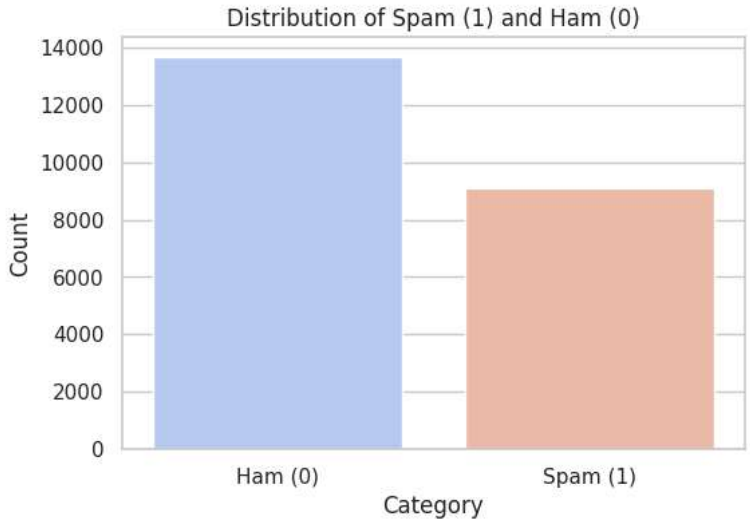
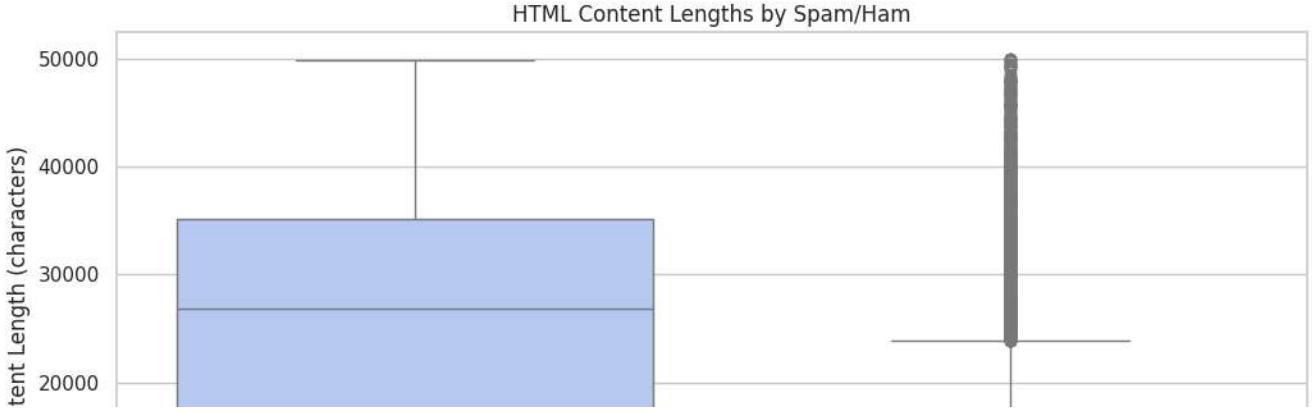
EXPLORATORY DATA ANALYSIS

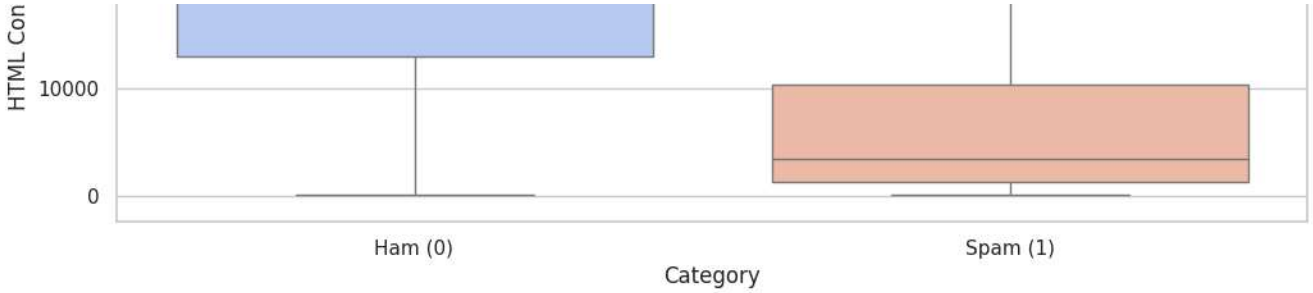
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from bs4 import BeautifulSoup

df = pd.read_csv("websites.csv")

# Dataset Overview
print("Dataset Shape:", df.shape)
print("Column Information:\n")
print(df.info())
print("\nMissing Values:\n", df.isnull().sum())
print("\nClass Distribution:\n", df['isPhish'].value_counts(normalize=True) * 100)

# Summary statistics
print("\nDescriptive Statistics:\n", df.describe())

# Visualizations
sns.set(style="whitegrid")

# 1. Class Distribution
plt.figure(figsize=(6, 4))
sns.countplot(data=df, x='isPhish', palette='coolwarm')
plt.title("Distribution of Spam (1) and Ham (0)")
plt.xlabel("Category")
plt.ylabel("Count")
plt.xticks(ticks=[0, 1], labels=["Ham (0)", "Spam (1)"])
plt.show()

# 2. HTML Content Length Distribution
plt.figure(figsize=(12, 6))
sns.boxplot(data=df, x='isPhish', y='html_length', palette='coolwarm')
plt.title("HTML Content Lengths by Spam/Ham")
plt.xlabel("Category")
plt.ylabel("HTML Content Length (characters)")
plt.xticks(ticks=[0, 1], labels=["Ham (0)", "Spam (1)"])
plt.show()

# Histogram of HTML Content Length
plt.figure(figsize=(10, 6))
df[df['isPhish'] == 0]['html_length'].plot.hist(bins=30, alpha=0.5, color='blue', label='Ham (0)')
df[df['isPhish'] == 1]['html_length'].plot.hist(bins=30, alpha=0.5, color='red', label='Spam (1)')
plt.title("HTML Content Length Distribution by Category")
plt.xlabel("HTML Content Length (characters)")
plt.ylabel("Frequency")
plt.legend()
plt.show()

# Density Plot of HTML Content Length
plt.figure(figsize=(12, 8))
sns.kdeplot(df[df['isPhish'] == 0]['html_length'], label='Ham (0)', color='blue', fill=True)
sns.kdeplot(df[df['isPhish'] == 1]['html_length'], label='Spam (1)', color='red', fill=True)
plt.title("HTML Length Distribution by Category (Spam vs Ham)")
plt.xlabel("HTML Content Length (characters)")
plt.ylabel("Density")
plt.legend()
plt.show()

# Correlation Matrix (if applicable)
numeric_features = df.select_dtypes(include=['float64', 'int64'])
correlation_matrix = numeric_features.corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', vmin=-1, vmax=1)
plt.title("Correlation Matrix of Numeric Features")
plt.show()

# Save Cleaned Data and Visualizations
df.to_csv("cleaned_dataset.csv", index=False)
```

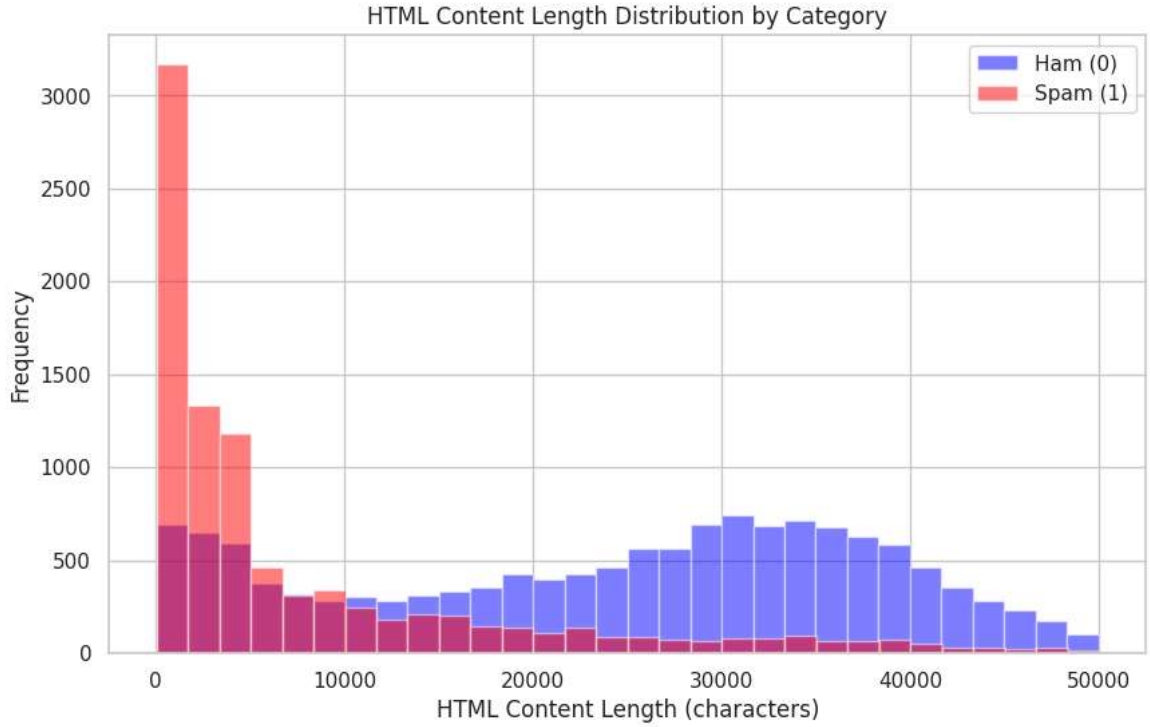
```
Dataset Shape: (22777, 4)
Column Information:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22777 entries, 0 to 22776
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   isPhish      22777 non-null  int64 
 1   Data         22777 non-null  object
 2   Text         22777 non-null  object
 3   html_length  22777 non-null  int64 
dtypes: int64(2), object(2)
memory usage: 711.9+ KB
None

Missing Values:
 isPhish        0
Data            0
Text            0
html_length     0
dtype: int64

Class Distribution:
 isPhish
0    59.999122
1    40.000878
Name: proportion, dtype: float64

Descriptive Statistics:
             isPhish   html_length
count  22777.000000  22777.000000
mean       0.400009  17784.898231
std        0.489910  14778.564459
min        0.000000     51.000000
25%        0.000000   3307.000000
50%        0.000000  15523.000000
75%        1.000000  31171.000000
max        1.000000  49999.000000
<ipython-input-3-9d247c6ce6aa>:24: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.countplot(data=df, x='isPhish', palette='coolwarm')
```

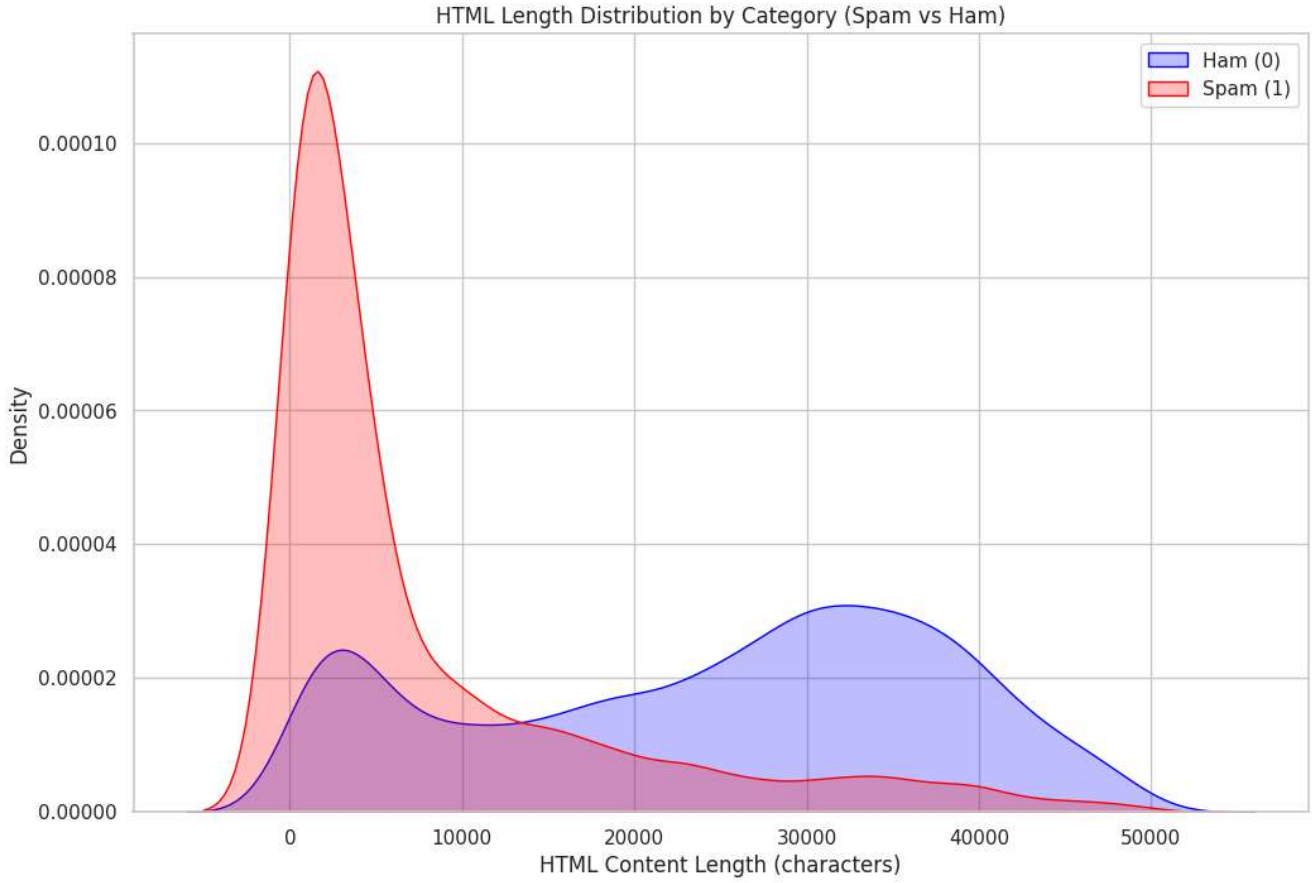



```
<ipython-input-3-9d247c6ce6aa>:33: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.boxplot(data=df, x='isPhish', y='html_length', palette='coolwarm')
```

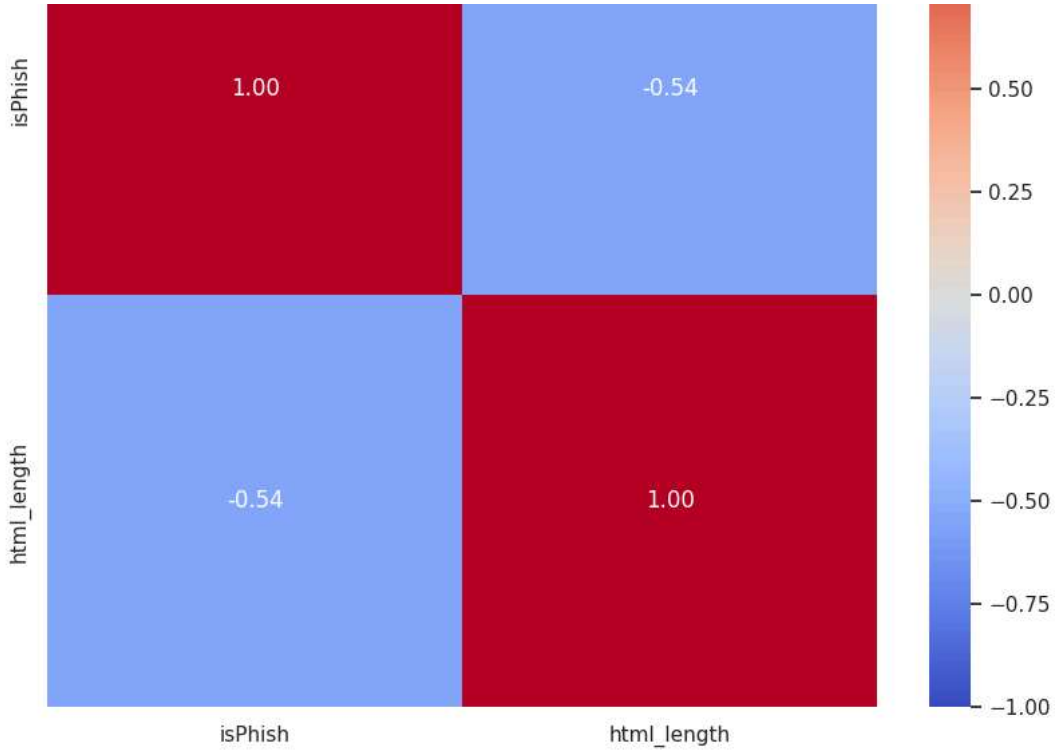



HTML Con
10000
0
Ham (0)
Spam (1)
Category

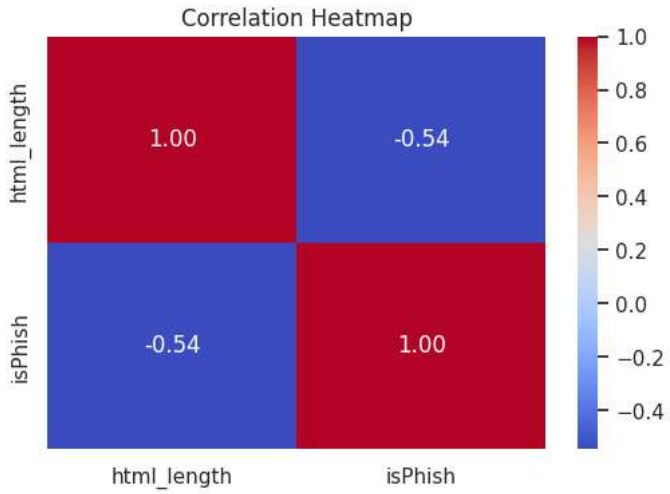

HTML Content Length Distribution by Category
Ham (0)
Spam (1)
Frequency
3000
2500
2000
1500
1000
500
0
0
10000
20000
30000
40000
50000
HTML Content Length (characters)


HTML Length Distribution by Category (Spam vs Ham)
Ham (0)
Spam (1)
Density
0.00010
0.00008
0.00006
0.00004
0.00002
0.00000
0
10000
20000
30000
40000
50000
HTML Content Length (characters)


Correlation Matrix of Numeric Features
1.00
0.75


isPhish
html_length
1.00
-0.54
-0.54
1.00
isPhish
html_length
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Compute correlation
correlation_matrix = df[['html_length', 'isPhish']].corr()

# Generate heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', cbar=True)
plt.title('Correlation Heatmap')
plt.show()
```



1: Data Cleaning

Extract Text from HTML: Remove HTML tags using libraries like BeautifulSoup

```
import pandas as pd
import numpy as np


# Step 1: Load the preprocessed dataset
df = pd.read_csv("websites.csv")
```

```
from bs4 import BeautifulSoup

def extract_text_from_html(html_content):
    soup = BeautifulSoup(html_content, 'html.parser')
    return soup.get_text()
```

Applying this function to Data column:

```
df['cleaned_text'] = df['Data'].apply(extract_text_from_html)
```

```
<ipython-input-5-549ce8b8e97b>:4: XMLParsedAsHTMLWarning: It looks like you're using an HTML parser to parse an XML document.

Assuming this really is an XML document, what you're doing might work, but you should know that using an XML parser will be more rel

If you want or need to use an HTML parser on this document, you can make this warning go away by filtering it. To do that, run this

    from bs4 import XMLParsedAsHTMLWarning
    import warnings

    warnings.filterwarnings("ignore", category=XMLParsedAsHTMLWarning)

  soup = BeautifulSoup(html_content, 'html.parser')
```

Step 2: Text Preprocessing

We will now preprocess the cleaned text:

2.1. Clean Text Remove unnecessary characters like URLs, special symbols, and digits.

```
import re
def clean_text(text):
    text = re.sub(r'http\S+|www.\S+', '', text)  # Remove URLs
    text = re.sub(r'<.*?>', '', text)  # Remove any remaining HTML tags
    text = re.sub(r'\d+', '', text)  # Remove digits
    text = re.sub(r'[^\w\s]', '', text)  # Remove special characters
    return text.lower()  # Convert to lowercase



df['cleaned_text'] = df['cleaned_text'].apply(clean_text)
```

2.2. Tokenize and Remove Stopwords **bold text**# Tokenize text into words and remove common stopwords (e.g., "the", "and").

```
import nltk
nltk.download('punkt', download_dir='/root/nltk_data')
nltk.download('stopwords', download_dir='/root/nltk_data')
nltk.download('wordnet', download_dir='/root/nltk_data')
nltk.download('omw-1.4', download_dir='/root/nltk_data')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
True
```

```
import nltk

# Download the 'punkt_tab' resource
nltk.download('punkt_tab', download_dir='/root/nltk_data')

from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords


stop_words = set(stopwords.words('english'))

def tokenize_and_remove_stopwords(text):
    tokens = word_tokenize(text)
    return [word for word in tokens if word not in stop_words]

df['tokens'] = df['cleaned_text'].apply(tokenize_and_remove_stopwords)
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

2.3. Lemmatization Reduce words to their root form (e.g., "running" → "run").

```
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')

lemmatizer = WordNetLemmatizer()

def lemmatize_tokens(tokens):
    return [lemmatizer.lemmatize(token) for token in tokens]

df['lemmatized_tokens'] = df['tokens'].apply(lemmatize_tokens)
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Step 3: Feature Representation (Converting Text to Numbers)

3.1. Convert Tokens to Sequences Use Tokenizer from Keras to encode words as integers.

```
!pip install tensorflow
from tensorflow.keras.preprocessing.text import Tokenizer

tokenizer = Tokenizer()
df['text_for_model'] = df['lemmatized_tokens'].apply(lambda tokens: ' '.join(tokens))
tokenizer.fit_on_texts(df['text_for_model'])
sequences = tokenizer.texts_to_sequences(df['text_for_model'])
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.11/dist-packages (2.18.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (25.2.10)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.6
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from tensorflow) (24.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3 in /usr/local/lib/py
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from tensorflow) (75.1.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.5.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (4.12.2)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.2)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.70.0)
Requirement already satisfied: tensorboard<2.19,>=2.18 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.18.0)
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.9.0)
Requirement already satisfied: numpy<2.1.0,>=1.26.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.26.4)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.12.1)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.11/dist-packages (from astunparse>=1.6.0->tensorflow) (0
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.14.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensor
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.10
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensorflow
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensorflow
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.11/dist-packages (from werkzeug>=1.0.1->tensorboard<2.19,
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorflow
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorfl
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.5.0
```

3.2. Padding Sequences

Ensure sequences have a uniform length.

```
from keras.preprocessing.sequence import pad_sequences

max_length = 200  # Define the sequence length
padded_sequences = pad_sequences(sequences, maxlen=max_length, padding='post')



# Save preprocessed data, tokenizer, and sequences
df.to_csv("preprocessed_text.csv", index=False)  # Save preprocessed DataFrame


import pickle


with open("tokenizer.pkl", "wb") as handle:  # Save tokenizer
    pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)

np.save("padded_sequences.npy", padded_sequences)  # Save padded sequences
```

RELOADING THE PRE PROCESSED PADDING SEQUENCES

```
# Load the preprocessed DataFrame
df = pd.read_csv("preprocessed_text.csv")
```

```
# Load the tokenizer
with open("tokenizer.pkl", "rb") as handle:
    tokenizer = pickle.load(handle)

# Load the padded sequences
padded_sequences = np.load("padded_sequences.npy")
```

GloVe Embeddings

GloVe (Global Vectors for Word Representation) is a pre-trained word embedding that captures the relationships between words. For example:

"King" - "Man" + "Woman" = "Queen" Here, we load GloVe embeddings and create an embedding matrix that maps each word in your vocabulary to its GloVe vector.

Why? This provides our model with more context and semantic understanding without needing massive amounts of data.

```
import numpy as np

# Path to the GloVe embeddings file
glove_file = 'glove.6B.100d.txt'  # Adjust the file path if necessary

# Load GloVe embeddings into a dictionary
embedding_index = {}
with open(glove_file, 'r', encoding='utf-8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        vector = np.asarray(values[1:], dtype='float32')
        embedding_index[word] = vector

print(f"Loaded {len(embedding_index)} word vectors from GloVe.")
```

```
Loaded 400000 word vectors from GloVe.
```

Create the Embedding Matrix

creating an embedding matrix that maps tokenizer's vocabulary to the GloVe embeddings.

```
embedding_dim = 100  # This should match the dimensions of the GloVe vectors you're using (e.g., 100d).

# Initialize the embedding matrix
embedding_matrix = np.zeros((len(tokenizer.word_index) + 1, embedding_dim))

# Map words in your tokenizer to GloVe embeddings
for word, index in tokenizer.word_index.items():
    embedding_vector = embedding_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[index] = embedding_vector

print(f"Embedding matrix shape: {embedding_matrix.shape}")
```

```
Embedding matrix shape: (305435, 100)
```

Saving the Embedding Matrix

```
import numpy as np

# Save the embedding matrix
np.save('embedding_matrix.npy', embedding_matrix)
print("Embedding matrix saved!")
```

```
Embedding matrix saved!
```

```
#Load the Embedding Matrix
#When you need to load the embedding matrix, use:

# Load the embedding matrix
embedding_matrix = np.load('embedding_matrix.npy')
print("Embedding matrix loaded!")
```

```
Embedding matrix loaded!
```

Step 4: Dataset Preparation

X and y: X = padded_sequences y = df['isPhish']

```
import numpy as np


X = np.array(padded_sequences)
y = df['isPhish'].values
```

2.Split into training and test sets:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

Decision Tree Implementation with 10-Fold Cross Validation

Train the Decision Tree:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix

# Create pipeline: StandardScaler + DecisionTreeClassifier
pipeline_dt = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', DecisionTreeClassifier(random_state=42))
])

# Perform 10-fold cross-validation
cv_accuracy_dt = cross_val_score(pipeline_dt, X_train, y_train, cv=10, scoring='accuracy')
cv_f1_dt = cross_val_score(pipeline_dt, X_train, y_train, cv=10, scoring='f1')
cv_recall_dt = cross_val_score(pipeline_dt, X_train, y_train, cv=10, scoring='recall')

# Display results
print(f"Decision Tree - Mean Accuracy: {cv_accuracy_dt.mean():.4f}")
print(f"Decision Tree - Mean F1 Score: {cv_f1_dt.mean():.4f}")
print(f"Decision Tree - Mean Recall (Attack): {cv_recall_dt.mean():.4f}")

# Fit the model on the entire training set
pipeline_dt.fit(X_train, y_train)

# Predict on test set
y_pred_dt = pipeline_dt.predict(X_test)

# Evaluate the model on test data
print("Decision Tree Classification Report:\n", classification_report(y_test, y_pred_dt))
print("Decision Tree Confusion Matrix:\n", confusion_matrix(y_test, y_pred_dt))
```

```
Decision Tree - Mean Accuracy: 0.8469
Decision Tree - Mean F1 Score: 0.8128
Decision Tree - Mean Recall (Attack): 0.8308
Decision Tree Classification Report:
               precision    recall  f1-score   support

            0       0.88      0.86      0.87      2734
            1       0.79      0.83      0.81      1822

    accuracy                           0.85      4556
   macro avg       0.84      0.84      0.84      4556
weighted avg       0.85      0.85      0.85      4556

Decision Tree Confusion Matrix:
 [[2342  392]
 [ 311 1511]]
```

Random Forest

```
from sklearn.pipeline import Pipeline # Importing the Pipeline class
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix

# Create pipeline: StandardScaler + RandomForestClassifier
pipeline_rf = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', RandomForestClassifier(random_state=42))
])

# Perform 10-fold cross-validation
cv_accuracy_rf = cross_val_score(pipeline_rf, X_train, y_train, cv=10, scoring='accuracy')
cv_f1_rf = cross_val_score(pipeline_rf, X_train, y_train, cv=10, scoring='f1')
cv_recall_rf = cross_val_score(pipeline_rf, X_train, y_train, cv=10, scoring='recall')

# Display results
print(f"Random Forest - Mean Accuracy: {cv_accuracy_rf.mean():.4f}")
print(f"Random Forest - Mean F1 Score: {cv_f1_rf.mean():.4f}")
print(f"Random Forest - Mean Recall (Attack): {cv_recall_rf.mean():.4f}")

# Fit the model on the entire training set
pipeline_rf.fit(X_train, y_train)

# Predict on test set
y_pred_rf = pipeline_rf.predict(X_test)

# Evaluate the model on test data
print("Random Forest Classification Report:\n", classification_report(y_test, y_pred_rf))
print("Random Forest Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rf))
```

```
Random Forest - Mean Accuracy: 0.8317
Random Forest - Mean F1 Score: 0.7724
Random Forest - Mean Recall (Attack): 0.7138
Random Forest Classification Report:
                precision    recall  f1-score   support

           0       0.84      0.90      0.87      2734
           1       0.83      0.74      0.78      1822

    accuracy                           0.83      4556
   macro avg       0.83      0.82      0.82      4556
weighted avg       0.83      0.83      0.83      4556

Random Forest Confusion Matrix:
 [[2460  274]
 [ 478 1344]]
```

ADDABOOST

```
from sklearn.ensemble import AdaBoostClassifier

# Create pipeline: StandardScaler + AdaBoostClassifier
pipeline_ab = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', AdaBoostClassifier(random_state=42))
])

# Perform 10-fold cross-validation
cv_accuracy_ab = cross_val_score(pipeline_ab, X_train, y_train, cv=10, scoring='accuracy')
cv_f1_ab = cross_val_score(pipeline_ab, X_train, y_train, cv=10, scoring='f1')
cv_recall_ab = cross_val_score(pipeline_ab, X_train, y_train, cv=10, scoring='recall')

# Display results
print(f"AdaBoost - Mean Accuracy: {cv_accuracy_ab.mean():.4f}")
print(f"AdaBoost - Mean F1 Score: {cv_f1_ab.mean():.4f}")
print(f"AdaBoost - Mean Recall (Attack): {cv_recall_ab.mean():.4f}")

# Fit the model on the entire training set
pipeline_ab.fit(X_train, y_train)

# Predict on test set
y_pred_ab = pipeline_ab.predict(X_test)

# Evaluate the model on test data
print("AdaBoost Classification Report:\n", classification_report(y_test, y_pred_ab))
print("AdaBoost Confusion Matrix:\n", confusion_matrix(y_test, y_pred_ab))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the defau
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the defau
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the defau
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the defau
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the defau
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the defau
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the defau
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the defau
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the defau
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the defau
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the defau
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the defau
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the defau
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the defau
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the defau
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the defau
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the defau
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the defau
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the defau
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the defau
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the defau
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the defau
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the defau
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the defau
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the defau
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the defau
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the defau
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the defau
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the defau
```

KNN

```
from sklearn.neighbors import KNeighborsClassifier

# Create pipeline: StandardScaler + KNeighborsClassifier
pipeline_knn = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', KNeighborsClassifier())
])

# Perform 10-fold cross-validation
cv_accuracy_knn = cross_val_score(pipeline_knn, X_train, y_train, cv=10, scoring='accuracy')
cv_f1_knn = cross_val_score(pipeline_knn, X_train, y_train, cv=10, scoring='f1')
cv_recall_knn = cross_val_score(pipeline_knn, X_train, y_train, cv=10, scoring='recall')

# Display results
print(f"KNN - Mean Accuracy: {cv_accuracy_knn.mean():.4f}")
print(f"KNN - Mean F1 Score: {cv_f1_knn.mean():.4f}")
print(f"KNN - Mean Recall (Attack): {cv_recall_knn.mean():.4f}")

# Fit the model on the entire training set
pipeline_knn.fit(X_train, y_train)

# Predict on test set
y_pred_knn = pipeline_knn.predict(X_test)
```

```
# Evaluate the model on test data
print("KNN Classification Report:\n", classification_report(y_test, y_pred_knn))
print("KNN Confusion Matrix:\n", confusion_matrix(y_test, y_pred_knn))
```

```
KNN - Mean Accuracy: 0.7177
KNN - Mean F1 Score: 0.6778
KNN - Mean Recall (Attack): 0.7385
KNN Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.72      0.76      1366
           1       0.63      0.71      0.67       911

    accuracy                           0.72      2277
   macro avg       0.71      0.72      0.71      2277
weighted avg       0.73      0.72      0.72      2277

KNN Confusion Matrix:
 [[989 377]
 [263 648]]
```

SVM

```
from sklearn.svm import SVC

# Create pipeline: StandardScaler + SVC
pipeline_svm = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', SVC(random_state=42))
])

# Perform 10-fold cross-validation
cv_accuracy_svm = cross_val_score(pipeline_svm, X_train, y_train, cv=10, scoring='accuracy')
cv_f1_svm = cross_val_score(pipeline_svm, X_train, y_train, cv=10, scoring='f1')
cv_recall_svm = cross_val_score(pipeline_svm, X_train, y_train, cv=10, scoring='recall')

# Display results
print(f"SVM - Mean Accuracy: {cv_accuracy_svm.mean():.4f}")
print(f"SVM - Mean F1 Score: {cv_f1_svm.mean():.4f}")
print(f"SVM - Mean Recall (Attack): {cv_recall_svm.mean():.4f}")

# Fit the model on the entire training set
pipeline_svm.fit(X_train, y_train)

# Predict on test set
y_pred_svm = pipeline_svm.predict(X_test)

# Evaluate the model on test data
print("SVM Classification Report:\n", classification_report(y_test, y_pred_svm))
print("SVM Confusion Matrix:\n", confusion_matrix(y_test, y_pred_svm))
```

```
SVM - Mean Accuracy: 0.7693
SVM - Mean F1 Score: 0.7210
SVM - Mean Recall (Attack): 0.7466
SVM Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.79      0.80      1366
           1       0.69      0.71      0.70       911

    accuracy                           0.76      2277
   macro avg       0.75      0.75      0.75      2277
weighted avg       0.76      0.76      0.76      2277

SVM Confusion Matrix:
 [[1076  290]
 [ 260  651]]
```

ONE CLASS SVM

```
import numpy as np
import pandas as pd
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import classification_report, confusion_matrix
import joblib  # For saving the trained model

# ✅ Load Data
df = pd.read_csv("preprocessed_text_clean.csv")
X = np.load("padded_sequences_new.npy")

# ✅ Ensure Data Alignment
valid_indices = df['text_for_model'].notna().values.nonzero()[0]
X = X[valid_indices]
y = df.loc[valid_indices, "isPhish"].values

# ✅ Split into Train & Test Sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)



# ✅ Create pipeline: StandardScaler + SVM
pipeline_svm = Pipeline([
    ('scaler', StandardScaler()),  # Standardizing features
    ('classifier', SVC(kernel='rbf', random_state=42))  # Using RBF kernel for better separation
])

# ✅ Perform 10-fold cross-validation
cv_accuracy_svm = cross_val_score(pipeline_svm, X_train, y_train, cv=10, scoring='accuracy')
cv_f1_svm = cross_val_score(pipeline_svm, X_train, y_train, cv=10, scoring='f1')
cv_recall_svm = cross_val_score(pipeline_svm, X_train, y_train, cv=10, scoring='recall')

# ✅ Display Cross-validation Results
print(f"SVM - Mean Accuracy: {cv_accuracy_svm.mean():.4f}")
print(f"SVM - Mean F1 Score: {cv_f1_svm.mean():.4f}")
print(f"SVM - Mean Recall (Attack): {cv_recall_svm.mean():.4f}")
```

```
SVM - Mean Accuracy: 0.6139
SVM - Mean F1 Score: 0.0009
SVM - Mean Recall (Attack): 0.0004
```

```
# ✅ Fit the Model on Full Training Set
pipeline_svm.fit(X_train, y_train)

# ✅ Predict on Test Data
y_pred_svm = pipeline_svm.predict(X_test)

# ✅ Evaluate Performance
print("SVM Classification Report:\n", classification_report(y_test, y_pred_svm))
print("SVM Confusion Matrix:\n", confusion_matrix(y_test, y_pred_svm))
```

```
SVM Classification Report:
                  precision    recall  f1-score   support

               0       0.61      1.00      0.76      2725
               1       0.50      0.00      0.00      1715

        accuracy                           0.61      4440
       macro avg       0.56      0.50      0.38      4440
    weighted avg       0.57      0.61      0.47      4440

    SVM Confusion Matrix:
     [[2724    1]
     [1714    1]]
```

Round 2 - SVM +RANDOM FOREST

```
import joblib
import numpy as np
import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split, cross_val_score, StratifiedKFold
from scipy.sparse import hstack

# ✅ Step 1: Load Preprocessed Data
```

```
df = pd.read_csv("preprocessed_text_clean.csv")
X_text = df["text_for_model"].fillna("")
y = df["isPhish"].values

# ✅ Step 2: Convert Text to **TF-IDF Features**
vectorizer = TfidfVectorizer(max_features=3000)
X_tfidf = vectorizer.fit_transform(X_text)

# ✅ Step 3: Extract **HTML Length** and Combine Features
X_html_length = df[["html_length"]].values
X_combined = hstack([X_tfidf, X_html_length])  # ✅ Final Input Features

# ✅ Save TF-IDF Vectorizer for Web App
joblib.dump(vectorizer, "tfidf_vectorizer.pkl")

# ✅ Step 4: 10-Fold Cross-Validation Setup
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# ✅ Step 5: Define **SVM Model**
svm_model = Pipeline([
    ('scaler', StandardScaler(with_mean=False)),
    ('classifier', SVC(C=10, gamma='scale', kernel='rbf', class_weight='balanced', probability=True, random_state=42))
])

# ✅ Step 6: Define **Random Forest Model**
rf_model = RandomForestClassifier(n_estimators=200, max_depth=15, random_state=42)

# ✅ Step 7: Perform 10-Fold Cross-Validation **Before Training on Full Dataset**
print("\n🚀 Performing 10-Fold Cross-Validation...")

svm_scores = cross_val_score(svm_model, X_combined, y, cv=cv, scoring='f1')  # 🔥 Use F1-score
rf_scores = cross_val_score(rf_model, X_combined, y, cv=cv, scoring='f1')

# ✅ Print Cross-Validation Results
print("\n📌 **SVM 10-Fold Cross-Validation F1-Scores:**", svm_scores)
print("📌 **SVM Mean F1-Score:** {:.4f}".format(svm_scores.mean()))
print("📌 **SVM Standard Deviation:** {:.4f}".format(svm_scores.std()))

print("\n📌 **Random Forest 10-Fold Cross-Validation F1-Scores:**", rf_scores)
print("📌 **Random Forest Mean F1-Score:** {:.4f}".format(rf_scores.mean()))
print("📌 **Random Forest Standard Deviation:** {:.4f}".format(rf_scores.std()))

# ✅ Step 8: Train-Test Split **With Stratification**
X_train, X_test, y_train, y_test = train_test_split(X_combined, y, test_size=0.2, stratify=y, random_state=42)

# ✅ Step 9: Train Final Models on **Train Set Only**
print("\n🚀 Training Final Models on Training Set...")
svm_model.fit(X_train, y_train)
rf_model.fit(X_train, y_train)

# ✅ Step 10: Save Models
joblib.dump(svm_model, "pipeline_svm.pkl")
joblib.dump(rf_model, "random_forest.pkl")

print("\n✅ Models Saved: SVM + Random Forest")

# ✅ Step 11: Evaluate Performance on **Test Set**
y_pred_svm = svm_model.predict(X_test)
y_pred_rf = rf_model.predict(X_test)

print("\n📌 **Final SVM Performance on Test Set:**")
print(classification_report(y_test, y_pred_svm))

print("\n📌 **Final Random Forest Performance on Test Set:**")
print(classification_report(y_test, y_pred_rf))
```

```
🚀 Performing 10-Fold Cross-Validation...

📌 **SVM 10-Fold Cross-Validation F1-Scores:** [0.86447761 0.85014925 0.88197299 0.87200957 0.88668224 0.88380282
 0.84267631 0.88862418 0.86966686 0.88402367]
📌 **SVM Mean F1-Score:** 0.8724
📌 **SVM Standard Deviation:** 0.0151

📌 **Random Forest 10-Fold Cross-Validation F1-Scores:** [0.83091226 0.82762623 0.83042973 0.83814008 0.83571837 0.82561464
 0.82332362 0.82906977 0.82292264 0.82407407]
📌 **Random Forest Mean F1-Score:** 0.8288
📌 **Random Forest Standard Deviation:** 0.0049

🚀 Training Final Models on Training Set...

✅ Models Saved: SVM + Random Forest
```

```
📌 **Final SVM Performance on Test Set:**
              precision    recall  f1-score   support

           0       0.91      0.93      0.92      2725
           1       0.89      0.85      0.87      1715

    accuracy                           0.90      4440
   macro avg       0.90      0.89      0.90      4440
weighted avg       0.90      0.90      0.90      4440


📌 **Final Random Forest Performance on Test Set:**
              precision    recall  f1-score   support

           0       0.90      0.88      0.89      2725
           1       0.81      0.84      0.83      1715

    accuracy                           0.86      4440
   macro avg       0.85      0.86      0.86      4440
weighted avg       0.86      0.86      0.86      4440
```

```
import joblib
import numpy as np
import pandas as pd
from sklearn.metrics import classification_report, confusion_matrix

# ✅ Load Saved Models
svm_model = joblib.load("pipeline_svm.pkl")
rf_model = joblib.load("random_forest.pkl")

# ✅ Load Preprocessed Data
df = pd.read_csv("preprocessed_text_clean.csv")
X_text = df["text_for_model"].fillna("")
y = df["isPhish"].values

# ✅ Load TF-IDF Vectorizer
vectorizer = joblib.load("tfidf_vectorizer.pkl")
X_tfidf = vectorizer.transform(X_text)

# ✅ Extract **HTML Length** and Combine Features
X_html_length = df[["html_length"]].values
from scipy.sparse import hstack
X_combined = hstack([X_tfidf, X_html_length])  # ✅ Final Input Features

# ✅ Split Data for Final Evaluation
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_combined, y, test_size=0.2, stratify=y, random_state=42)

# ✅ Predict on Test Set
y_pred_svm = svm_model.predict(X_test)
y_pred_rf = rf_model.predict(X_test)

# ✅ Print Performance Metrics
print("\n📌 **Final SVM Performance on Test Set:**")
print(classification_report(y_test, y_pred_svm))

print("\n📌 **Final Random Forest Performance on Test Set:**")
print(classification_report(y_test, y_pred_rf))

# ✅ Print Confusion Matrices
print("\n🔹 **Confusion Matrix (SVM):**")
print(confusion_matrix(y_test, y_pred_svm))

print("\n🔹 **Confusion Matrix (Random Forest):**")
print(confusion_matrix(y_test, y_pred_rf))

# ✅ Print Performance Metric Definitions
print("\n📊 **Performance Metrics Explained:**")
print("🔹 **Accuracy:** Measures the overall correctness of the model.")
print("🔹 **Precision:** Out of all the predicted phishing cases, how many were actually phishing?")
print("🔹 **Recall (Sensitivity):** Out of all the actual phishing cases, how many were correctly predicted?")
print("🔹 **F1-Score:** The balance between precision and recall (good F1-score means we are both accurate and catching enough phishing
print("🔹 **Support:** The number of actual samples for each class (0: Non-phishing, 1: Phishing).")
```

```
📌 **Final SVM Performance on Test Set:**
              precision    recall  f1-score   support

           0       0.91      0.93      0.92      2725
           1       0.89      0.85      0.87      1715

    accuracy                           0.90      4440
```

```
   macro avg       0.90      0.89      0.90      4440
weighted avg       0.90      0.90      0.90      4440


📌 **Final Random Forest Performance on Test Set:**
              precision    recall  f1-score   support

           0       0.90      0.88      0.89      2725
           1       0.81      0.84      0.83      1715

    accuracy                           0.86      4440
   macro avg       0.85      0.86      0.86      4440
weighted avg       0.86      0.86      0.86      4440


🔹 **Confusion Matrix (SVM):**
[[2544  181]
 [ 251 1464]]

🔹 **Confusion Matrix (Random Forest):**
[[2397  328]
 [ 280 1435]]

📊 **Performance Metrics Explained:**
🔹 **Accuracy:** Measures the overall correctness of the model.
🔹 **Precision:** Out of all the predicted phishing cases, how many were actually phishing?
🔹 **Recall (Sensitivity):** Out of all the actual phishing cases, how many were correctly predicted?
🔹 **F1-Score:** The balance between precision and recall (good F1-score means we are both accurate and catching enough phishing ca
🔹 **Support:** The number of actual samples for each class (0: Non-phishing, 1: Phishing).
```

Tune SVM Parameters

```
from sklearn.experimental import enable_halving_search_cv  # Required for HalvingGridSearchCV
from sklearn.model_selection import HalvingGridSearchCV

param_grid = {
    'classifier__C': [1, 10],
    'classifier__gamma': ['scale', 0.1]
}

grid_search_svm = GridSearchCV(svm_model,
             param_grid,
             cv=5,
             scoring='f1',
             verbose=3,
             n_jobs=-1)


X_subset = X_train[:5000]
y_subset = y_train[:5000]

grid_search_svm.fit(X_subset, y_subset)

print("✅ Best SVM Parameters:", grid_search_svm.best_params_)
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
✅ Best SVM Parameters: {'classifier__C': 10, 'classifier__gamma': 'scale'}
```

```
# Apply best parameters
best_svm = Pipeline([
    ('scaler', StandardScaler(with_mean=False)),
    ('classifier', SVC(C=10, gamma='scale', kernel='rbf', class_weight='balanced', probability=True, random_state=42))
])

# Train on full training set
best_svm.fit(X_train, y_train)

# Save the optimized model
joblib.dump(best_svm, "svm_best.pkl")
print("\n✅ Optimized SVM Model Saved!")

# Predict on test set
y_pred_svm = best_svm.predict(X_test)

# Print classification report
print("\n📌 **Final SVM Performance on Test Set:**")
print(classification_report(y_test, y_pred_svm))
```

```
✅ Optimized SVM Model Saved!

📌 **Final SVM Performance on Test Set:**
              precision    recall  f1-score   support

           0       0.91      0.93      0.92      2725
           1       0.89      0.85      0.87      1715

    accuracy                           0.90      4440
   macro avg       0.90      0.89      0.90      4440
weighted avg       0.90      0.90      0.90      4440
```

```
from sklearn.metrics import confusion_matrix

# Standard SVM Predictions
print("\n📌 **Final SVM Performance on Test Set:**")
print(classification_report(y_test, y_pred_svm))

# ✅ Confusion Matrix
cm = confusion_matrix(y_test, y_pred_svm)
print("\n📌 **Confusion Matrix (SVM):**")
print(cm)
```

```
📌 **Final SVM Performance on Test Set:**
              precision    recall  f1-score   support

           0       0.91      0.93      0.92      2725
           1       0.89      0.85      0.87      1715

    accuracy                           0.90      4440
   macro avg       0.90      0.89      0.90      4440
weighted avg       0.90      0.90      0.90      4440


📌 **Confusion Matrix (SVM):**
[[2544  181]
 [ 251 1464]]
```

```
y_pred_proba = best_svm.predict_proba(X_test)[:, 1]  # Get probability scores
threshold = 0.35  # Lower threshold to catch more phishing cases

y_pred_adjusted = (y_pred_proba > threshold).astype(int)

print("\n📌 **SVM with Adjusted Threshold (0.35) Performance:**")
print(classification_report(y_test, y_pred_adjusted))

from sklearn.metrics import confusion_matrix

# Standard SVM Predictions
print("\n📌 **Final SVM Performance on Test Set:**")
print(classification_report(y_test, y_pred_svm))

# ✅ Confusion Matrix
cm = confusion_matrix(y_test, y_pred_svm)
print("\n📌 **Confusion Matrix (SVM):**")
print(cm)
```

```
📌 **SVM with Adjusted Threshold (0.35) Performance:**
              precision    recall  f1-score   support

           0       0.92      0.92      0.92      2725
           1       0.87      0.88      0.87      1715

    accuracy                           0.90      4440
   macro avg       0.89      0.90      0.89      4440
weighted avg       0.90      0.90      0.90      4440


📌 **Final SVM Performance on Test Set:**
              precision    recall  f1-score   support

           0       0.91      0.93      0.92      2725
           1       0.89      0.85      0.87      1715

    accuracy                           0.90      4440
   macro avg       0.90      0.89      0.90      4440
weighted avg       0.90      0.90      0.90      4440
```

```
📌 **Confusion Matrix (SVM):**
[[2544  181]
 [ 251 1464]]
```

SMOTE

```
from imblearn.over_sampling import SMOTE

# ✅ Automatically balance phishing and legitimate cases
smote = SMOTE(sampling_strategy='auto', random_state=42)  # Auto balances classes
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

print("✅ After SMOTE - Class Distribution:")
print(pd.Series(y_resampled).value_counts())  # Check new balance
```

```
✅ After SMOTE - Class Distribution:
1    10902
0    10902
Name: count, dtype: int64
```

Double-click (or enter) to edit

```
# ✅ Train on Resampled Data
svm_model.fit(X_resampled, y_resampled)
rf_model.fit(X_resampled, y_resampled)

# ✅ Predict on Test Set (Original Distribution)
y_pred_svm = svm_model.predict(X_test)
y_pred_rf = rf_model.predict(X_test)

# ✅ Print Performance
from sklearn.metrics import classification_report, confusion_matrix

print("\n📌 **Final SVM Performance on Test Set:**")
print(classification_report(y_test, y_pred_svm))

print("\n📌 **Final Random Forest Performance on Test Set:**")
print(classification_report(y_test, y_pred_rf))

# ✅ Print Confusion Matrices
print("\n📌 **SVM Confusion Matrix:**")
print(confusion_matrix(y_test, y_pred_svm))

print("\n📌 **Random Forest Confusion Matrix:**")
print(confusion_matrix(y_test, y_pred_rf))
```

```
📌 **Final SVM Performance on Test Set:**
              precision    recall  f1-score   support

           0       0.90      0.94      0.92      2725
           1       0.89      0.84      0.87      1715

    accuracy                           0.90      4440
   macro avg       0.90      0.89      0.89      4440
weighted avg       0.90      0.90      0.90      4440


📌 **Final Random Forest Performance on Test Set:**
              precision    recall  f1-score   support

           0       0.93      0.79      0.86      2725
           1       0.73      0.91      0.81      1715

    accuracy                           0.84      4440
   macro avg       0.83      0.85      0.83      4440
weighted avg       0.86      0.84      0.84      4440


📌 **SVM Confusion Matrix:**
[[2549  176]
 [ 268 1447]]

📌 **Random Forest Confusion Matrix:**
[[2156  569]
 [ 157 1558]]
```

```
import joblib
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from scipy.sparse import hstack
from sklearn.feature_extraction.text import TfidfVectorizer

#  Load Data
df = pd.read_csv("preprocessed_text_clean.csv")
X_text = df["text_for_model"].fillna("")
y = df["isPhish"].values

#  Convert Text to TF-IDF Features
vectorizer = TfidfVectorizer(max_features=3000)
X_tfidf = vectorizer.fit_transform(X_text)

#  Extract HTML Length & Combine Features
X_html_length = df[["html_length"]].values
X_combined = hstack([X_tfidf, X_html_length])

#  Save Vectorizer for Deployment
joblib.dump(vectorizer, "tfidf_vectorizer.pkl")

#  Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_combined, y, test_size=0.2, stratify=y, random_state=42)

#  Apply SMOTE for Balancing
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

print("\n✅ After SMOTE Class Distribution:")
print(pd.Series(y_train_resampled).value_counts())

#  Train Final Random Forest Model
rf_final = RandomForestClassifier(n_estimators=500, max_depth=20, random_state=42)
rf_final.fit(X_train_resampled, y_train_resampled)

#  Save Final Model
joblib.dump(rf_final, "random_forest_final.pkl")
print("\n✅ Final Random Forest Model Saved!")

#  Predict on Test Set
y_pred_rf = rf_final.predict(X_test)

#  Print Performance
print("\n📌 **Final Random Forest Performance on Test Set:**")
print(classification_report(y_test, y_pred_rf))

#  Confusion Matrix
print("\n📌 **Confusion Matrix (Random Forest):**")
print(confusion_matrix(y_test, y_pred_rf))
```

```
✅ After SMOTE Class Distribution:
1    10902
0    10902
Name: count, dtype: int64

✅ Final Random Forest Model Saved!

📌 **Final Random Forest Performance on Test Set:**
              precision    recall  f1-score   support

           0       0.93      0.82      0.87      2725
           1       0.76      0.91      0.83      1715

    accuracy                           0.85      4440
   macro avg       0.85      0.86      0.85      4440
weighted avg       0.87      0.85      0.86      4440


📌 **Confusion Matrix (Random Forest):**
[[2236  489]
 [ 161 1554]]
```

Voting Classifier with SVM (No SMOTE) + Random Forest (With SMOTE)

```
import joblib
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from scipy.sparse import hstack

#  Step 1: Load Data
df = pd.read_csv("preprocessed_text_clean.csv")
X_text = df["text_for_model"].fillna("")
y = df["isPhish"].values

#  Step 2: Convert Text to TF-IDF Features
vectorizer = TfidfVectorizer(max_features=3000)
X_tfidf = vectorizer.fit_transform(X_text)

# Step 3: Extract HTML Length & Combine Features
X_html_length = df[["html_length"]].values
X_combined = hstack([X_tfidf, X_html_length])

# Save Vectorizer for Deployment
joblib.dump(vectorizer, "tfidf_vectorizer.pkl")

#  Step 4: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_combined, y, test_size=0.2, stratify=y, random_state=42)

#  Step 5: Apply SMOTE for RF (but NOT for SVM)
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

print("\n✅ After SMOTE Class Distribution:")
print(pd.Series(y_train_resampled).value_counts())

#  Step 6: Train **SVM (Without SMOTE)**
svm_model = Pipeline([
    ('scaler', StandardScaler(with_mean=False)),
    ('classifier', SVC(C=10, gamma='scale', kernel='rbf', class_weight='balanced', probability=True, random_state=42))
])
svm_model.fit(X_train, y_train)

# Step 7: Train **Random Forest (With SMOTE)**
rf_model = RandomForestClassifier(n_estimators=500, max_depth=20, random_state=42)
rf_model.fit(X_train_resampled, y_train_resampled)

#  Step 8: Create Voting Classifier (Soft Voting with Weighted Models)
voting_model = VotingClassifier(
    estimators=[('SVM', svm_model), ('RF', rf_model)],
    voting='soft',  # Soft Voting (Uses probability scores)
    weights=[1, 2]  # Higher weight for RF (Detects more phishing cases)
)
voting_model.fit(X_train, y_train)  # Train on original train set

# ✅ Step 9: Save Models
joblib.dump(svm_model, "svm_no_smote.pkl")
joblib.dump(rf_model, "rf_with_smote.pkl")
joblib.dump(voting_model, "voting_classifier.pkl")
print("\n✅ Models Saved: SVM + RF + Voting Classifier")

#  Step 10: Predict Probabilities
y_pred_proba = voting_model.predict_proba(X_test)[:, 1]  # Extract probability scores

# ✅ Apply Threshold (0.35)
threshold = 0.35
y_pred_adjusted = (y_pred_proba > threshold).astype(int)

#  Step 11: Evaluate Performance with Adjusted Threshold
print("\n📌 **Final Voting Classifier Performance (Threshold = 0.35):**")
print(classification_report(y_test, y_pred_adjusted))

# Confusion Matrix
print("\n📌 **Confusion Matrix (Voting Classifier with Threshold 0.35):**")
print(confusion_matrix(y_test, y_pred_adjusted))
```

```
✅ After SMOTE Class Distribution:
```

```
1    10902
0    10902
Name: count, dtype: int64

✅ Models Saved: SVM + RF + Voting Classifier

📌 **Final Voting Classifier Performance (Threshold = 0.35):**
              precision    recall  f1-score   support

           0       0.95      0.84      0.89      2725
           1       0.78      0.93      0.85      1715

    accuracy                           0.87      4440
   macro avg       0.86      0.88      0.87      4440
weighted avg       0.88      0.87      0.87      4440


📌 **Confusion Matrix (Voting Classifier with Threshold 0.35):**
[[2280  445]
 [ 127 1588]]
```

```
# Step 4: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(
    X_combined, y, test_size=0.2, stratify=y, random_state=42
)

#  Save Test Data for Future Evaluation
joblib.dump(X_test, "X_test.pkl")
joblib.dump(y_test, "y_test.pkl")

print("\n Test Data Saved!")
```

```
✅ Test Data Saved!
```

```
import joblib
import numpy as np
import pandas as pd
from sklearn.metrics import classification_report, confusion_matrix

# Load Saved Models & Data
voting_model = joblib.load("voting_classifier.pkl")
X_test = joblib.load("X_test.pkl")
y_test = joblib.load("y_test.pkl")

#  Predict Probabilities
y_pred_proba = voting_model.predict_proba(X_test)[:, 1]

#  Experiment with Thresholds
for threshold in [0.35, 0.38, 0.40]:
    print(f"\n🔍 **Evaluating with Threshold = {threshold}**")
    y_pred_adjusted = (y_pred_proba > threshold).astype(int)

    # ✅ Print Metrics
    print("\n📌 **Performance Report:**")
    print(classification_report(y_test, y_pred_adjusted))

    # ✅ Print Confusion Matrix
    print("\n📌 **Confusion Matrix:**")
    print(confusion_matrix(y_test, y_pred_adjusted))
```

```
🔍 **Evaluating with Threshold = 0.35**

📌 **Performance Report:**
              precision    recall  f1-score   support

           0       0.95      0.84      0.89      2725
           1       0.78      0.93      0.85      1715

    accuracy                           0.87      4440
   macro avg       0.86      0.88      0.87      4440
weighted avg       0.88      0.87      0.87      4440


📌 **Confusion Matrix:**
[[2280  445]
 [ 127 1588]]

🔍 **Evaluating with Threshold = 0.38**

📌 **Performance Report:**
              precision    recall  f1-score   support
```

```
              0       0.94      0.87      0.90      2725
              1       0.82      0.92      0.86      1715

       accuracy                           0.89      4440
      macro avg       0.88      0.89      0.88      4440
   weighted avg       0.89      0.89      0.89      4440


📌 **Confusion Matrix:**
[[2368  357]
 [ 142 1573]]

🔍 **Evaluating with Threshold = 0.4**

📌 **Performance Report:**
              precision    recall  f1-score   support

           0       0.94      0.89      0.91      2725
           1       0.84      0.91      0.87      1715

    accuracy                           0.90      4440
   macro avg       0.89      0.90      0.89      4440
weighted avg       0.90      0.90      0.90      4440


📌 **Confusion Matrix:**
[[2426  299]
 [ 157 1558]]
```

Best Threshold Choice

Lower FP (Legitimate as Phishing)? → Choose 0.40

Lower FN (Phishing as Legitimate)? → Choose 0.35

Balanced Approach? → 0.38 is the best trade-off!

Suggested Final Model Decision

*Use Threshold = 0.38 *

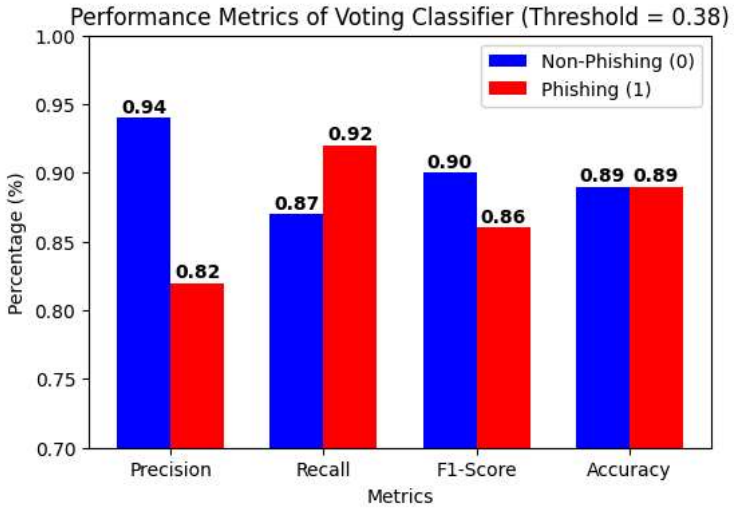
It balances False Positives and False Negatives while maintaining high accuracy (89%) and good phishing recall (92%).

RESULTS

Performance Metrics of Voting Classifier (Threshold = 0.38)

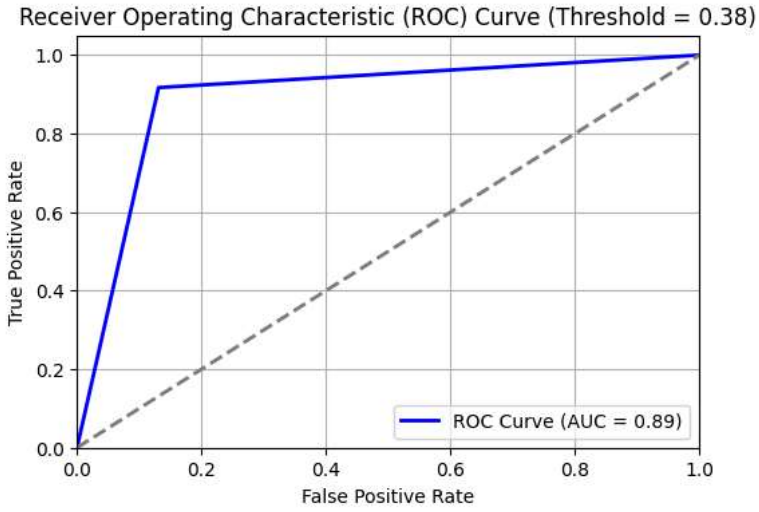
```
import numpy as np
import matplotlib.pyplot as plt # Import the pyplot module from matplotlib


# Performance Metrics
metrics = ["Precision", "Recall", "F1-Score", "Accuracy"]
non_phishing = [0.94, 0.87, 0.90, 0.89]
phishing = [0.82, 0.92, 0.86, 0.89]

x = np.arange(len(metrics))  # X-axis locations
width = 0.35  # Bar width

# Plot
fig, ax = plt.subplots(figsize=(6, 4))
bars1 = ax.bar(x - width/2, non_phishing, width, label="Non-Phishing (0)", color="blue")
bars2 = ax.bar(x + width/2, phishing, width, label="Phishing (1)", color="red")

# Labels & Formatting
ax.set_xlabel("Metrics")
ax.set_ylabel("Percentage (%)")
ax.set_title("Performance Metrics of Voting Classifier (Threshold = 0.38)")
ax.set_xticks(x)
ax.set_xticklabels(metrics)
ax.legend()

# Annotate Bars
for bars in [bars1, bars2]:
    for bar in bars:
        height = bar.get_height()
        ax.annotate(f'{height:.2f}', xy=(bar.get_x() + bar.get_width() / 2, height),
                    xytext=(0, 3), textcoords="offset points", ha='center', fontsize=10, fontweight='bold')
```

```
plt.ylim(0.7, 1)  # Y-axis limit
plt.show()
```



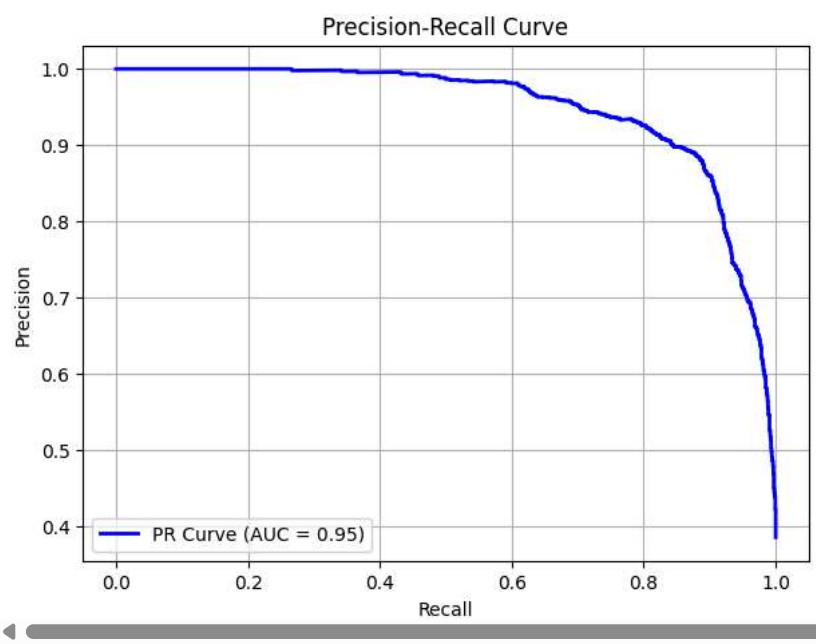
Receiver Operating Characteristic (ROC) Curve (Threshold = 0.38)

```
from sklearn.metrics import roc_curve, auc
import joblib # Import joblib for loading models

# Load Saved Models & Data
voting_model = joblib.load("voting_classifier.pkl")  # Load your voting classifier
X_test = joblib.load("X_test.pkl")
y_test = joblib.load("y_test.pkl")

# Predict Probabilities
y_pred_proba = voting_model.predict_proba(X_test)[:, 1]  # Get probability scores

# Assuming y_test contains the true labels and y_probs are predicted probabilities for class 1
# Replace y_true with y_test (true labels)
# Calculate y_pred_adjusted based on your optimal threshold (0.38 in this case)
threshold = 0.38
y_pred_adjusted = (y_pred_proba > threshold).astype(int)
fpr, tpr, _ = roc_curve(y_test, y_pred_adjusted)
roc_auc = auc(fpr, tpr)

# Plot ROC Curve
plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver Operating Characteristic (ROC) Curve (Threshold = 0.38)")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```



Precision-Recall (PR) Curve

```
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve, auc
import joblib

# Load saved test data
X_test = joblib.load("X_test.pkl")
y_test = joblib.load("y_test.pkl")

# Load the saved Voting Classifier model
voting_clf = joblib.load("voting_classifier.pkl")  # Ensure the model is saved with this name

# Get predicted probabilities
y_scores = voting_clf.predict_proba(X_test)[:, 1]  # Probabilities for class 1 (phishing)

# Compute Precision-Recall curve
precision, recall, _ = precision_recall_curve(y_test, y_scores)
pr_auc = auc(recall, precision)

# Plot Precision-Recall Curve
plt.figure(figsize=(7, 5))
plt.plot(recall, precision, color="blue", lw=2, label=f"PR Curve (AUC = {pr_auc:.2f})")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve")
plt.legend(loc="best")
plt.grid(True)
plt.show()
```

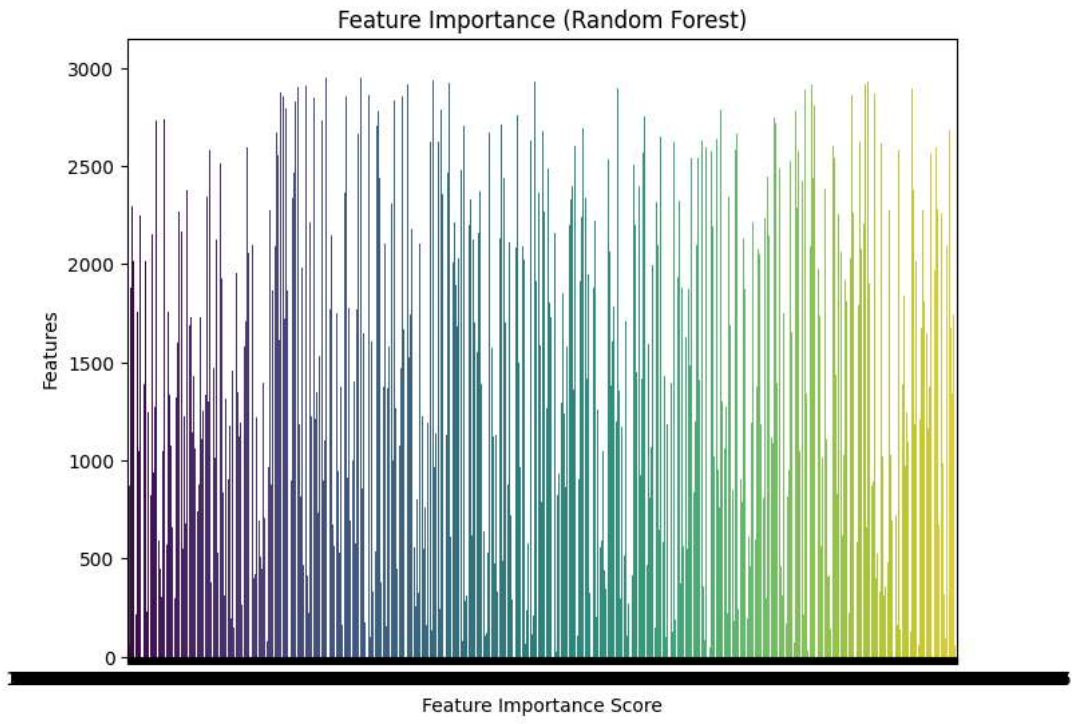


Feature Importance (Random Forest) 🌳

This visualization helps understand which features have the most impact on the model.

```
import numpy as np
import pandas as pd
import seaborn as sns
import joblib # Import joblib for loading models

# Load the saved Random Forest model
rf_model = joblib.load("rf_with_smote.pkl")  # Ensure the RF model is saved with this name

# Get feature importance
feature_importance = rf_model.feature_importances_

# Assuming you have feature names
# feature_names = X_test.columns  # Update if needed #This line is causing the error
# Create a range of numbers representing the feature indices
feature_names = np.arange(X_test.shape[1])

# Create DataFrame
feat_imp_df = pd.DataFrame({"Feature": feature_names, "Importance": feature_importance})
feat_imp_df = feat_imp_df.sort_values(by="Importance", ascending=False)

# Plot Feature Importance
plt.figure(figsize=(8, 6))
sns.barplot(x=feat_imp_df["Importance"], y=feat_imp_df["Feature"], palette="viridis")
plt.xlabel("Feature Importance Score")
plt.ylabel("Features")
plt.title("Feature Importance (Random Forest)")
plt.show()
```

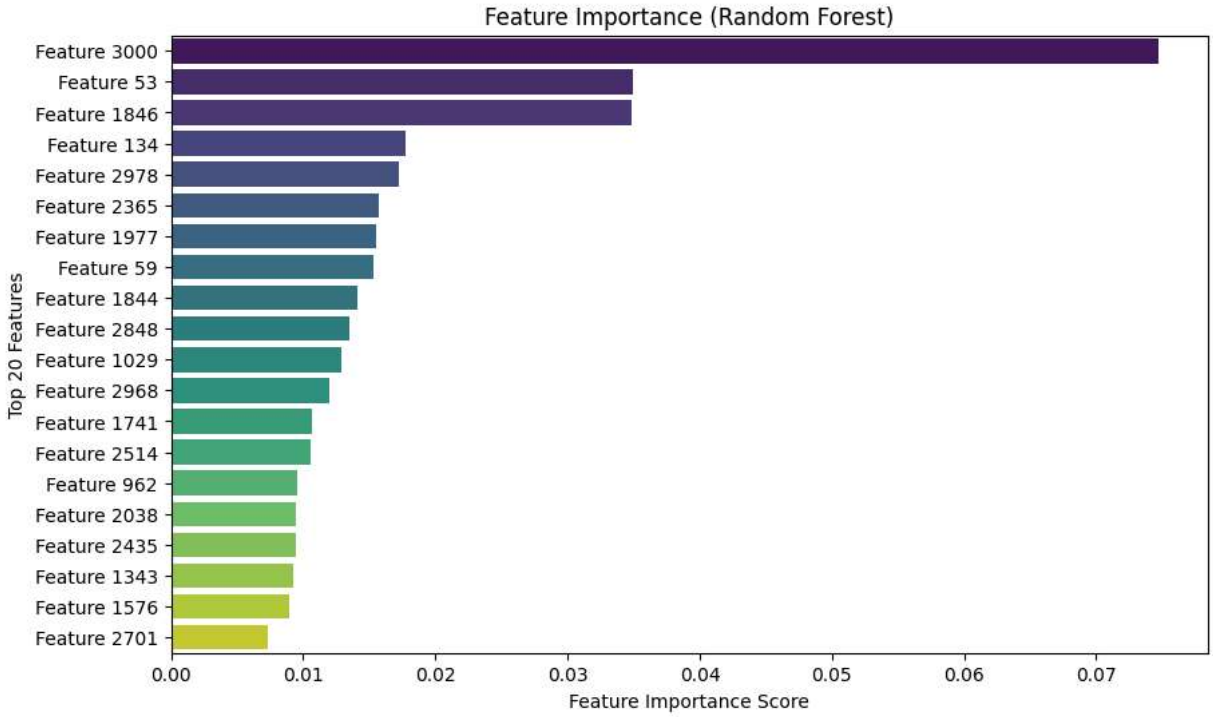
```
<ipython-input-14-7afd5a04ae9c>:23: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.barplot(x=feat_imp_df["Importance"], y=feat_imp_df["Feature"], palette="viridis")
```




```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import joblib

# Load the saved Random Forest model
rf_model = joblib.load("rf_with_smote.pkl")

# Load the test data
X_test = joblib.load("X_test.pkl")
```

```
# Convert X_test to dense format if sparse
if hasattr(X_test, "toarray"):
    X_test = X_test.toarray()

# Load feature names (Replace this with actual names if available)
feature_names = [f"Feature {i}" for i in range(X_test.shape[1])]

# Get feature importance
feature_importance = rf_model.feature_importances_

# Create DataFrame and sort by importance
feat_imp_df = pd.DataFrame({"Feature": feature_names, "Importance": feature_importance})
feat_imp_df = feat_imp_df.sort_values(by="Importance", ascending=False).head(20)  # Top 20 features

# Plot Feature Importance (Top 20)
plt.figure(figsize=(10, 6))
sns.barplot(x=feat_imp_df["Importance"], y=feat_imp_df["Feature"], palette="viridis")
plt.xlabel("Feature Importance Score")
plt.ylabel("Top 20 Features")
plt.title("Feature Importance (Random Forest)")
plt.show()
```

```
<ipython-input-16-6155e8ae500f>:29: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `le

  sns.barplot(x=feat_imp_df["Importance"], y=feat_imp_df["Feature"], palette="viridis")
```



Expiremention

HYBRID MODEL

We can combine the strengths of:

Random Forest (RF): Excellent at detecting normal instances (low False Positives). One-Class SVM (OC-SVM): High recall for attacks (low False Negatives). AdaBoost: Handles borderline cases better.

The hybrid pipeline consists of:

Preprocessing: Standardizing the data for SVM and other models. One-Class SVM: Trained only on normal data to detect anomalies. Ensemble Model: Random Forest and AdaBoost combined to refine the classification.

```
import numpy as np
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.svm import OneClassSVM
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, f1_score, recall_score
from sklearn.model_selection import KFold
```

```
from sklearn.pipeline import Pipeline
from sklearn.base import BaseEstimator, ClassifierMixin, clone

# Step 1: Custom Hybrid Model Class
class HybridModel(BaseEstimator, ClassifierMixin):
    def __init__(self, one_class_svm=None, random_forest=None, adaboost=None):
        self.one_class_svm = one_class_svm
        self.random_forest = random_forest
        self.adaboost = adaboost

    def fit(self, X, y):
        # Train One-Class SVM on normal instances (y == 0)
        self.one_class_svm_ = clone(self.one_class_svm)
        self.one_class_svm_.fit(X[y == 0])

        # Train Random Forest and AdaBoost on all data
        self.random_forest_ = clone(self.random_forest)
        self.random_forest_.fit(X, y)

        self.adaboost_ = clone(self.adaboost)
        self.adaboost_.fit(X, y)
        return self

    def predict(self, X):
        # Stage 1: One-Class SVM Prediction
        ocsvm_pred = self.one_class_svm_.predict(X)
        ocsvm_pred = (ocsvm_pred == 1).astype(int)  # Convert to binary (1 = normal, 0 = anomaly)

        # Stage 2: Predictions from Random Forest and AdaBoost
        rf_pred = self.random_forest_.predict(X)
        ab_pred = self.adaboost_.predict(X)

        # Combine predictions: Majority voting
        final_pred = (rf_pred + ab_pred + ocsvm_pred) >= 2  # Majority vote
        return final_pred.astype(int)

# Step 2: Pipeline Setup
scaler = StandardScaler()
one_class_svm = OneClassSVM(kernel='rbf', nu=0.1, gamma='scale')
random_forest = RandomForestClassifier(n_estimators=100, random_state=42)
adaboost = AdaBoostClassifier(n_estimators=50, random_state=42)

hybrid_model = HybridModel(one_class_svm=one_class_svm, random_forest=random_forest, adaboost=adaboost)

pipeline = Pipeline([
    ('scaler', scaler),
    ('hybrid', hybrid_model)
])

# Step 3: 10-Fold Cross-Validation
kf = KFold(n_splits=10, shuffle=True, random_state=42)
accuracy_scores = []
f1_scores = []
recall_scores = []

for train_index, val_index in kf.split(X_train):
    # Split the data
```