# Configuration Manual for A Deep Learning Phishing Email Classifier Combined with NLP

MSc Research Project
Masters in Cyber Security

Ebong, Maurice A.
Student ID: 20148127

School of Computing
National College of Ireland

Supervisor: Dr. Vikas Sahni

**National College of Ireland**

**MSc Project Submission Sheet**

**School of Computing**

**Student Name:** **Ebong, Maurice A.**

**Student ID:** **20148127**

**Programme:** **Master of Science in Cyber Security** **Year:** **2022**

**Module:** ……**Research Project**……..……

**Lecturer:** …**Dr…Vikas…Sahni**…..……

**Submission Due Date:**

……**16th April 2022**……..……

**Project Title:** **A Deep Learning Phishing Email Classifier Combined with NLP**

**Word Count:** **2571** **Page Count: 23**

I hereby certify that the information contained in this (my submission) is information pertaining to the 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 authors' written or electronic work is illegal (plagiarism) and may result in disciplinary action.

**Signature:** …… **Ebong, Maurice A.……**

**Date:** ……**09th April 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 the 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 for A Deep Learning Phishing Email Classifier Combined with NLP

EBONG, MAURICE A.
20148127

# 1 Introduction

This document presents the step-by-step guide to creating the code implementation of the research topic "A Deep Learning Phishing Email Classifier Combined with NLP". The research aims to classify emails as either legitimate emails or phishing emails by applying NLP processes and using machine learning and deep learning techniques in this classification. The Jose Nazario phishing email corpus and the Enron email dataset were used to carry out this research work.

The remainder of this report is organized as: Section 2 discusses the system specification in terms of hardware requirement and software requirement, Section 3 will the software installation guide and the development environment with the Python libraries. Section 4 will present code implementation and evaluation as carried out by this research work and Section 5 conclude the report.

# 2 System Specification

## 2.1 Hardware Requirement

The hardware used for carrying code implementation had a RAM of 16GB, the processor type was Intel core i7 with the processing speed of 1.99GHz, and a storage type of SSD with a 512GB capacity.

## 2.2 Software Requirement

Microsoft Windows 10 Operating System is used to carry out the code implementation with a stack of other software solutions. These software solutions include

- Anaconda Navigator – This is an open-source software used for creating and managing a working environment for different Python programming language versions packed with a set of libraries for that environment. Note that multiple work environments can be created for different Python versions on a PC with anaconda navigation.

- Jupyter Notebook – This software is an interactive Python integrated development environment (IDE) that executes Python code in blocks or cells. This IDE runs on a web browser.

- Web Browser (Microsoft Edge Browser) – This software is used to render Jupyter Notebook input IDE for executing code blocks or cells.

# 3 Software Installation Guide, Python Libraries and Environment Setup

This section will the installation of Anaconda Navigator software, list all Python libraries to be installed to successfully implement the proposed solution for this project, and create the virtual Python environment Anaconda then install all libraries listed earlier.

## 3.1 Installation Guide

To install Anaconda on your windows 10 OS, download the Anaconda installer for Windows from the Anaconda website. After a successful download of the installer file, locate the file on your PC and follow the instruction listed below:

- Double click on the installer file to run the installer application. Once the installer wizard is launched as shown in figure 1 below, click "Next" to continue with the installation.


**Figure 1: Install Anaconda wizard startup window**

- Figure 2 below is the License Agreement window, click "I Agree" to continue with the installation

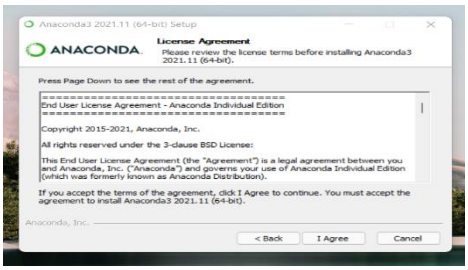



**Figure 2: License agreement window**

- Figure 3 below is the Select Installation Type window, click "Next" to continue

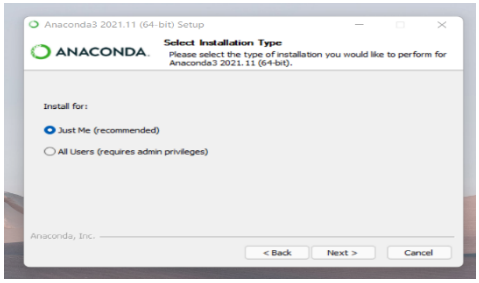



**Figure 3: Select installation type window**

- Figure 4 below is the Choose Install Location window, click "Next" to select the default installation path to proceed to the next stage in the installation

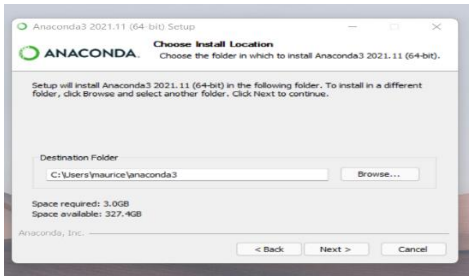



**Figure 4: Set the path for anaconda installation**

- Figure 5 below is the Advance Installation Options window, click "Install" to continue with the installation

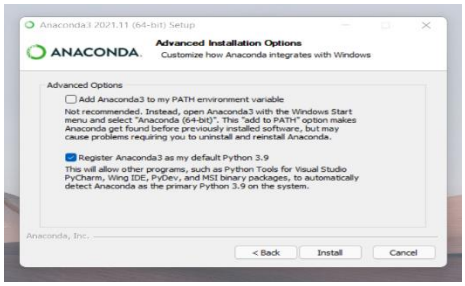



**Figure 5: Advance installation options window**

- Figure 6 below is the Installation Complete window, click "Next" to continue with the setup

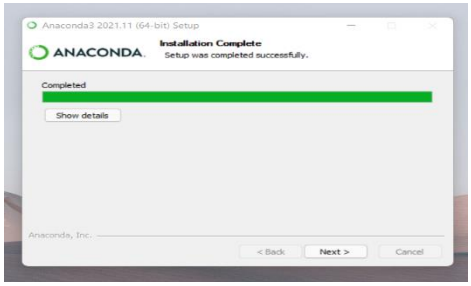


**Figure 6: Anaconda installation in progress window**

- Figure 7 below is the Install Options window for PyCharm installation, click “Next” to skip installation of PyCharm IDE


**Figure 7: PyCharm IDE install window**

- Figure 8 below is the Completing Anaconda3 Setup window, click “Finish” to complete the setup

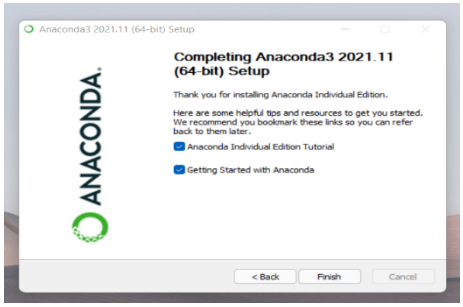



**Figure 8: Complete anaconda installation window**

## 3.2 Development Environment

After installing the Anaconda software, launch the anaconda command line. By default, an environment called ***base*** was created. To set up a new environment where Python’s Tensorflow libraries and other Python libraries used to successfully implement this project will be installed. The following instruction will guide the creation of a new Anaconda environment

- On Windows open the Start menu, scroll to the Anaconda folder, expand the folder, and select Anaconda Command Prompt from the options to launch the command line utility for Anaconda.

**Figure 9: Anaconda command prompt utility window**

- To install the current release of CPU-only TensorFlow and Python 3.6 interpreter, run the following command on the open Anaconda command prompt

  conda create -n tfenv tensorflow python=3.6
  conda activate tfenv

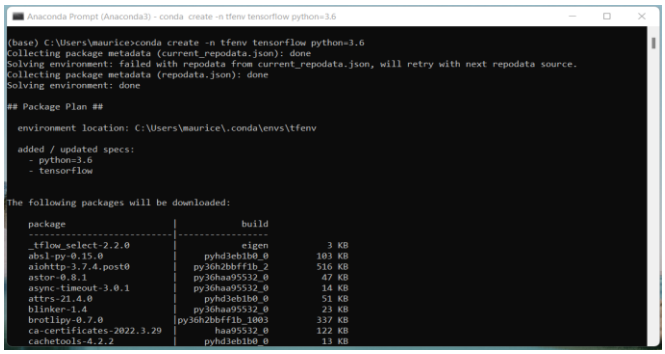



**Figure 10: Setting up anaconda environment with Python version 3.6 and Tensorflow dependencies**

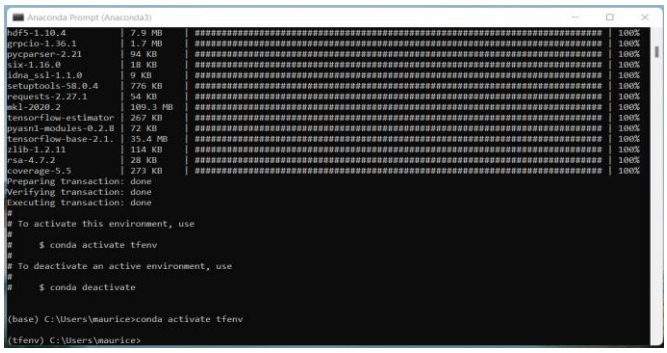



**Figure 11: Completed anaconda environment setup and environment activation to enable installation of Python dependencies**

## 3.3 Python Libraries

In the previous section, Anaconda environment "tfenv" was created and activated. Before proceeding to create a new project, the table below shows the list of Python's libraries and their installation command used to install the libraries on the Anaconda command prompt in the newly created environment

| Python Library | Anaconda Installation Command |
|---|---|
| nltk | conda install -c anaconda nltk |
| pandas | conda install -c anaconda pandas |
| seaborn | conda install -c conda-forge seaborn |
| scikit-learn | conda install -c intel scikit-learn |
| scikit-learn-intelex | conda install nltk scikit-learn-intelex |

| | |
|---|---|
| imbalanced-learn | conda install -c conda-forge imbalanced-learn |
| bs4 | conda install -c conda-forge bs4 |

**Figure 12: Python libraries and their anaconda installation command**

# 4 Implementation and Evaluation

## 4.1 Start a new project

To start the project coding implementation, click on the Start menu, scroll to the Anaconda folder, expand the folder, and select Anaconda Navigator to launch the software.

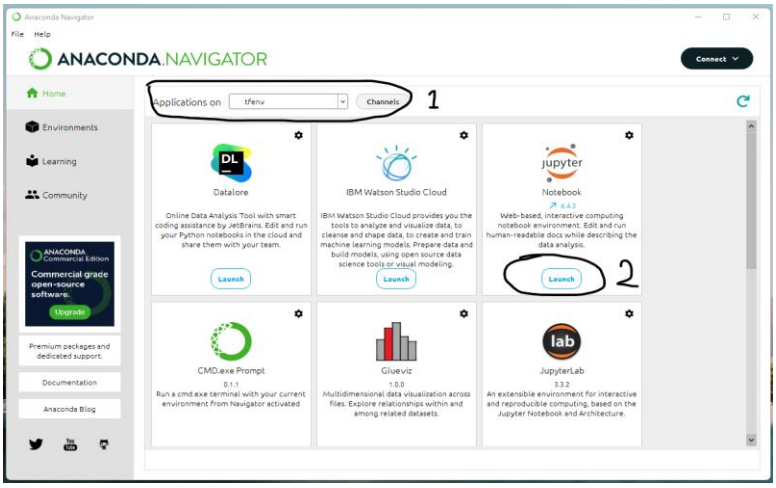



**Figure 13: Anaconda navigator startup window**

Figure 13 above is the start-up window of Anaconda Navigator, from the dropdown labelled as 1, select the "tfenv" environment on which to launch the Jupyter notebook and click the launch button labelled 2.

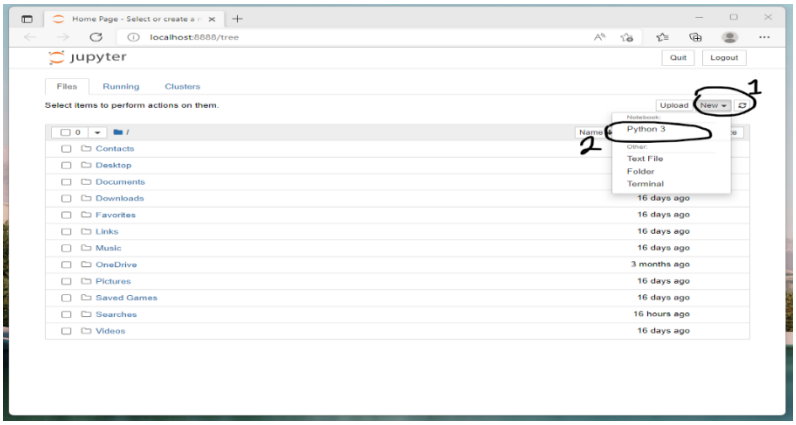



**Figure 14: Jupyter notebook start up view on the browser**

Figure 14 above is the start-up window on the browser for Jupyter notebook, click the dropdown button marked 1 to open the option menu for creating new project and click on menu option labelled "Python 3" marked 2 to start a new project.

## 4.2 Install Python Libraries using the PIP command

```
In [1]: !pip install wordninja
        !pip install pyenchant
        !pip install wordcloud

Requirement already satisfied: wordninja in /home/akrosoft/anaconda3/env
s/deep-learn/lib/python3.6/site-packages (2.0.0)
Requirement already satisfied: pyenchant in /home/akrosoft/anaconda3/env
s/deep-learn/lib/python3.6/site-packages (3.2.2)
Requirement already satisfied: wordcloud in /home/akrosoft/anaconda3/env
s/deep-learn/lib/python3.6/site-packages (1.8.1)
Requirement already satisfied: numpy>=1.6.1 in /home/akrosoft/anaconda3/e
nvs/deep-learn/lib/python3.6/site-packages (from wordcloud) (1.19.2)
Requirement already satisfied: matplotlib in /home/akrosoft/anaconda3/env
s/deep-learn/lib/python3.6/site-packages (from wordcloud) (3.1.3)
Requirement already satisfied: pillow in /home/akrosoft/anaconda3/envs/de
ep-learn/lib/python3.6/site-packages (from wordcloud) (8.4.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 i
n /home/akrosoft/anaconda3/envs/deep-learn/lib/python3.6/site-packages (f
rom matplotlib->wordcloud) (3.0.4)
Requirement already satisfied: cycler>=0.10 in /home/akrosoft/anaconda3/e
nvs/deep-learn/lib/python3.6/site-packages (from matplotlib->wordcloud)
(0.11.0)
Requirement already satisfied: python-dateutil>=2.1 in /home/akrosoft/ana
```

**Figure 15: Code block use to install Python dependencies using PIP install command**

## 4.3 Import Python's Dependencies

```
In [2]: import os
        import re
        import time
        import string
        import nltk
        import wordninja
        import enchant
        import math

        import pandas as pd
        import numpy as np

        import seaborn as sns
        import matplotlib.pyplot as plt
        %matplotlib inline

        from nltk.corpus import stopwords
        from nltk.tokenize import RegexpTokenizer
        from nltk.stem.snowball import SnowballStemmer
        from nltk.stem.porter import PorterStemmer

        from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
        from sklearn.model_selection import GridSearchCV, StratifiedKFold, train_test_split
        from sklearn.ensemble import RandomForestClassifier
        from sklearn import svm
        from sklearn.metrics import classification_report, confusion_matrix, precision_score, roc_auc_score

        from imblearn.over_sampling import SMOTE
        from imblearn.under_sampling import RandomUnderSampler

        import tensorflow as tf
        from tensorflow.keras.models import Sequential, Model
        from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
        from tensorflow.keras.layers import *
        from tensorflow.keras.wrappers.scikit_learn import KerasClassifier

        from bs4 import BeautifulSoup
        from html.parser import HTMLParser
        from wordcloud import WordCloud, ImageColorGenerator, STOPWORDS
        from PIL import Image

        nltk.download('stopwords')

        [nltk_data] Downloading package stopwords to
        [nltk_data]     /home/akrosoft/nltk_data...
        [nltk_data]   Package stopwords is already up-to-date!

Out[2]: True
```

**Figure 16: Code block use to import Python dependencies**

## 4.4 Declared Variables, custom data type and functions

```
In [3]:
dataset_directory = "dataset/extraction/"

legitimate_emails = "emails.csv"
phishing_emails_1 = "phishing3.txt"
phishing_emails_2 = "private-phishing4.txt"

label_1 = "email"
label_2 = "label"
label_3 = "file"

data = None
process_phishing_emails = None
model_scores = dict()
base_model_scores = dict()


ps = PorterStemmer()
stemmer = SnowballStemmer("english")
d = enchant.Dict("en_US")
cv = CountVectorizer()
oversampling = SMOTE()
tfidf_transformer = TfidfTransformer()

opt_adam = 'adam'
loss_type = "categorical_crossentropy"
kernel_init ='random_uniform'
activate_relu = 'relu'
activate_sig = 'sigmoid'
activate_soft = 'softmax'

svm_label ="SVM"
rfc_label ="RFC"
dnn_label ="DNN"

accuracy_label = "Accuracy"
auc_label = "AUC"
precision_label = "Precision"
```

**Figure 17: Code block use for variables declaration**

```
In [4]:
class GetPageContent(HTMLParser):
    def __init__(self):
        super().__init__()
        self.page_content = ""

    def handle_data(self, data):
        if data:
            self.page_content += data

    def get_content(self):
        return self.page_content

    def error(self, message):
        pass
```

**Figure 18: Code block showing class definition for parsing html string and retrieve the content of as string**

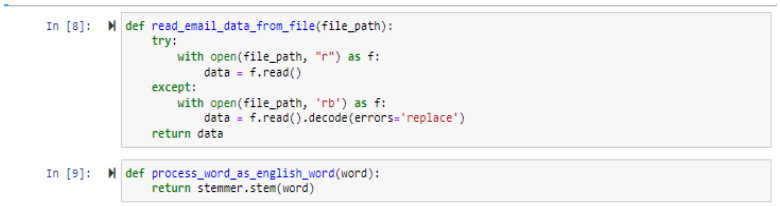
```
In [5]:
def split_concatenated_words(concatenated_word):
    return wordninja.split(concatenated_word)
```

```
In [6]:
def generate_np_array_with_default_value_and_fixed_size(word, length):
    initial_list = list()
    for i in range(length):
        initial_list.append(word)
    return np.array(initial_list, dtype=object)
```

**Figure 19: These code blocks show the functions (i) to split group of words joined together as a single word and (ii) to generate a NumPy array of a given length and default string**

```
In [7]:
def process_raw_email_data(raw_data):
    processing_emails = list()
    email_list = raw_data.split("</html>")

    for email in email_list:
        email_fragments = email.split("<html>")
        if len(email_fragments) == 2:
            html = email_fragments[1]
            soup = BeautifulSoup(html, 'html.parser')
            email_text = soup.get_text()
            email_text = email_text.replace("\n", "")
            email_text = email_text.replace("&nb=", "")
            email_text = email_text.replace("=", "")
            email_text = email_text.replace("sp;", "")

            email_header = email_fragments[0]
            soup = BeautifulSoup(email_header, 'html.parser')
            email_header = soup.get_text()
            email_header = email_header.replace("\n", "")
            email_header = email_header.replace("&nb=", "")
            email_header = email_header.replace("=", "")
            email_header = email_header.replace("sp;", "")
            corrected_email = email_header + " \n\n " + email_text

            processing_emails.append(corrected_email)

    return processing_emails
```

**Figure 20: Code block showing the function used to convert email string to processed email list removing HTML tags and other special characters**

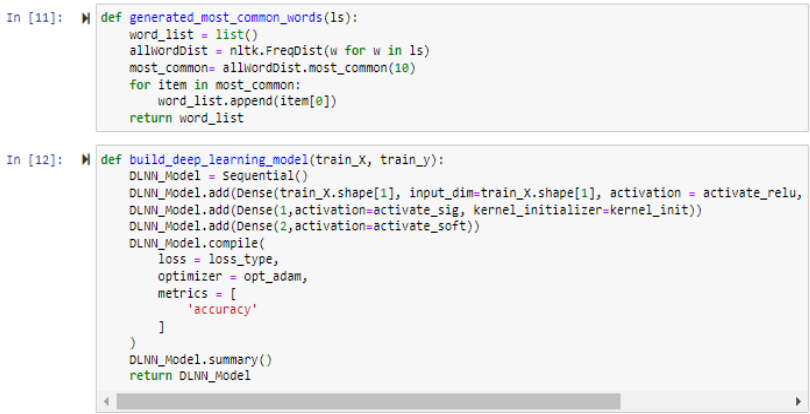


```
In [8]:
def read_email_data_from_file(file_path):
    try:
        with open(file_path, "r") as f:
            data = f.read()
    except:
        with open(file_path, 'rb') as f:
            data = f.read().decode(errors='replace')
    return data

In [9]:
def process_word_as_english_word(word):
    return stemmer.stem(word)
```

**Figure 21: These code blocks showing the functions (i) to read the content of a file and return a string of the content of the file (ii) to stem a word to its root word using SnowballStemmer modules**

```
In [10]:
def process_text(text):
    tokens = tokenizer.tokenize(text)
    message = ' '.join(tokens).lower()

    words_bag = message.split(" ")
    for i in range(0, len(words_bag)-1):
        try:
            if (len(words_bag[i]) > 15):
                if (len(words_bag[i]) <= 50):
                    word_list = split_concatenated_words(words_bag[i])
                    phrase = ' ' . join(word_list)
                    words_bag.pop(i)
                    words_bag.insert(i, phrase)
                else:
                    words_bag.pop(i)
        except:
            pass

    message = ' ' . join(words_bag)

    token = message.split()
    token = [process_word_as_english_word(word) for word in token if not (word in stopwords.words('

    return token
```

**Figure 22: Code block showing the function use to tokenize the content of an email**



```
In [11]:
def generated_most_common_words(ls):
    word_list = list()
    allWordDist = nltk.FreqDist(w for w in ls)
    most_common= allWordDist.most_common(10)
    for item in most_common:
        word_list.append(item[0])
    return word_list

In [12]:
def build_deep_learning_model(train_X, train_y):
    DLNN_Model = Sequential()
    DLNN_Model.add(Dense(train_X.shape[1], input_dim=train_X.shape[1], activation = activate_relu,
    DLNN_Model.add(Dense(1,activation=activate_sig, kernel_initializer=kernel_init))
    DLNN_Model.add(Dense(2,activation=activate_soft))
    DLNN_Model.compile(
        loss = loss_type,
        optimizer = opt_adam,
        metrics = [
            'accuracy'
        ]
    )
    DLNN_Model.summary()
    return DLNN_Model
```

**Figure 23: These code blocks showing the functions (i) to generate the list 10 most common words in a given email tokens and (ii) to build an instance of DNN model used in analyzing the phishing email corpus**

```
In [13]:
def generate_wordcloud_plot(text, mask, max_words, plot_title, image_color):
    stopwords = set(STOPWORDS)
    wc = WordCloud(
        background_color='white',
        stopwords = stopwords,
        max_words = max_words,
        max_font_size = 120,
        random_state = 42,
        mask = mask
    )
    wc.generate(text)
    plt.figure(figsize=(25.0, 15.0))
    if image_color:
        image_colors = ImageColorGenerator(mask);
        plt.imshow(wc.recolor(color_func=image_colors), interpolation="bilinear");
        plt.title(plot_title, fontdict={'size': 50,
                                        'verticalalignment': 'bottom'})
    else:
        plt.imshow(wc);
        plt.title(plot_title, fontdict={'size': 50, 'color': 'red',
                                        'verticalalignment': 'bottom'})
    plt.axis('off');
    plt.tight_layout()
```

**Figure 24: Code block showing the function use to generate wordcloud plot list of words**

```
In [14]:
def to_percentage(number):
    number = number * 100
    factor = 10 ** 2
    return math.floor(number * factor) / factor
```

**Figure 25: Code block showing the function use to convert number to its percentage equivalent**

## 4.5 Load datasets into IDE

```
In [15]:
file_path_1 = os.path.join(os.getcwd(), dataset_directory, phishing_emails_1)
file_path_2 = os.path.join(os.getcwd(), dataset_directory, phishing_emails_2)

temp_data_1 = read_email_data_from_file(file_path_1)
temp_data_2 = read_email_data_from_file(file_path_2)

data = temp_data_1 + "\n" + temp_data_2
```

```
In [16]:
process_phishing_emails = np.array(process_raw_email_data(data))
phishing_label = generate_np_array_with_default_value_and_fixed_size('phishing', process_phishing_e
```

```
In [17]:
phishing_email_corpus = np.array([process_phishing_emails, phishing_label])
phishing_email = pd.DataFrame(np.transpose(phishing_email_corpus), columns=[label_1, label_2], inde
```

**Figure 26: These code blocks showing the code use (i) to read phishing email corpus as string data, (ii) to convert email string data to email list using numpy arrays and generating column label for marking email as a phishing email, and (iii) convert the generate NumPy arrays to panda's data frame used for analyzing the algorithm to be implemented**

```
In [18]:
legitimate_email = pd.read_csv(os.path.join(os.getcwd(), dataset_directory, legitimate_emails))
legitimate_email.columns = [label_3, label_1]
legitimate_email[label_2] = legitimate_email[label_1].apply(lambda x: "legitimate" if x!="" else ""
legitimate_email.drop([label_3], axis=1, inplace=True)
legitimate_email = legitimate_email.sample(n=5000)
```

**Figure 27: Code block showing the code used to read legitimate emails from a CSV file**

## 4.6 Data pre-processing and data visualization

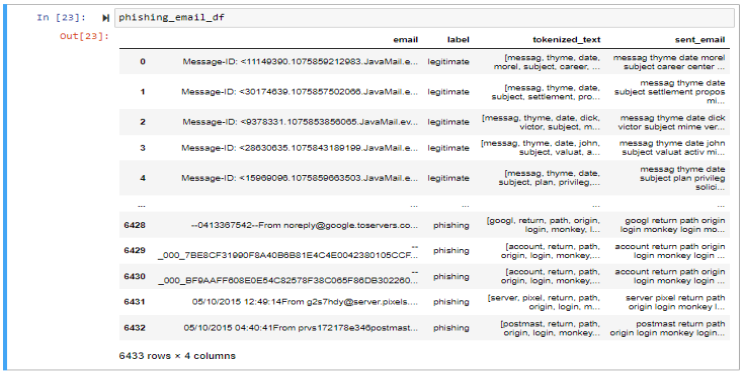
```
In [19]:
phishing_email_df = pd.concat([legitimate_email, phishing_email], ignore_index=True)
```

```
In [20]:
tokenizer = RegexpTokenizer(r'[A-Za-z]+')
```

```
In [21]:
phishing_email_df['tokenized_text'] = phishing_email_df['email'].apply(process_text)
```

```
In [22]:
phishing_email_df['sent_email'] = phishing_email_df['tokenized_text'].map(lambda ls: ' '.join(ls))
```

**Figure 28: These code blocks show the code use (i) to merge phishing email data frame and legitimate email data frame, (iii) to create an instance of the RegexpTokenizer module used in the tokenizing email string, (iii) to generate a token from email text and (iv) to generate sent email from the email token**



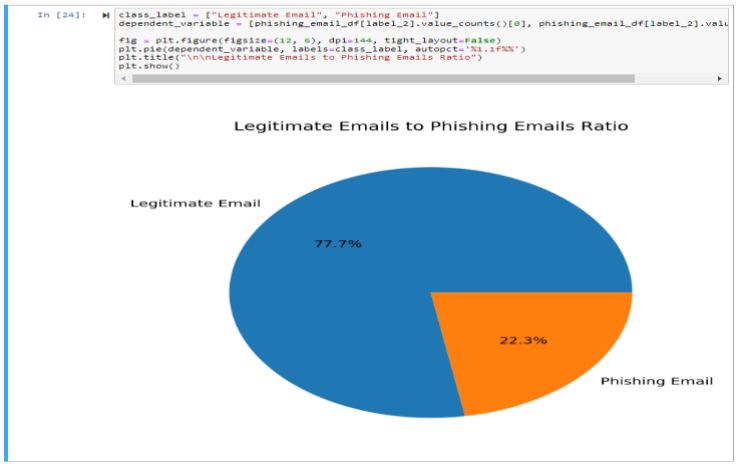
```
In [23]: phishing_email_df
```

Out[23]:

| | email | label | tokenized_text | sent_email |
|---|---|---|---|---|
| 0 | Message-ID: <11149390.1075859212983.JavaMail.e... | legitimate | [messag, thyme, date, morel, subject, career, ... | messag thyme date morel subject career center ... |
| 1 | Message-ID: <30174639.1075857502066.JavaMail.e... | legitimate | [messag, thyme, date, subject, settlement, pro... | messag thyme date subject settlement propos mi... |
| 2 | Message-ID: <9378331.1075853856065.JavaMail.ev... | legitimate | [messag, thyme, date, dick, victor, subject, m... | messag thyme date dick victor subject mime ver... |
| 3 | Message-ID: <28630635.1075843189199.JavaMail.e... | legitimate | [messag, thyme, date, john, subject, valuat, a... | messag thyme date john subject valuat activ mi... |
| 4 | Message-ID: <15969096.1075859663503.JavaMail.e... | legitimate | [messag, thyme, date, subject, plan, privileg,... | messag thyme date subject plan privileg solici... |
| ... | ... | ... | ... | ... |
| 6428 | --0413367542--From noreply@google.toservers.co... | phishing | [googl, return, path, origin, login, monkey, l... | googl return path origin login monkey login mo... |
| 6429 | _000_7BE8CF31990F8A40B6B81E4C4E00423B0105CCF... | phishing | [account, return, path, origin, login, monkey,... | account return path origin login monkey login ... |
| 6430 | _000_BF9AAFF608E0E54C82578F38C065F86DB302260... | phishing | [account, return, path, origin, login, monkey,... | account return path origin login monkey login ... |
| 6431 | 05/10/2015 12:49:14From g2s7hdy@server.pixels.... | phishing | [server, pixel, return, path, origin, login, m... | server pixel return path origin login monkey l... |
| 6432 | 05/10/2015 04:40:41From prvs172178e346postmast... | phishing | [postmast, return, path, origin, login, monkey... | postmast return path origin login monkey login... |

6433 rows × 4 columns

**Figure 29: Code block showing the summary phishing email dataset used to analyze the algorithms to implemented**



```
In [24]: class_label = ["Legitimate Email", "Phishing Email"]
dependent_variable = [phishing_email_df[label_2].value_counts()[0], phishing_email_df[label_2].valu

fig = plt.figure(figsize=(12, 6), dpi=144, tight_layout=False)
plt.pie(dependent_variable, labels=class_label, autopct='%1.1f%%')
plt.title("\n\nLegitimate Emails to Phishing Emails Ratio")
plt.show()
```



**Figure 30: Code block showing the pie plot binary class distribution of the phishing email corpus**

```
In [25]: legitimate_emails_group = phishing_email_df[phishing_email_df[label_2] == 'legitimate']
phishing_emails_group = phishing_email_df[phishing_email_df[label_2] == 'phishing']
```

**Figure 31: Code block showing the division of the dataset into phishing email and legitimate emails class**

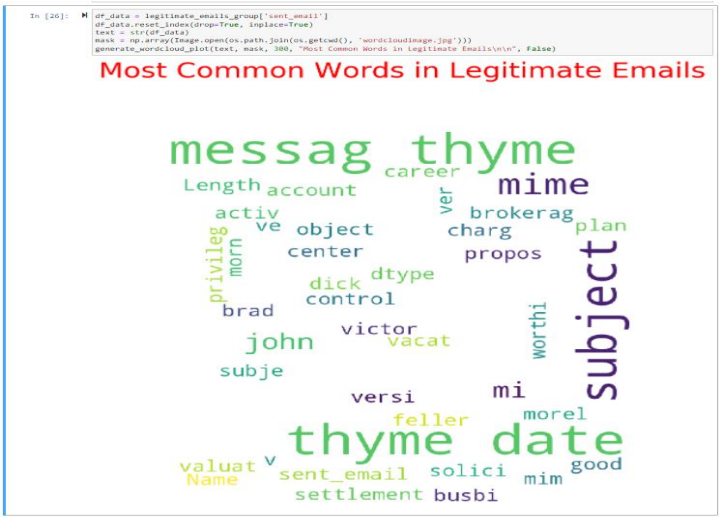



**Figure 32: Code block showing the wordcloud plot for legitimate email class**

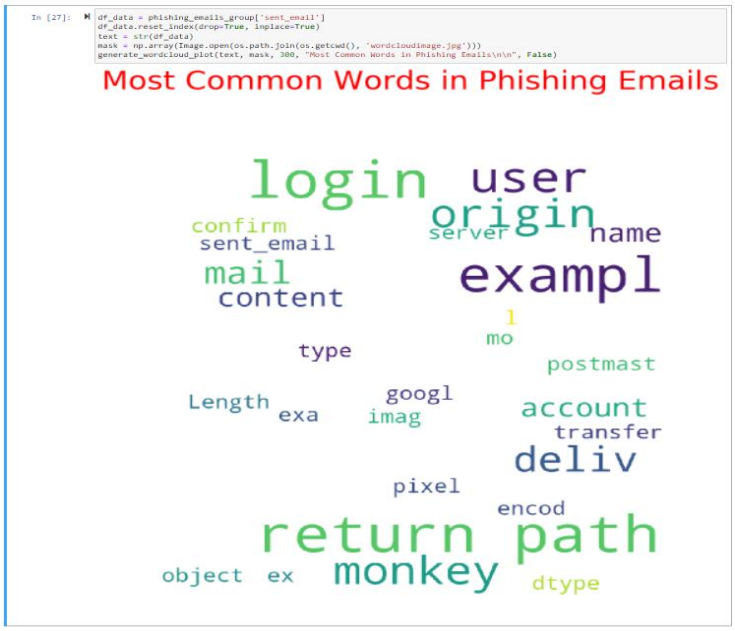



**Figure 33: Code block showing the wordcloud plot for phishing email class**

## 4.7 Models' implementation and Analysis of result

```
In [28]: X = cv.fit_transform(phishing_email_df['sent_email'])

         y = pd.get_dummies(phishing_email_df[label_2])
         y = y.iloc[:, 1].values

In [29]: X, y = oversampling.fit_resample(X, y)

In [30]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)
         print("Train X: ", X_train.shape)
         print("Train y: ", y_train.shape)

         Train X:  (7000, 11899)
         Train y:  (7000,)
```

**Figure 34: These code blocks show the code use (i) to generate independent variables labelled X and dependent variables labelled y, (ii) to eliminate the imbalance in the dataset, and (iii) split the dataset into training set and testing set**

```
In [31]:
SVM_model = svm.SVC()

training_start_time = time.time()

SVM_model.fit(X_train, y_train)

training_end_time = time.time()
training_svm_time_diff = training_end_time - training_start_time
```

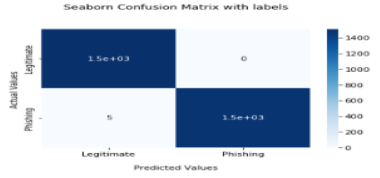
```
In [32]:
testing_start_time = time.time()

SVM_model.score(X_test, y_test)

testing_end_time = time.time()
testing_svm_time_diff = testing_end_time - testing_start_time
```

**Figure 35: These code blocks showing the code use (i) to create SVM model and train the SVM model and (ii) to evaluate the performance trained SVM model**

```
In [33]:
print('\n\nBASE SVM MODEL SUMMARY REPORT')
print('================================================================================')
print('================================================================================\n\n')

print('Training Time :', training_svm_time_diff, " seconds")
print('Testing Time :', testing_svm_time_diff, " seconds")

print('\nTraining Accuracy :', SVM_model.score(X_train, y_train))
print('Testing Accuracy :', SVM_model.score(X_test, y_test))
y_pred = SVM_model.predict(X_test)
confusion_mat = confusion_matrix(y_test, y_pred)

print('\nCLASSIFICATION REPORT\n')
print(classification_report(y_pred, y_test, target_names =['Phishing','Legitimate']))

print('\nCONFUSION MATRIX')
plt.figure(figsize= (6,4))
ax = sns.heatmap(confusion_mat, annot = True, cmap="Blues")
ax.set_title('Seaborn Confusion Matrix with labels\n\n');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');
ax.xaxis.set_ticklabels(['Legitimate','Phishing'])
ax.yaxis.set_ticklabels(['Legitimate','Phishing'])
plt.show()

print("\n\nMODEL EVALUATION SCORES\n")
base_model_scores[svm_label] = dict()
base_model_scores[svm_label][accuracy_label] = to_percentage(accuracy_score(y_test, y_pred))
base_model_scores[svm_label][auc_label] = to_percentage(roc_auc_score(y_test, y_pred))
base_model_scores[svm_label][precision_label] = to_percentage(precision_score(y_test, y_pred))

print("Accuracy Score: " + str(base_model_scores[svm_label][accuracy_label]))
print("AUC Score: " + str(base_model_scores[svm_label][auc_label]))
print("Precision Score: " + str(base_model_scores[svm_label][precision_label]))
print()
print()
```

**Figure 36: Code block showing the code use to display SVM model analysis and results**

```
BASE SVM MODEL SUMMARY REPORT
================================================================================
================================================================================

Training Time : 3.367511749267578  seconds
Testing Time : 1.1377758979797363  seconds

Training Accuracy : 0.9998571428571429
Testing Accuracy : 0.9983333333333333

CLASSIFICATION REPORT

              precision    recall  f1-score   support

    Phishing       1.00      1.00      1.00      1516
  Legitimate       1.00      1.00      1.00      1484

    accuracy                           1.00      3000
   macro avg       1.00      1.00      1.00      3000
weighted avg       1.00      1.00      1.00      3000

CONFUSION MATRIX
```




```
MODEL EVALUATION SCORES

Accuracy Score: 99.83
AUC Score: 99.83
Precision Score: 100.0
```

**Figure 37: Code block showing details of the execution of code block in figure 36**

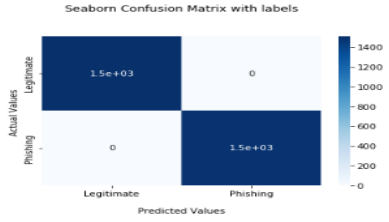
```
In [34]: tuned_parameters = {'kernel': ['linear', 'rbf'], 'gamma': [1e-3, 1e-4], 'C': [1, 10, 100, 500]}

In [35]: SVM_tuned_model = GridSearchCV(svm.SVC(), tuned_parameters)

         training_start_time = time.time()

         SVM_tuned_model.fit(X_train, y_train)

         training_end_time = time.time()
         training_tuned_svm_time_diff = training_end_time - training_start_time

In [36]: testing_start_time = time.time()

         SVM_tuned_model.score(X_test, y_test)

         testing_end_time = time.time()
         testing_tuned_svm_time_diff = testing_end_time - testing_start_time
```

**Figure 38: These code blocks showing the code use (i) to create tune parameter SVM model, (ii) to perform hyperparameter tuning for SVM model using GridSearchCV module and training SVM model, and (iii) to evaluate the performance trained tuned SVM model**

```
In [37]: print('\n\nHYPERPARAMETER TUNED SVM MODEL SUMMARY REPORT')
         print('=========================================================================')
         print('=========================================================================\n\n')

         print('Training Time :', training_tuned_svm_time_diff, " seconds")
         print('Testing Time :', testing_tuned_svm_time_diff, " seconds")

         print('\nTraining Accuracy :', SVM_tuned_model.score(X_train, y_train))
         print('Testing Accuracy :', SVM_tuned_model.score(X_test, y_test))
         y_pred = SVM_tuned_model.predict(X_test)
         confusion_mat = confusion_matrix(y_test, y_pred)

         print('\nCLASSIFICATION REPORT\n')
         print(classification_report(y_pred, y_test, target_names =['Phishing','Legitimate']))

         print('\nCONFUSION MATRIX')
         plt.figure(figsize= (6,4))
         ax = sns.heatmap(confusion_mat, annot = True, cmap="Blues")
         ax.set_title('Seaborn Confusion Matrix with labels\n\n');
         ax.set_xlabel('\nPredicted Values')
         ax.set_ylabel('Actual Values ');
         ax.xaxis.set_ticklabels(['Legitimate','Phishing'])
         ax.yaxis.set_ticklabels(['Legitimate','Phishing'])
         plt.show()

         print("\n\nMODEL EVALUATION SCORES\n")
         model_scores[svm_label] = dict()
         model_scores[svm_label][accuracy_label] = to_percentage(accuracy_score(y_test, y_pred))
         model_scores[svm_label][auc_label] = to_percentage(roc_auc_score(y_test, y_pred))
         model_scores[svm_label][precision_label] = to_percentage(precision_score(y_test, y_pred))

         print("Accuracy Score: " + str(model_scores[svm_label][accuracy_label]))
         print("AUC Score: " + str(model_scores[svm_label][auc_label]))
         print("Precision Score: " + str(model_scores[svm_label][precision_label]))
         print()
         print()
```

**Figure 39: Code block showing the code use to display tuned SVM model analysis and results**

```
HYPERPARAMETER TUNED SVM MODEL SUMMARY REPORT
=========================================================================
=========================================================================


Training Time : 133.42724633216858  seconds
Testing Time : 0.2123410701751709  seconds

Training Accuracy : 1.0
Testing Accuracy : 1.0

CLASSIFICATION REPORT

              precision    recall  f1-score   support

    Phishing       1.00      1.00      1.00      1511
  Legitimate       1.00      1.00      1.00      1489

    accuracy                           1.00      3000
   macro avg       1.00      1.00      1.00      3000
weighted avg       1.00      1.00      1.00      3000

CONFUSION MATRIX
```




```
MODEL EVALUATION SCORES

Accuracy Score: 100.0
AUC Score: 100.0
Precision Score: 100.0
```

**Figure 40: Code block showing details of the execution of code block in figure 39**

```
In [38]:
svm_performance_df = pd.DataFrame({
    "SVM": [base_model_scores[svm_label][accuracy_label], base_model_scores[svm_label][auc_label],
    "TUNED SVM": [model_scores[svm_label][accuracy_label], model_scores[svm_label][auc_label], mode
}, index=[accuracy_label, auc_label, precision_label])

In [39]:
svm_performance_df

Out[39]:
```

| | SVM | TUNED SVM |
|---|---|---|
| Accuracy | 99.83 | 100.0 |
| AUC | 99.83 | 100.0 |
| Precision | 100.00 | 100.0 |

**Figure 41: These code blocks showing the code use (i) to convert output of analyzed SVM and tuned SVM into panda's data frame, (ii) to view the summary of analyzed SVM models in tabular form**

```
In [40]:
svm_performance_df.plot(kind="bar")
plt.title("Phishing Email Classification Performance Multiple Bar Chart for SVM Models")
plt.xlabel("Evaluation Metrics")
plt.ylabel("Evaluation in Percentage(%)")

Out[40]: Text(0, 0.5, 'Evaluation in Percentage(%)')
```

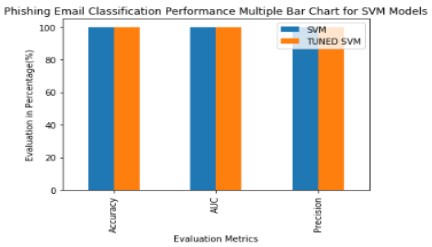



**Figure 42: Code block showing the code use to plot a multiple bar chart comparing implemented SVM models and the output generated by executing the code**

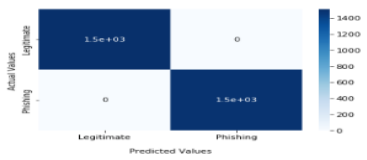
```
In [41]:
rfc_model = RandomForestClassifier(random_state=42)

In [42]:
training_start_time = time.time()

rfc_model.fit(X_train, y_train)

training_end_time = time.time()
training_rfc_time_diff = training_end_time - training_start_time

In [43]:
testing_start_time = time.time()

pred = rfc_model.predict(X_test)

testing_end_time = time.time()
testing_rfc_time_diff = testing_end_time - testing_start_time

In [44]:
accuracy_score(y_test,pred)

Out[44]: 1.0
```

**Figure 43: These code blocks showing the code use (i) to create RFC model instance, (ii) train the RFC model, (iii) generate prediction for performance trained RFC model and, (iv) to evaluate the performance trained RFC model**

```
In [45]:
print('\n\nBASE RFC MODEL SUMMARY REPORT')
print('=======================================================================')
print('=======================================================================\n\n')

print('Training Time :', training_rfc_time_diff, " seconds")
print('Testing Time :', testing_rfc_time_diff, " seconds")

print('\nTraining Accuracy :', accuracy_score(rfc_model.predict(X_train), y_train))
print('Testing Accuracy :', accuracy_score(rfc_model.predict(X_test), y_test))
y_pred = rfc_model.predict(X_test)
confusion_mat = confusion_matrix(y_test, y_pred)

print('\nCLASSIFICATION REPORT\n')
print(classification_report(y_pred, y_test, target_names =['Phishing','Legitimate']))

print('\nCONFUSION MATRIX')
plt.figure(figsize= (6,4))
ax = sns.heatmap(confusion_mat, annot = True, cmap="Blues")
ax.set_title('Seaborn Confusion Matrix with labels\n\n');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');
ax.xaxis.set_ticklabels(['Legitimate','Phishing'])
ax.yaxis.set_ticklabels(['Legitimate','Phishing'])
plt.show()

print("\n\nMODEL EVALUATION SCORES\n")
base_model_scores[rfc_label] = dict()
base_model_scores[rfc_label][accuracy_label] = to_percentage(accuracy_score(y_test, y_pred))
base_model_scores[rfc_label][auc_label] = to_percentage(roc_auc_score(y_test, y_pred))
base_model_scores[rfc_label][precision_label] = to_percentage(precision_score(y_test, y_pred))

print("Accuracy Score: " + str(base_model_scores[rfc_label][accuracy_label]))
print("AUC Score: " + str(base_model_scores[rfc_label][auc_label]))
print("Precision Score: " + str(base_model_scores[rfc_label][precision_label]))
print()
print()
```

**Figure 44: Code block showing the code use to display RFC model analysis and results**

```
BASE RFC MODEL SUMMARY REPORT
=======================================================================
=======================================================================

Training Time : 0.8450956344604492  seconds
Testing Time : 0.08963203430175781  seconds

Training Accuracy : 1.0
Testing Accuracy : 1.0

CLASSIFICATION REPORT

              precision    recall  f1-score   support

    Phishing       1.00      1.00      1.00      1511
  Legitimate       1.00      1.00      1.00      1489

    accuracy                           1.00      3000
   macro avg       1.00      1.00      1.00      3000
weighted avg       1.00      1.00      1.00      3000

CONFUSION MATRIX
```




```
MODEL EVALUATION SCORES

Accuracy Score: 100.0
AUC Score: 100.0
Precision Score: 100.0
```

**Figure 45: Code block showing details of the execution of code block in figure 44**

```
In [46]:
param_grid = {
    'n_estimators': [100, 200, 1000],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [80, 90, 100],
    'criterion' :['gini', 'entropy'],
    'bootstrap': [True],
}

In [47]:
rfc=RandomForestClassifier(random_state=42)
tuned_model_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)

In [48]:
training_start_time = time.time()

tuned_model_rfc.fit(X_train, y_train)

training_end_time = time.time()
training_tuned_rfc_time_diff = training_end_time - training_start_time

In [49]:
testing_start_time = time.time()

pred=tuned_model_rfc.predict(X_test)

testing_end_time = time.time()
testing_tuned_rfc_time_diff = testing_end_time - testing_start_time

In [50]:
accuracy_score(y_test,pred)

Out[50]: 1.0
```

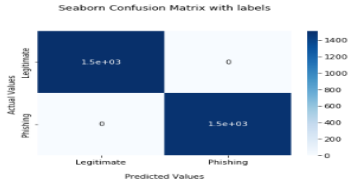
**Figure 46: These code blocks showing the code use (i) to create tune parameter RFC model, (ii) to perform hyperparameter tuning for RFC model using GridSearchCV module, (iii) to**

**training tuned RFC model, (iv) generate a prediction for performance trained tuned RFC model and, (v) to evaluate the performance trained tuned RFC model**

```
In [51]:
print('\n\nHYPERPARAMETER TUNED RFC MODEL SUMMARY REPORT')
print('=============================================================================')
print('=============================================================================\n\n')

print('Training Time :', training_tuned_rfc_time_diff, " seconds")
print('Testing Time :', testing_tuned_rfc_time_diff, " seconds")

print('\nTraining Accuracy :', accuracy_score(tuned_model_rfc.predict(X_train), y_train))
print('Testing Accuracy :', accuracy_score(tuned_model_rfc.predict(X_test), y_test))
y_pred = tuned_model_rfc.predict(X_test)
confusion_mat = confusion_matrix(y_test, y_pred)

print('\nCLASSIFICATION REPORT\n')
print(classification_report(y_pred, y_test, target_names =['Phishing','Legitimate']))

print('\nCONFUSION MATRIX')
plt.figure(figsize= (6,4))
ax = sns.heatmap(confusion_mat, annot = True, cmap="Blues")
ax.set_title('Seaborn Confusion Matrix with labels\n\n');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');
ax.xaxis.set_ticklabels(['Legitimate','Phishing'])
ax.yaxis.set_ticklabels(['Legitimate','Phishing'])
plt.show()

print("\n\nMODEL EVALUATION SCORES\n")
model_scores[rfc_label] = dict()
model_scores[rfc_label][accuracy_label] = to_percentage(accuracy_score(y_test, y_pred))
model_scores[rfc_label][auc_label] = to_percentage(roc_auc_score(y_test, y_pred))
model_scores[rfc_label][precision_label] = to_percentage(precision_score(y_test, y_pred))

print("Accuracy Score: " + str(model_scores[rfc_label][accuracy_label]))
print("AUC Score: " + str(model_scores[rfc_label][auc_label]))
print("Precision Score: " + str(model_scores[rfc_label][precision_label]))
print()
print()
```

**Figure 47: Code block showing the code used to display tuned RFC model analysis and results**

```
HYPERPARAMETER TUNED RFC MODEL SUMMARY REPORT
=============================================================================
=============================================================================

Training Time : 943.6954696178436  seconds
Testing Time : 0.09256100654602051  seconds

Training Accuracy : 1.0
Testing Accuracy : 1.0

CLASSIFICATION REPORT

              precision    recall  f1-score   support

    Phishing       1.00      1.00      1.00      1511
  Legitimate       1.00      1.00      1.00      1489

    accuracy                           1.00      3000
   macro avg       1.00      1.00      1.00      3000
weighted avg       1.00      1.00      1.00      3000

CONFUSION MATRIX
```




```
MODEL EVALUATION SCORES

Accuracy Score: 100.0
AUC Score: 100.0
Precision Score: 100.0
```

**Figure 48: Code block showing details of the execution of code block in figure 47**

```
In [52]:
rfc_performance_df = pd.DataFrame({
    "RFC": [base_model_scores[rfc_label][accuracy_label], base_model_scores[rfc_label][auc_label],
    "TUNED RFC": [model_scores[rfc_label][accuracy_label], model_scores[rfc_label][auc_label], mode
}, index=[accuracy_label, auc_label, precision_label])

In [53]:
rfc_performance_df
```

Out[53]:

| | RFC | TUNED RFC |
|---|---|---|
| Accuracy | 100.0 | 100.0 |
| AUC | 100.0 | 100.0 |
| Precision | 100.0 | 100.0 |

**Figure 49: These code blocks showing the code use (i) to convert output of analyzed RFC and tuned RFC into panda's data frame, (ii) to view the summary of analyzed RFC models in tabular form**

```
In [54]: rfc_performance_df.plot(kind="bar")
         plt.title("Phishing Email Classification Performance Multiple Bar Chart for RFC Models")
         plt.xlabel("Evaluation Metrics")
         plt.ylabel("Evaluation in Percentage(%)")

Out[54]: Text(0, 0.5, 'Evaluation in Percentage(%)')
```

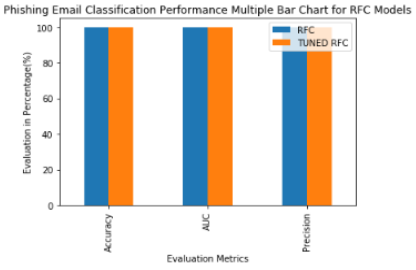



**Figure 50: Code block showing the code use to plot a multiple bar chart comparing implemented RFC models and the output generated by executing the code**

```
In [55]: NN_train_y = pd.get_dummies(phishing_email_df['label'])
         NN_train_y = NN_train_y.iloc[:, 1].values

         NN_train_X = cv.fit_transform(phishing_email_df['sent_email'])
         NN_train_X = tfidf_transformer.fit_transform(NN_train_X)

         NN_train_X, NN_train_y = oversampling.fit_resample(NN_train_X, NN_train_y)

         NN_train_X = NN_train_X.A

In [56]: X_train, X_test, y_train, y_test = train_test_split(NN_train_X, NN_train_y, test_size = 0.3, random
         print("Train X: ", X_train.shape)
         print("Train y: ", y_train.shape)

Train X:  (7000, 11899)
Train y:  (7000,)
```

**Figure 51: These code blocks showing the code use (i) to generate independent variables labelled NN_train_X and dependent variable labelled NN_train_y for DNN analysis and eliminate any imbalance in the dataset, (ii) split the dataset into training set and testing set**

```
In [57]: deep_learning_model = KerasClassifier(lambda:build_deep_learning_model(X_train, y_train), epochs=1
         training_start_time = time.time()
         deep_learning_model.fit(X_train, y_train)
         training_end_time = time.time()
         training_dnn_time_diff = training_end_time - training_start_time

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 11899)             141598100
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 11900
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 4
=================================================================
Total params: 141,610,004
Trainable params: 141,610,004
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
70/70 [==============================] - 31s 450ms/step - loss: 0.5585 - accuracy: 0.9301
Epoch 2/10
70/70 [==============================] - 31s 448ms/step - loss: 0.5156 - accuracy: 0.9989
Epoch 3/10
70/70 [==============================] - 32s 456ms/step - loss: 0.4859 - accuracy: 1.0000
Epoch 4/10
70/70 [==============================] - 32s 451ms/step - loss: 0.4589 - accuracy: 1.0000
Epoch 5/10
70/70 [==============================] - 32s 453ms/step - loss: 0.4340 - accuracy: 1.0000
Epoch 6/10
70/70 [==============================] - 32s 452ms/step - loss: 0.4109 - accuracy: 1.0000
Epoch 7/10
70/70 [==============================] - 32s 450ms/step - loss: 0.3893 - accuracy: 1.0000
Epoch 8/10
70/70 [==============================] - 32s 452ms/step - loss: 0.3692 - accuracy: 1.0000
Epoch 9/10
70/70 [==============================] - 32s 452ms/step - loss: 0.3504 - accuracy: 1.0000
Epoch 10/10
70/70 [==============================] - 32s 451ms/step - loss: 0.3328 - accuracy: 1.0000
```

**Figure 52: Code block showing the code to create an instance of DNN model and train the DNN model using the training dataset**

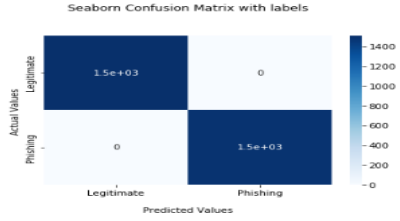
```
In [58]:
y_pred_train = deep_learning_model.predict(X_train)

testing_start_time = time.time()

y_pred = deep_learning_model.predict(X_test)

testing_end_time = time.time()
testing_dnn_time_diff = testing_end_time - testing_start_time
```

**Figure 53: Code block showing the code use to generate prediction for performance trained tuned DNN model**

```
In [59]:
print('\n\nDNN MODEL SUMMARY REPORT')
print('========================================================================')
print('========================================================================\n\n')

print('Training Time :', training_dnn_time_diff, " seconds")
print('Testing Time :', testing_dnn_time_diff, " seconds")

print('\nTraining Accuracy :', accuracy_score(y_train, y_pred_train))
print('Testing Accuracy :', accuracy_score(y_test, y_pred))

confusion_mat = confusion_matrix(y_test, y_pred)

print('\nCLASSIFICATION REPORT\n')
print(classification_report(y_pred, y_test, target_names =['Phishing','Legitimate']))

print('\nCONFUSION MATRIX')
plt.figure(figsize= (6,4))
ax = sns.heatmap(confusion_mat, annot = True, cmap="Blues")
ax.set_title('Seaborn Confusion Matrix with labels\n\n');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');
ax.xaxis.set_ticklabels(['Legitimate','Phishing'])
ax.yaxis.set_ticklabels(['Legitimate','Phishing'])
plt.show()

print("\n\nMODEL EVALUATION SCORES\n")
model_scores[dnn_label] = dict()
model_scores[dnn_label][accuracy_label] = to_percentage(accuracy_score(y_test, y_pred))
model_scores[dnn_label][auc_label] = to_percentage(roc_auc_score(y_test, y_pred))
model_scores[dnn_label][precision_label] = to_percentage(precision_score(y_test, y_pred))

print("Accuracy Score: " + str(model_scores[dnn_label][accuracy_label]))
print("AUC Score: " + str(model_scores[dnn_label][auc_label]))
print("Precision Score: " + str(model_scores[dnn_label][precision_label]))
print()
print()
```

**Figure 54: Code block showing the code use to display DNN model analysis and results**

```
DNN MODEL SUMMARY REPORT
========================================================================
========================================================================

Training Time : 436.8488972187042  seconds
Testing Time : 2.285269260406494  seconds

Training Accuracy : 1.0
Testing Accuracy : 1.0

CLASSIFICATION REPORT

              precision    recall  f1-score   support

    Phishing       1.00      1.00      1.00      1511
  Legitimate       1.00      1.00      1.00      1489

    accuracy                           1.00      3000
   macro avg       1.00      1.00      1.00      3000
weighted avg       1.00      1.00      1.00      3000

CONFUSION MATRIX
```




```
MODEL EVALUATION SCORES

Accuracy Score: 100.0
AUC Score: 100.0
Precision Score: 100.0
```

**Figure 55: Code block showing details of the execution of code block in figure 54**

```
In [60]:
dnn_performance_df = pd.DataFrame({
    "DNN": [model_scores[dnn_label][accuracy_label], model_scores[dnn_label][auc_label], model_scor
}, index=[accuracy_label, auc_label, precision_label])

In [61]:
dnn_performance_df

Out[61]:
```

| | DNN |
|---|---|
| Accuracy | 100.0 |
| AUC | 100.0 |
| Precision | 100.0 |

**Figure 56: These code blocks showing the code use (i) to convert output of analyzed DNN model into panda's data frame, (ii) to view the summary of analyzed DNN model in tabular form**

```
In [62]:
performance_df = pd.DataFrame({
    "SVM": [model_scores[svm_label][accuracy_label], model_scores[svm_label][auc_label], model_scor
    "RFC": [model_scores[rfc_label][accuracy_label], model_scores[rfc_label][auc_label], model_scor
    "DNN": [model_scores[dnn_label][accuracy_label], model_scores[dnn_label][auc_label], model_scor
}, index=[accuracy_label, auc_label, precision_label])

In [63]:
performance_df

Out[63]:
```

| | SVM | RFC | DNN |
|---|---|---|---|
| Accuracy | 100.0 | 100.0 | 100.0 |
| AUC | 100.0 | 100.0 | 100.0 |
| Precision | 100.0 | 100.0 | 100.0 |

**Figure 57: These code blocks showing the code use (i) to convert all analyzed models (SVM, RFC, and DNN) into panda's data frame, (ii) to view the summary of all analyzed models in tabular form**

```
In [64]:
performance_df.plot(kind="bar")
plt.title("Phishing Email Classification Performance Multiple Bar Chart for Optimized Models")
plt.xlabel("Evaluation Metrics")
plt.ylabel("Evaluation in Percentage(%)")

Out[64]: Text(0, 0.5, 'Evaluation in Percentage(%)')
```

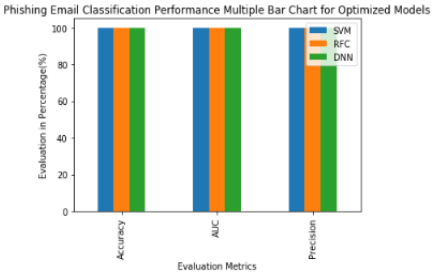



**Figure 58: Code block showing the code use to plot a multiple bar chart comparing all models implemented and the output generated by executing the code**

# 5 Conclusion

This configuration document shows the instructions used to perform the code implementation of the research project. The implementation uses Jose Nazario's phishing email corpus and Enron email dataset to analyse two machine learning algorithms namely Support Vector Machine (SVM) and Random Forest Classifier (RFC) and a deep learning algorithm namely

Deep Neural Network (DNN). The code blocks, the plots created and the table shown in this document helped in the actualization of the set objectives of this research work.

Any Research personnel who intend to carry out this same research using the same dataset and the implemented algorithms is sure to obtain similar results as seen from the output presented in this analysis following the instruction outlined in this document.

# 6 References

2022. [online] Available at: <https://www.datacamp.com/community/tutorials/installing-anaconda-windows> [Accessed 18 April 2022].

Docs.anaconda.com. 2022. *Installing on Windows — Anaconda documentation*. [online] Available at: <https://docs.anaconda.com/anaconda/install/windows/> [Accessed 18 April 2022].

Docs.anaconda.com. 2022. *TensorFlow — Anaconda documentation*. [online] Available at: <https://docs.anaconda.com/anaconda/user-guide/tasks/tensorflow/> [Accessed 18 April 2022].

Medium. 2022. *Install Tensorflow and Keras using Anaconda Navigator—without the command line*. [online] Available at: <https://towardsdatascience.com/https-medium-com-ekapope-v-install-tensorflow-and-keras-using-anaconda-navigator-without-command-line-b0bc41dbd038> [Accessed 18 April 2022].