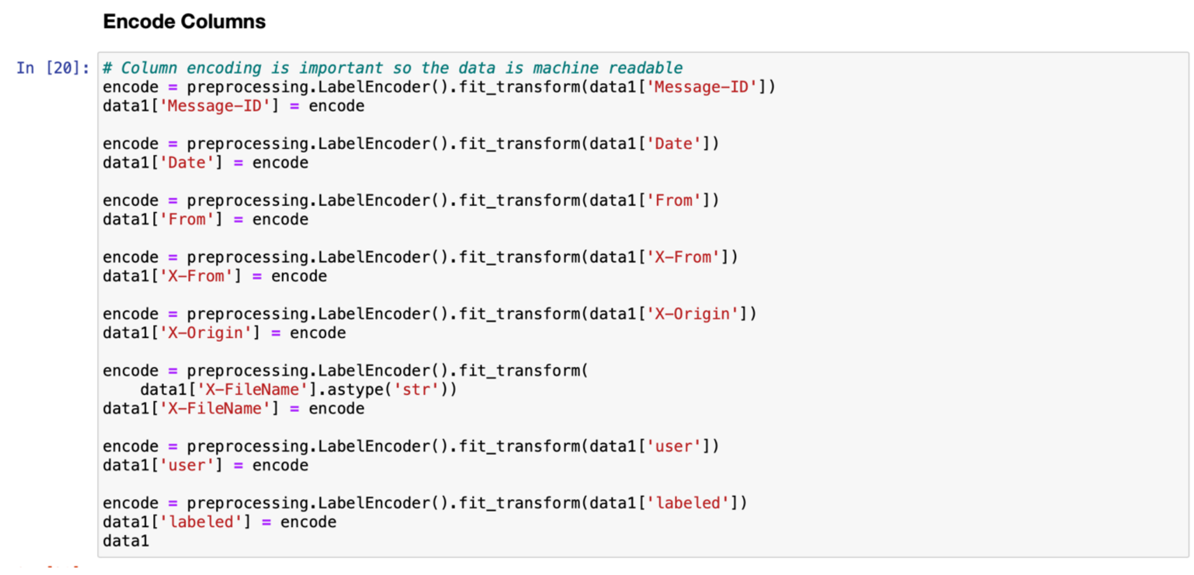
Figure 17: Visual representation of the balanced data



```
In [18]: #data1.to_csv('Spearphising_Final.csv', index=False)

In [19]: # Read column names from file
         #cols = list(pd.read_csv("Spearphising_Final.csv", nrows=1))
         #print(cols)

         # Use list comprehension to remove the unwanted column in **usecol**
         #data1 = pd.read_csv("Spearphising_Final.csv",
         #                    usecols=[i for i in cols if i != 'Unnamed: 0'])
```

Figure 18: Removal of unwanted Columns

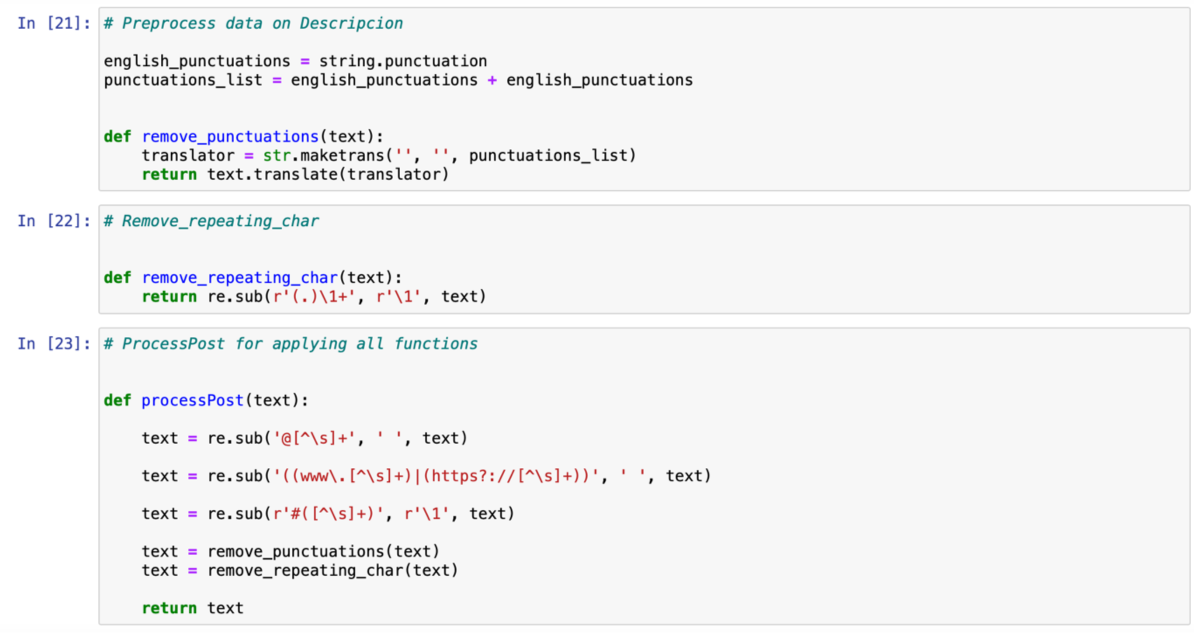
As a part of the data processing, encoding is carried out which involves turning the columns from non-numeric to numeric numbers so that they can be read by the machine learning model. If the columns are not in numbers it will result in an error. The new dataset is subsequently pre-processed by removing punctuations and repeating characters and the processPost is applied to the subject and content of the new dataset. The subject and content of the emails were tokenized and scanned for stopwords which were taken out to ensure they are machine readable. After that, a list of the top 15 words in the emails was selected and a visual display was presented using a rainbow color format.



Encode Columns

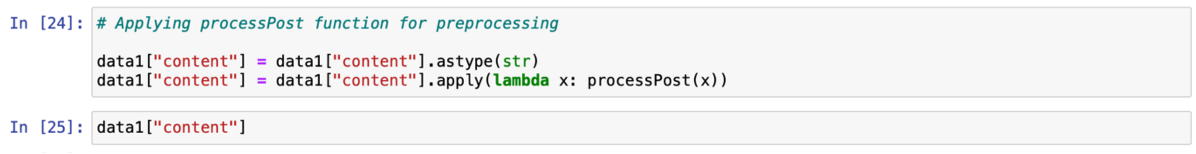
```
In [20]: # Column encoding is important so the data is machine readable
         encode = preprocessing.LabelEncoder().fit_transform(data1['Message-ID'])
         data1['Message-ID'] = encode

         encode = preprocessing.LabelEncoder().fit_transform(data1['Date'])
         data1['Date'] = encode

         encode = preprocessing.LabelEncoder().fit_transform(data1['From'])
         data1['From'] = encode

         encode = preprocessing.LabelEncoder().fit_transform(data1['X-From'])
         data1['X-From'] = encode

         encode = preprocessing.LabelEncoder().fit_transform(data1['X-Origin'])
         data1['X-Origin'] = encode

         encode = preprocessing.LabelEncoder().fit_transform(
             data1['X-FileName'].astype('str'))
         data1['X-FileName'] = encode

         encode = preprocessing.LabelEncoder().fit_transform(data1['user'])
         data1['user'] = encode

         encode = preprocessing.LabelEncoder().fit_transform(data1['labeled'])
         data1['labeled'] = encode
         data1
```

Figure 19: Encoding of Columns



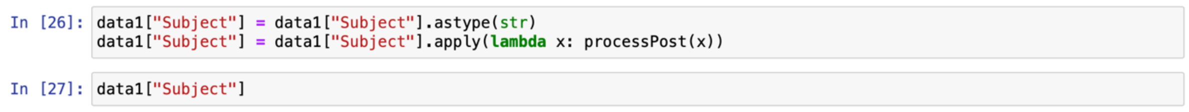
```
In [21]: # Preprocess data on Descripcion

         english_punctuations = string.punctuation
         punctuations_list = english_punctuations + english_punctuations


         def remove_punctuations(text):
             translator = str.maketrans('', '', punctuations_list)
             return text.translate(translator)

In [22]: # Remove_repeating_char


         def remove_repeating_char(text):
             return re.sub(r'(.)\1+', r'\1', text)

In [23]: # ProcessPost for applying all functions


         def processPost(text):

             text = re.sub('@[^\s]+', ' ', text)

             text = re.sub('((www\.[^\s]+)|(https?://[^\s]+))', ' ', text)

             text = re.sub(r'#([^\s]+)', r'\1', text)

             text = remove_punctuations(text)
             text = remove_repeating_char(text)

             return text
```

Figure 20: Pre-processing of Data



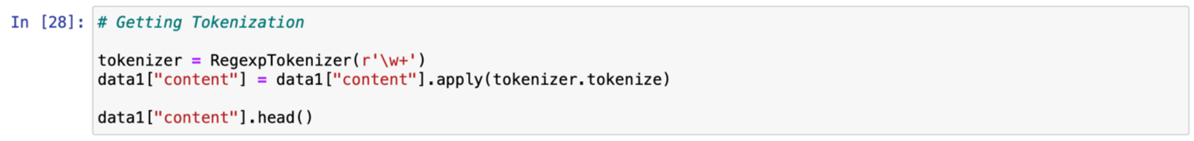
```
In [24]: # Applying processPost function for preprocessing

         data1["content"] = data1["content"].astype(str)
         data1["content"] = data1["content"].apply(lambda x: processPost(x))

In [25]: data1["content"]
```

Figure 21: Application of the pre-processed data in Content



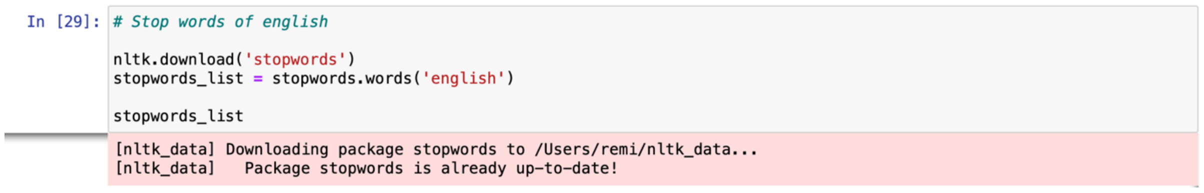
```
In [26]: data1["Subject"] = data1["Subject"].astype(str)
         data1["Subject"] = data1["Subject"].apply(lambda x: processPost(x))

In [27]: data1["Subject"]
```

Figure 22: Application of the pre-processed data in Subject



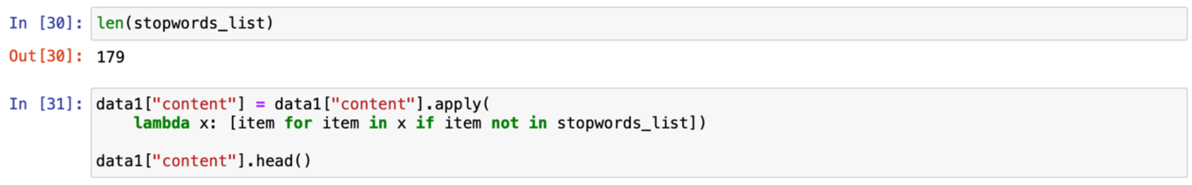
```
In [28]: # Getting Tokenization

         tokenizer = RegexpTokenizer(r'\w+')
         data1["content"] = data1["content"].apply(tokenizer.tokenize)

         data1["content"].head()
```

Figure 23: Tokenization



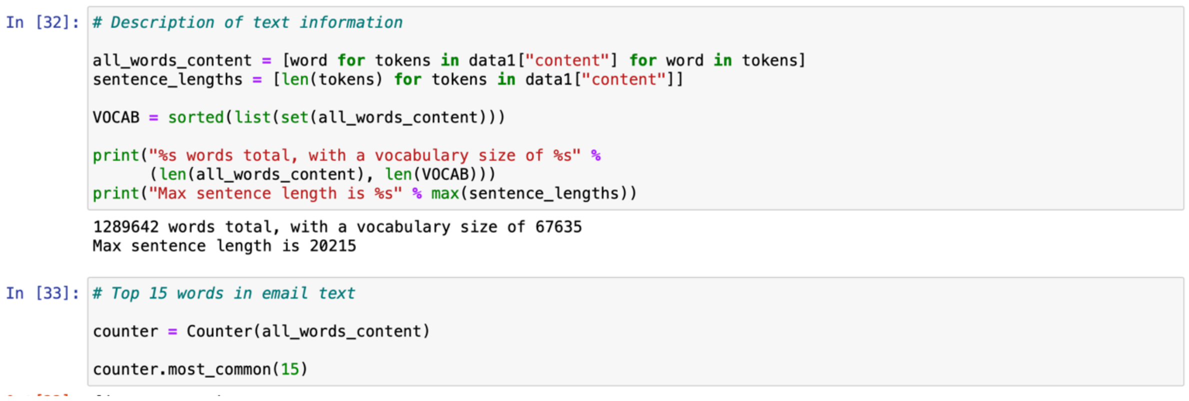
```
In [29]: # Stop words of english

         nltk.download('stopwords')
         stopwords_list = stopwords.words('english')

         stopwords_list

         [nltk_data] Downloading package stopwords to /Users/remi/nltk_data...
         [nltk_data]   Package stopwords is already up-to-date!
```

Figure 24: Stopwords using NLTK



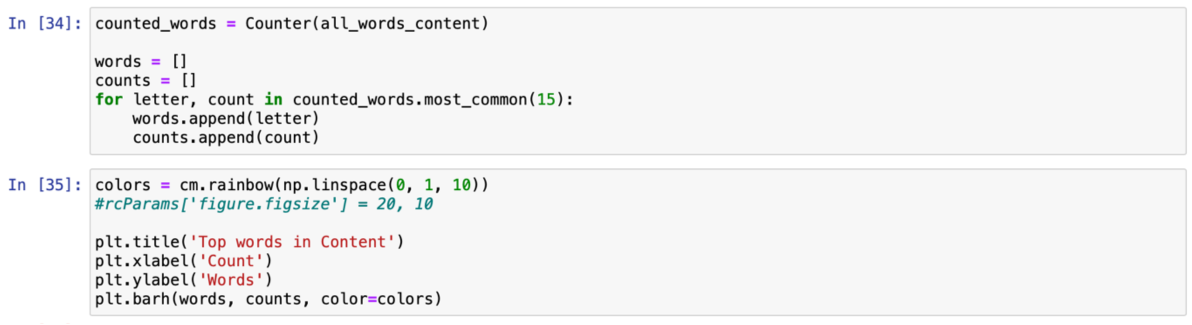
```
In [30]: len(stopwords_list)
Out[30]: 179

In [31]: data1["content"] = data1["content"].apply(
             lambda x: [item for item in x if item not in stopwords_list])

         data1["content"].head()
```

Figure 25: Length of Stopwords



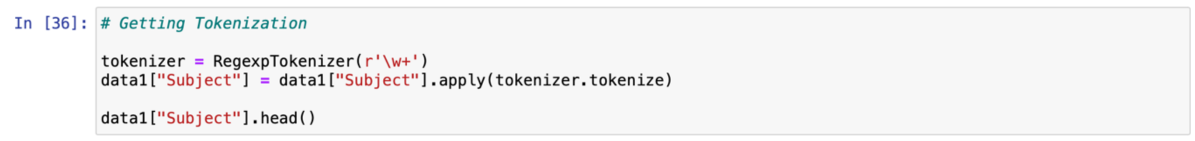
```
In [32]: # Description of text information

         all_words_content = [word for tokens in data1["content"] for word in tokens]
         sentence_lengths = [len(tokens) for tokens in data1["content"]]

         VOCAB = sorted(list(set(all_words_content)))

         print("%s words total, with a vocabulary size of %s" %
               (len(all_words_content), len(VOCAB)))
         print("Max sentence length is %s" % max(sentence_lengths))

         1289642 words total, with a vocabulary size of 67635
         Max sentence length is 20215

In [33]: # Top 15 words in email text

         counter = Counter(all_words_content)

         counter.most_common(15)
```

Figure 26: List of top 15 words in the email text



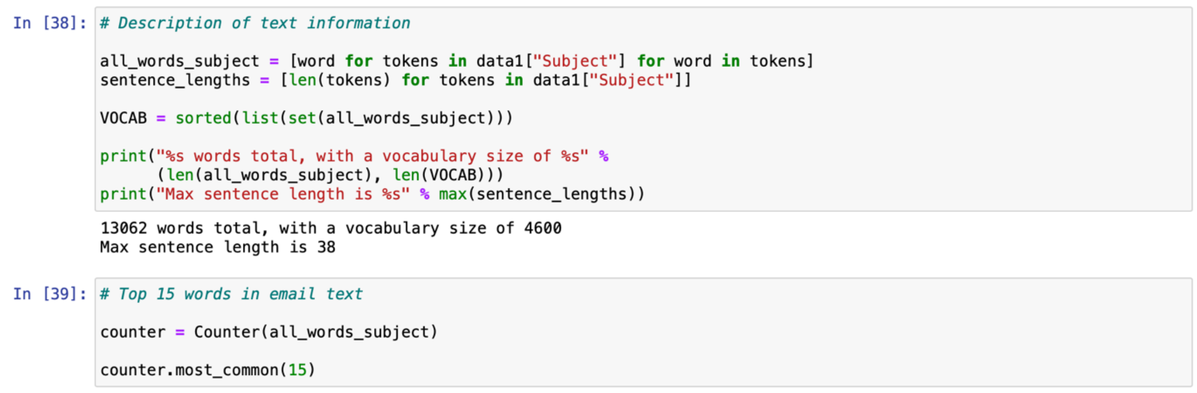
```
In [34]: counted_words = Counter(all_words_content)

         words = []
         counts = []
         for letter, count in counted_words.most_common(15):
             words.append(letter)
             counts.append(count)

In [35]: colors = cm.rainbow(np.linspace(0, 1, 10))
         #rcParams['figure.figsize'] = 20, 10

         plt.title('Top words in Content')
         plt.xlabel('Count')
         plt.ylabel('Words')
         plt.barh(words, counts, color=colors)
```

Figure 27: Visual display of top words in the content



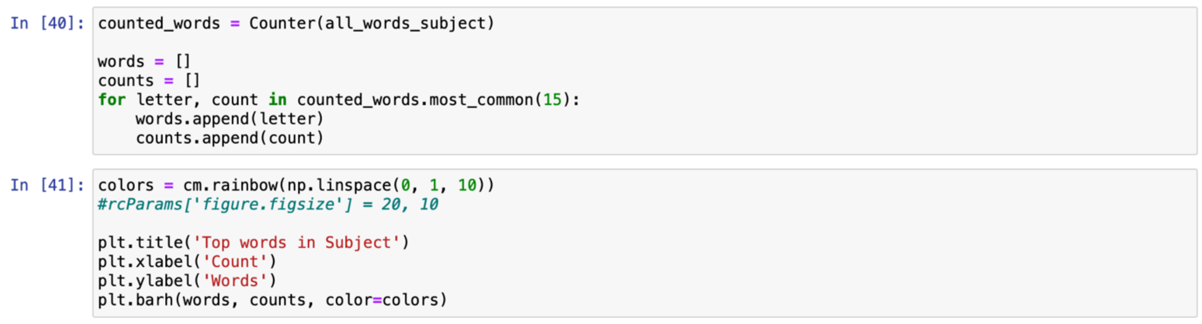
```
In [36]: # Getting Tokenization

         tokenizer = RegexpTokenizer(r'\w+')
         data1["Subject"] = data1["Subject"].apply(tokenizer.tokenize)

         data1["Subject"].head()
```

Figure 28: Tokenization of data in the subject



```
In [38]: # Description of text information

         all_words_subject = [word for tokens in data1["Subject"] for word in tokens]
         sentence_lengths = [len(tokens) for tokens in data1["Subject"]]

         VOCAB = sorted(list(set(all_words_subject)))

         print("%s words total, with a vocabulary size of %s" %
               (len(all_words_subject), len(VOCAB)))
         print("Max sentence length is %s" % max(sentence_lengths))

         13062 words total, with a vocabulary size of 4600
         Max sentence length is 38

In [39]: # Top 15 words in email text

         counter = Counter(all_words_subject)

         counter.most_common(15)
```

Figure 29: List of top 15 words in the subject

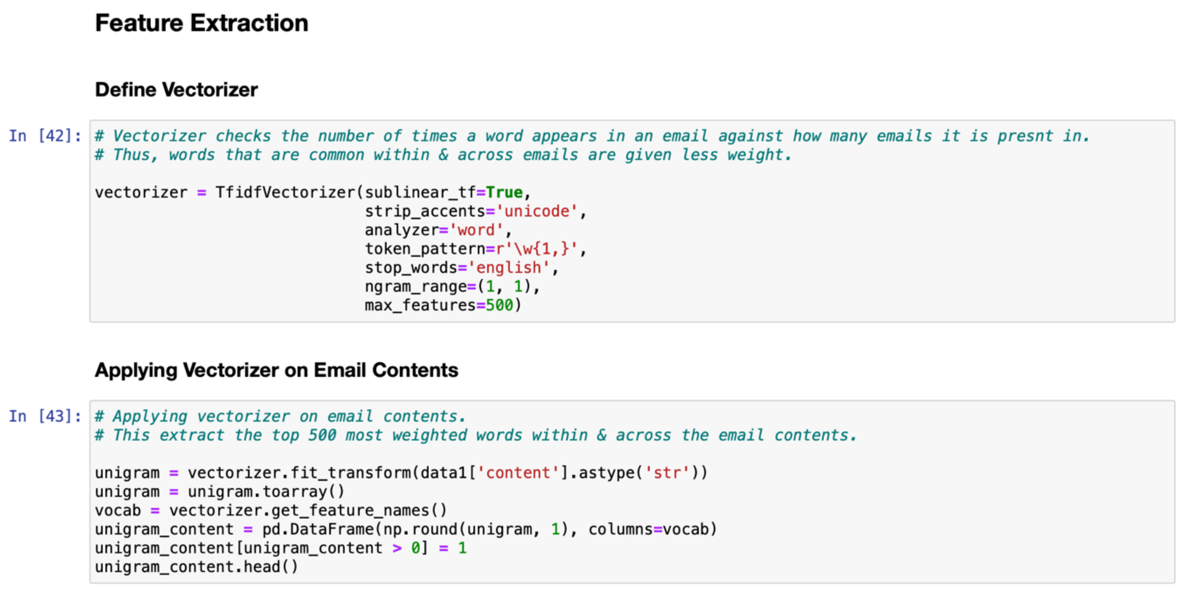


```
In [40]: counted_words = Counter(all_words_subject)

         words = []
         counts = []
         for letter, count in counted_words.most_common(15):
             words.append(letter)
             counts.append(count)

In [41]: colors = cm.rainbow(np.linspace(0, 1, 10))
         #rcParams['figure.figsize'] = 20, 10

         plt.title('Top words in Subject')
         plt.xlabel('Count')
         plt.ylabel('Words')
         plt.barh(words, counts, color=colors)
```

Figure 30: Visual display of the top words in the subject

Feature Extraction

Next, we carried out feature extraction on the new dataset through the application of vectorization on the subject and content of emails. This involves the reduction of redundant data by checking the number of times a word appears in an email against the number of emails it appears in. Words that appear often within and across several emails are given less weight in comparison to words that are not common in emails. Here the top 500 most weighted words within the email content and body are extracted and compared with contents within the rest of the emails. To increase the speed of learning and the accuracy of the model, the email and subject column contents are dropped after they have been vectorized and the vectorized features are subsequently concatenated with the new dataset.



Feature Extraction

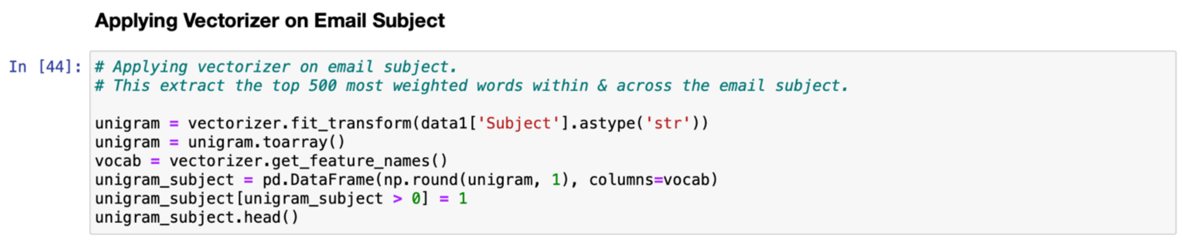
Define Vectorizer

```
In [42]: # Vectorizer checks the number of times a word appears in an email against how many emails it is presnt in.
         # Thus, words that are common within & across emails are given less weight.

         vectorizer = TfidfVectorizer(sublinear_tf=True,
                                      strip_accents='unicode',
                                      analyzer='word',
                                      token_pattern=r'\w{1,}',
                                      stop_words='english',
                                      ngram_range=(1, 1),
                                      max_features=500)
```

Applying Vectorizer on Email Contents

```
In [43]: # Applying vectorizer on email contents.
         # This extract the top 500 most weighted words within & across the email contents.

         unigram = vectorizer.fit_transform(data1['content'].astype('str'))
         unigram = unigram.toarray()
         vocab = vectorizer.get_feature_names()
         unigram_content = pd.DataFrame(np.round(unigram, 1), columns=vocab)
         unigram_content[unigram_content > 0] = 1
         unigram_content.head()
```

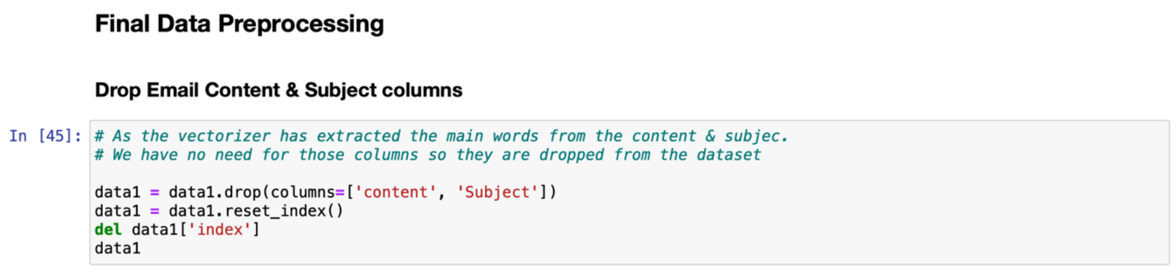
Figure 31: Vectorizer Defined and applied to Email Contents



Applying Vectorizer on Email Subject

```
In [44]: # Applying vectorizer on email subject.
         # This extract the top 500 most weighted words within & across the email subject.

         unigram = vectorizer.fit_transform(data1['Subject'].astype('str'))
         unigram = unigram.toarray()
         vocab = vectorizer.get_feature_names()
         unigram_subject = pd.DataFrame(np.round(unigram, 1), columns=vocab)
         unigram_subject[unigram_subject > 0] = 1
         unigram_subject.head()
```

Figure 32: Applying Vectorizer on Email Subjects

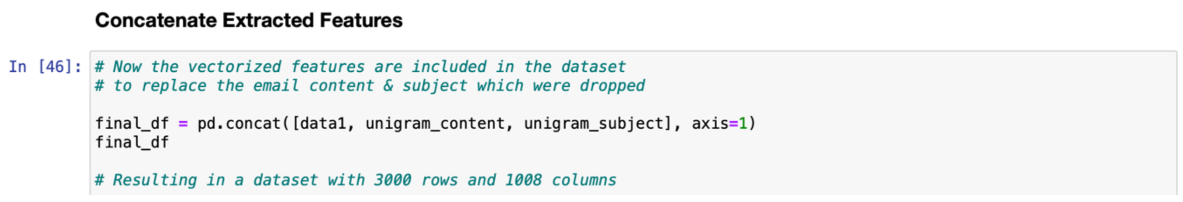


Final Data Preprocessing

Drop Email Content & Subject columns

```
In [45]: # As the vectorizer has extracted the main words from the content & subjec.
         # We have no need for those columns so they are dropped from the dataset

         data1 = data1.drop(columns=['content', 'Subject'])
         data1 = data1.reset_index()
         del data1['index']
         data1
```

Figure 33: Dropping off the Email Content and Subject Columns



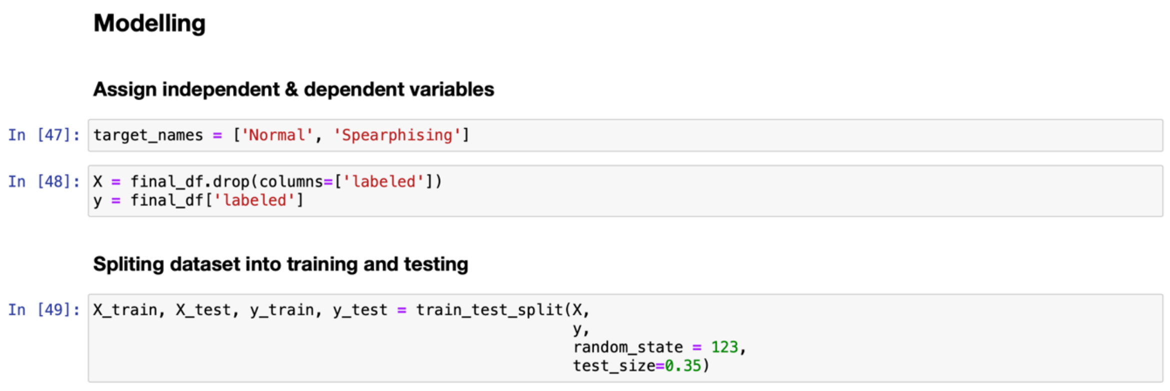
Concatenate Extracted Features

```
In [46]: # Now the vectorized features are included in the dataset
         # to replace the email content & subject which were dropped

         final_df = pd.concat([data1, unigram_content, unigram_subject], axis=1)
         final_df

         # Resulting in a dataset with 3000 rows and 1008 columns
```

Figure 34: Concatenation of Extracted Features

Modelling

Machine learning is a powerful tool for analyzing data and extracting patterns and anomalies. However, these algorithms need usable data in order to draw valid conclusions



Modelling

Assign independent & dependent variables

```
In [47]: target_names = ['Normal', 'Spearphising']
```

```
In [48]: X = final_df.drop(columns=['labeled'])
         y = final_df['labeled']
```

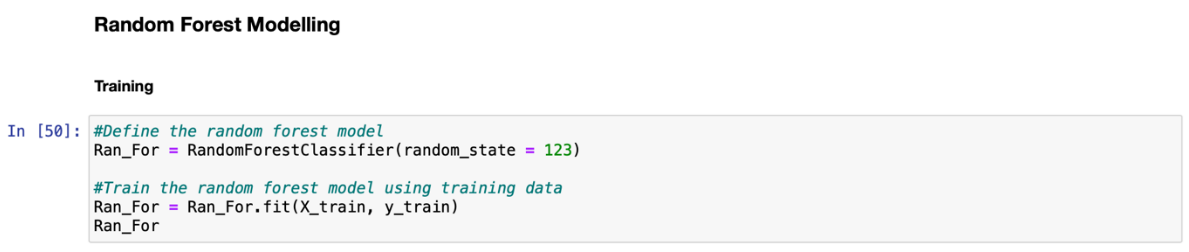
Spliting dataset into training and testing

```
In [49]: X_train, X_test, y_train, y_test = train_test_split(X,
                                                             y,
                                                             random_state = 123,
                                                             test_size=0.35)
```

Figure 35: Dependent and independent variables are assigned

Experiment 1 – Random Forest Modelling

Experiment 1 involved comparing the engagement levels 0, 1, 2 and 3. The frames depicting these states were extracted from the pre-processed folder and stored into a new train and test folder.

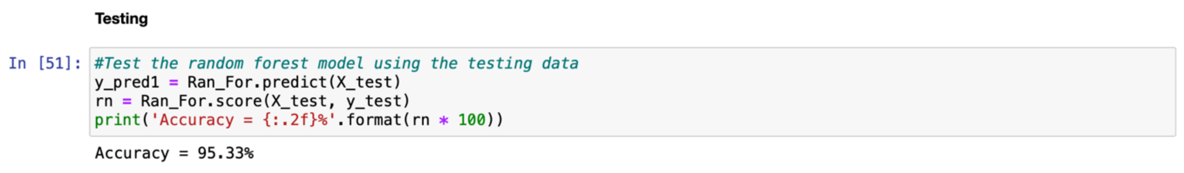


Random Forest Modelling

Training

```
In [50]: #Define the random forest model
         Ran_For = RandomForestClassifier(random_state = 123)

         #Train the random forest model using training data
         Ran_For = Ran_For.fit(X_train, y_train)
         Ran_For
```

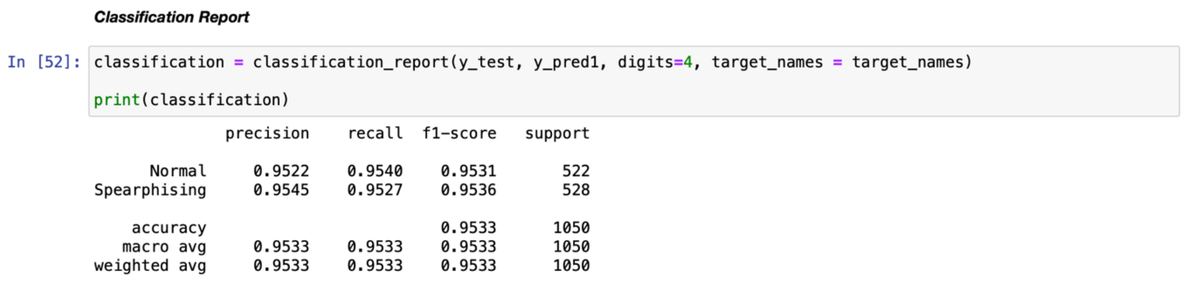
Figure 36: Training of Dataset



Testing

```
In [51]: #Test the random forest model using the testing data
         y_pred1 = Ran_For.predict(X_test)
         rn = Ran_For.score(X_test, y_test)
         print('Accuracy = {:.2f}%'.format(rn * 100))

         Accuracy = 95.33%
```

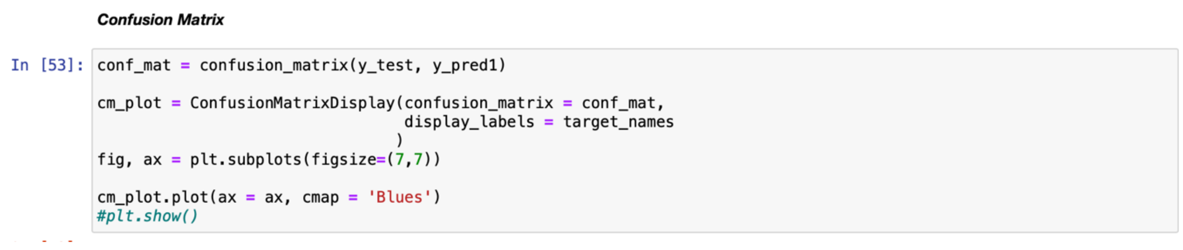
Figure 37: Testing of Dataset



Classification Report

```
In [52]: classification = classification_report(y_test, y_pred1, digits=4, target_names = target_names)

         print(classification)

                       precision    recall  f1-score   support

               Normal     0.9522    0.9540    0.9531       522
         Spearphising     0.9545    0.9527    0.9536       528

             accuracy                         0.9533      1050
            macro avg     0.9533    0.9533    0.9533      1050
         weighted avg     0.9533    0.9533    0.9533      1050
```

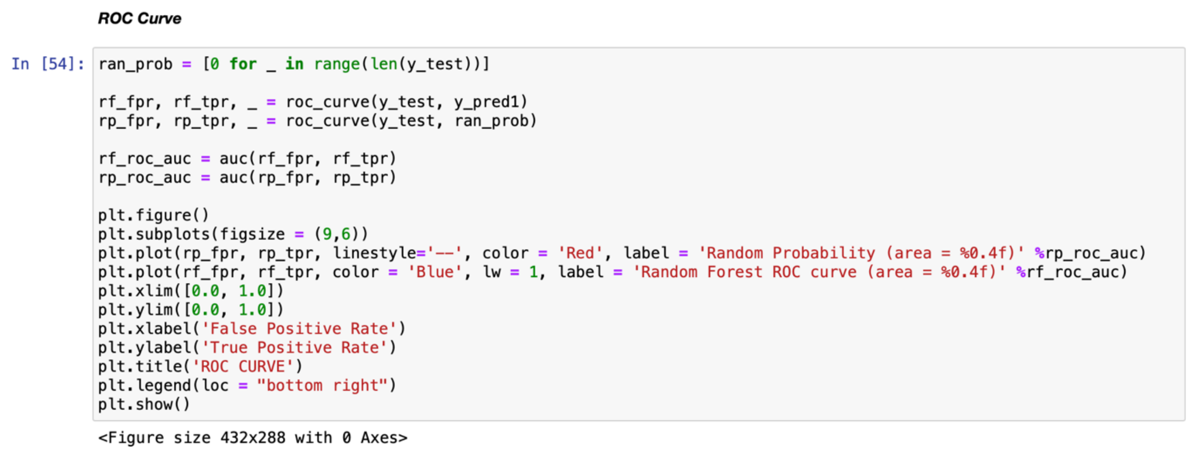
Figure 38: Classification Report of Dataset



Confusion Matrix

```
In [53]: conf_mat = confusion_matrix(y_test, y_pred1)

         cm_plot = ConfusionMatrixDisplay(confusion_matrix = conf_mat,
                                          display_labels = target_names
                                         )
         fig, ax = plt.subplots(figsize=(7,7))

         cm_plot.plot(ax = ax, cmap = 'Blues')
         #plt.show()
```

Figure 39: Random Forest Confusion matrix

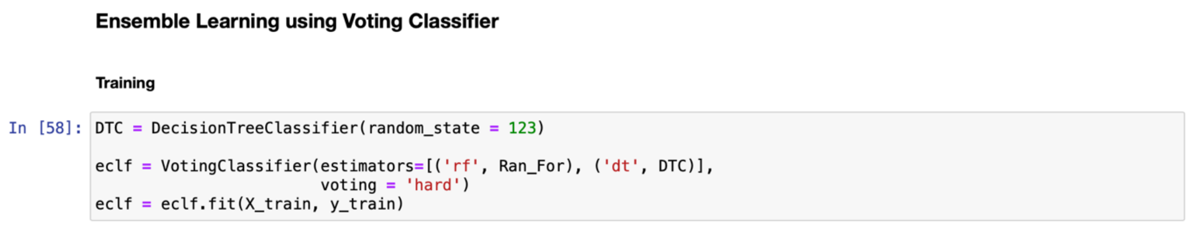


ROC Curve

```
In [54]: ran_prob = [0 for _ in range(len(y_test))]

         rf_fpr, rf_tpr, _ = roc_curve(y_test, y_pred1)
         rp_fpr, rp_tpr, _ = roc_curve(y_test, ran_prob)

         rf_roc_auc = auc(rf_fpr, rf_tpr)
         rp_roc_auc = auc(rp_fpr, rp_tpr)

         plt.figure()
         plt.subplots(figsize = (9,6))
         plt.plot(rp_fpr, rp_tpr, linestyle='--', color = 'Red', label = 'Random Probability (area = %0.4f)' %rp_roc_auc)
         plt.plot(rf_fpr, rf_tpr, color = 'Blue', lw = 1, label = 'Random Forest ROC curve (area = %0.4f)' %rf_roc_auc)
         plt.xlim([0.0, 1.0])
         plt.ylim([0.0, 1.0])
         plt.xlabel('False Positive Rate')
         plt.ylabel('True Positive Rate')
         plt.title('ROC CURVE')
         plt.legend(loc = "bottom right")
         plt.show()

         <Figure size 432x288 with 0 Axes>
```

Figure 40: Random Forest Roc Curve

Experiment II – Ensemble Learning using Voting Classifier

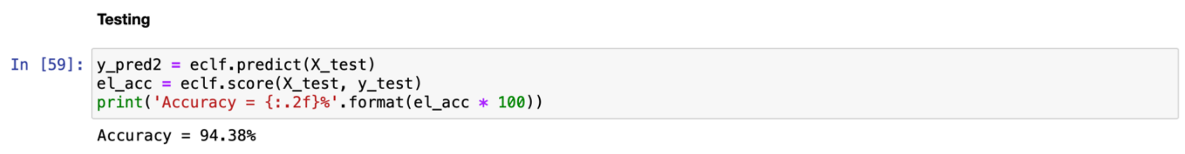


Ensemble Learning using Voting Classifier

Training

```
In [58]: DTC = DecisionTreeClassifier(random_state = 123)

         eclf = VotingClassifier(estimators=[('rf', Ran_For), ('dt', DTC)],
                                 voting = 'hard')
         eclf = eclf.fit(X_train, y_train)
```

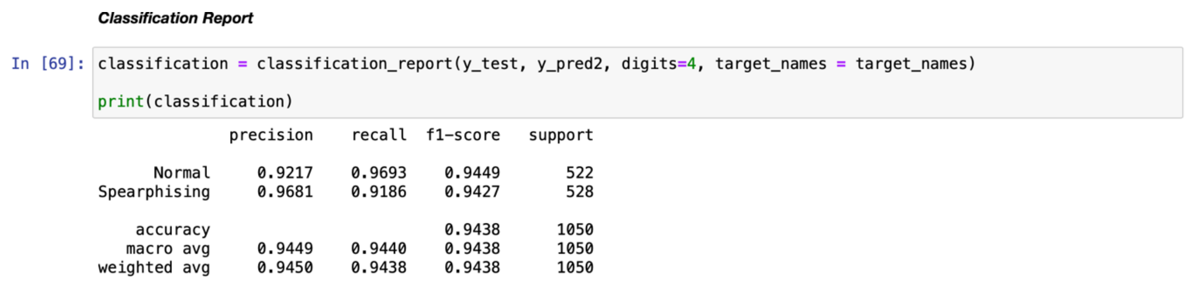
Figure 41: Training of Dataset



Testing

```
In [59]: y_pred2 = eclf.predict(X_test)
         el_acc = eclf.score(X_test, y_test)
         print('Accuracy = {:.2f}%'.format(el_acc * 100))

         Accuracy = 94.38%
```

Figure 42: Testing of Dataset

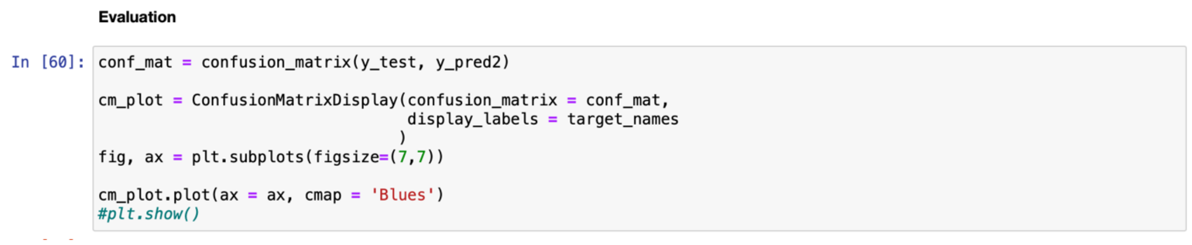


Classification Report

```
In [69]: classification = classification_report(y_test, y_pred2, digits=4, target_names = target_names)

         print(classification)
```

```
              precision    recall  f1-score   support

      Normal     0.9217    0.9693    0.9449       522
Spearphising     0.9681    0.9186    0.9427       528

    accuracy                         0.9438      1050
   macro avg     0.9449    0.9440    0.9438      1050
weighted avg     0.9450    0.9438    0.9438      1050
```

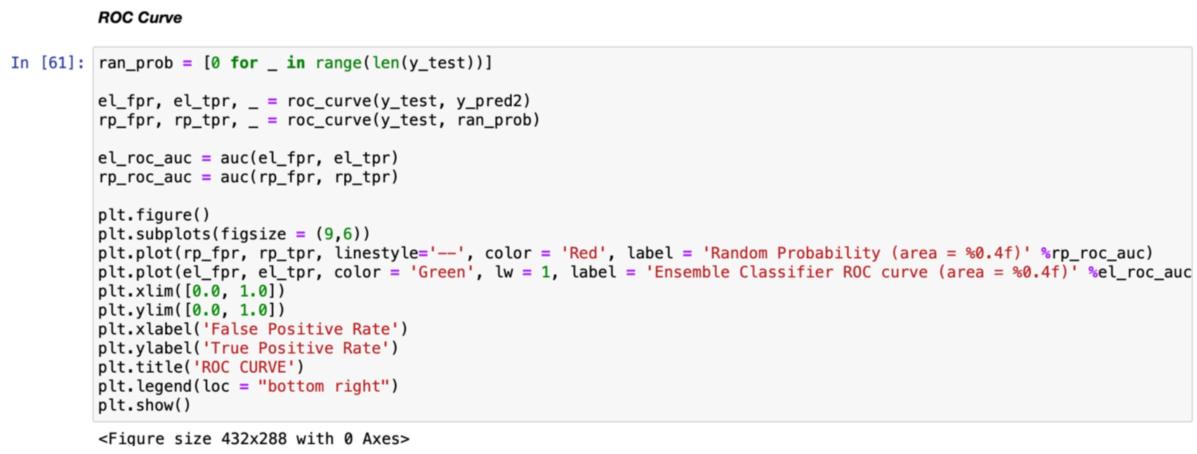
Figure 43: Classification Report



Evaluation

```
In [60]: conf_mat = confusion_matrix(y_test, y_pred2)

         cm_plot = ConfusionMatrixDisplay(confusion_matrix = conf_mat,
                                          display_labels = target_names
                                         )
         fig, ax = plt.subplots(figsize=(7,7))

         cm_plot.plot(ax = ax, cmap = 'Blues')
         #plt.show()
```

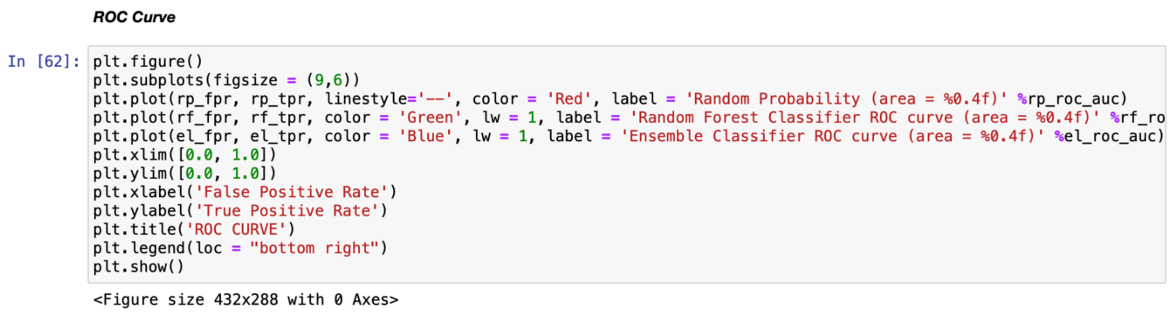
Figure 44: Ensemble Learning Confusion Matrix



ROC Curve

```
In [61]: ran_prob = [0 for _ in range(len(y_test))]

         el_fpr, el_tpr, _ = roc_curve(y_test, y_pred2)
         rp_fpr, rp_tpr, _ = roc_curve(y_test, ran_prob)

         el_roc_auc = auc(el_fpr, el_tpr)
         rp_roc_auc = auc(rp_fpr, rp_tpr)

         plt.figure()
         plt.subplots(figsize = (9,6))
         plt.plot(rp_fpr, rp_tpr, linestyle='--', color = 'Red', label = 'Random Probability (area = %0.4f)' %rp_roc_auc)
         plt.plot(el_fpr, el_tpr, color = 'Green', lw = 1, label = 'Ensemble Classifier ROC curve (area = %0.4f)' %el_roc_auc
         plt.xlim([0.0, 1.0])
         plt.ylim([0.0, 1.0])
         plt.xlabel('False Positive Rate')
         plt.ylabel('True Positive Rate')
         plt.title('ROC CURVE')
         plt.legend(loc = "bottom right")
         plt.show()

         <Figure size 432x288 with 0 Axes>
```

Figure 45: Ensemble Learning Roc Curve



ROC Curve

```
In [62]: plt.figure()
         plt.subplots(figsize = (9,6))
         plt.plot(rp_fpr, rp_tpr, linestyle='--', color = 'Red', label = 'Random Probability (area = %0.4f)' %rp_roc_auc)
         plt.plot(rf_fpr, rf_tpr, color = 'Green', lw = 1, label = 'Random Forest Classifier ROC curve (area = %0.4f)' %rf_ro
         plt.plot(el_fpr, el_tpr, color = 'Blue', lw = 1, label = 'Ensemble Classifier ROC curve (area = %0.4f)' %el_roc_auc)
         plt.xlim([0.0, 1.0])
         plt.ylim([0.0, 1.0])
         plt.xlabel('False Positive Rate')
         plt.ylabel('True Positive Rate')
         plt.title('ROC CURVE')
         plt.legend(loc = "bottom right")
         plt.show()

         <Figure size 432x288 with 0 Axes>
```

Figure 46: Roc Curve for Random Forest Classifier and Ensemble Learning