ECE 57000 Project

Step 1: Preprocess the Dataset

Task 1. Import the following libraries

In [78]:
```
# Import libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import SGDClassifier
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve,
import seaborn as sns
import matplotlib.pyplot as plt
import textwrap
```

Task 2. Load and Split Dataset

In [79]:
```
# Create Data class that loads and splits data
class Data:
  def __init__(self):
    self.df = pd.read_csv('sample_emails.csv')
    self.random_df = pd.read_csv('50_random_emails.csv')

  # Sample email data for training and testing
  def split_sample_data(self):
    X_train, X_test, y_train, y_test = train_test_split(self.df['text'],
    self.X_train = X_train
    self.X_test = X_test
    self.y_train = y_train
    self.y_test = y_test

  # 50 Random Purdue email data for testing
  def split_random_data(self):
    self.X_random = self.random_df['text']
    self.y_random = self.random_df['spam']

# Initialize Data class
dataset = Data()
dataset.split_sample_data()
dataset.split_random_data()
```

Task 3.View Dataset Head

In [80]:
```
# View dataset head
dataset.df.head()
```

Out[80]:

	text	spam
0	15.0.0.0\n\n2024-04-09T13:14:53\n\nSome people...	0
1	15.0.0.0\n\n2022-10-28T05:30:40\n\nWhen replyi...	0
2	15.0.0.0\n\n2022-11-11T04:07:03\n\n---- \n\n: ...	0
3	2024-12-02T23:43:34\n\n---- \n\n: Use caution ...	0
4	2025-02-19T02:11:15\n\n,\nPKA\n\n DX\n\n '25\n...	0

In [81]:
```
# View dataset information
dataset.df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10629 entries, 0 to 10628
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   text    10629 non-null  object
 1   spam    10629 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 166.2+ KB
```

In [82]:
```
# View shape of dataset
print(f"Shape of Sample Dataset: [{dataset.df.shape[0]}, {dataset.df.shap
print(f"Shape of Random Dataset: [{dataset.random_df.shape[0]}, {dataset.
```

```
Shape of Sample Dataset: [10629, 2]
Shape of Random Dataset: [50, 2]
```

Step 2: Train SVM Model Based on SGD

Implement a function that trains the Support Vector Machine (SVM) using the Stochastic Gradient Descent (SGD) optimizer, since SGDClassifier with hinge loss is equivlanet to the SVM.

$$objective\ function = \min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\max(0,\ 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b))$$

To iteratively update the model over a specified number of epochs, max_iter was set to 1 and warm_start was set to True, which enable controling the training process using partial_fit() while preserving the model's learned parameters between iterations. Additionally, the function prints the hinge loss between epochs using below formula.

$$L(y, f(x)) = \max(0, 1 - y \cdot f(x))$$

$$f(x_i) = \mathbf{w}^\top \mathbf{x}_i + b$$

In [83]:
```
def calculate_hinge_loss(y_true, y_pred):
  y_replace_0 = y_true.replace(0, -1) # replace 0 to -1 (class of ham ema
  product = y_replace_0 * y_pred
```

```
  hinge_losses = np.maximum(0, (1 - product))
  hinge_losses = np.mean(hinge_losses)
  return hinge_losses

def train_svm_model(dataset, epochs):
  # Vectorize X data for training
  vectorizer = CountVectorizer(stop_words='english') # remove English sto
  X_train_vector = vectorizer.fit_transform(dataset.X_train)

  # Initialize SGD-based SVM model
  sgd = SGDClassifier(loss='hinge', max_iter=1, warm_start=True, random_s
  classes = [0, 1] # 0: ham, 1: spam

  # Repeat training and print hinge loss
  for epoch in range(epochs):
      sgd.partial_fit(X_train_vector, dataset.y_train, classes=classes)

      # Calculate hinge loss
      y_pred = sgd.decision_function(X_train_vector)
      hinge_losses = calculate_hinge_loss(dataset.y_train, y_pred)

      print(f"Epoch {epoch+1}/{epochs}, Hinge Loss: {hinge_losses:.4f}")

  return sgd, vectorizer

# Initialize and train the model using n epochs
n = 10
sgd, vectorizer = train_svm_model(dataset, n)
```

```
Epoch 1/10, Hinge Loss: 0.1967
Epoch 2/10, Hinge Loss: 0.0147
Epoch 3/10, Hinge Loss: 0.0017
Epoch 4/10, Hinge Loss: 0.0001
Epoch 5/10, Hinge Loss: 0.0001
Epoch 6/10, Hinge Loss: 0.0000
Epoch 7/10, Hinge Loss: 0.0001
Epoch 8/10, Hinge Loss: 0.0000
Epoch 9/10, Hinge Loss: 0.0007
Epoch 10/10, Hinge Loss: 0.0001
```

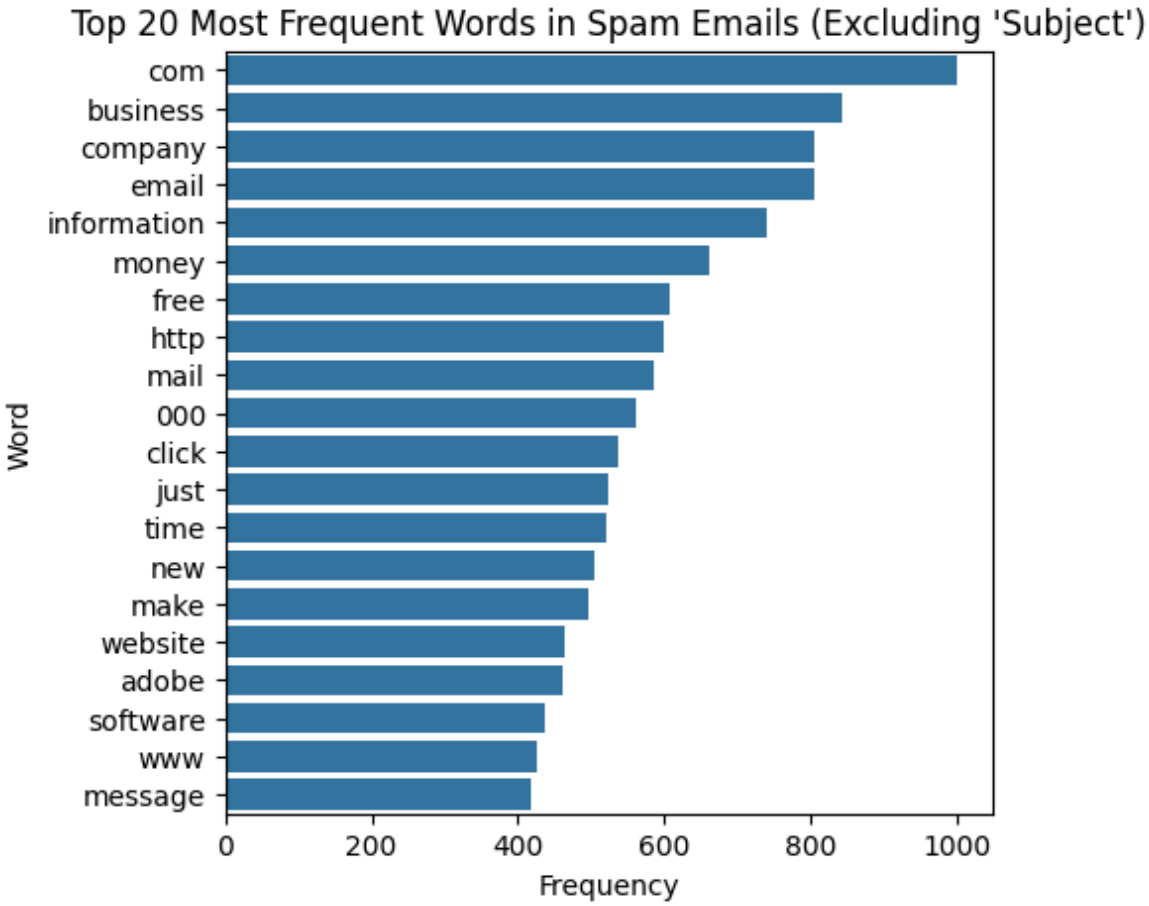
Step 3. Evaluate the SVM Model

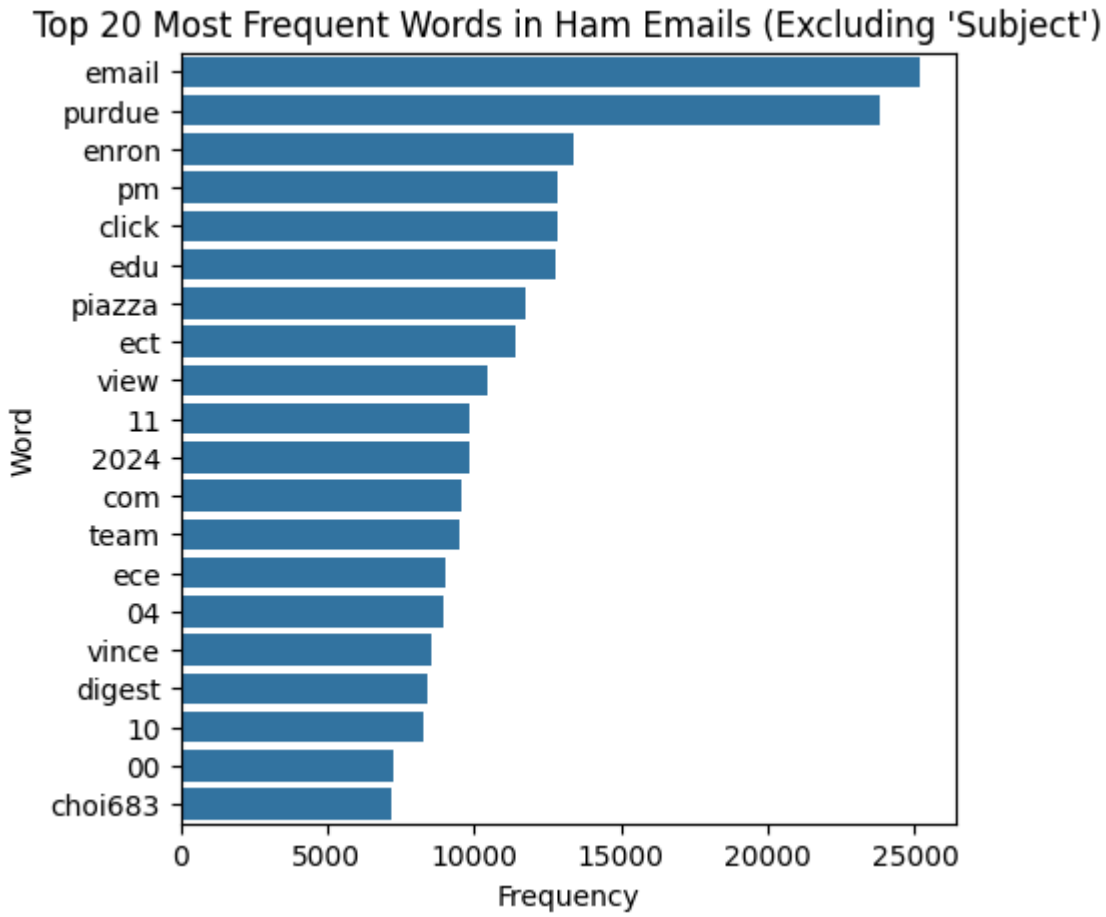
Task 1. Plot the Most Frequent Spam/Ham Words

In [84]:

```
def plot_frequent_words(dataset, n, email_type):
    # Remove word "Subject" (All texts contain it)
    texts = dataset.df[dataset.df['spam'] == email_type]['text']
    texts = texts.str.replace(r"(?i)subject:\s*", "", regex=True)

    # Vectorize texts
    vectorizer = CountVectorizer(stop_words='english')
    text_vector = vectorizer.fit_transform(texts)

    # Count the words
    counts = text_vector.sum(axis=0).A1
    words = vectorizer.get_feature_names_out()
```

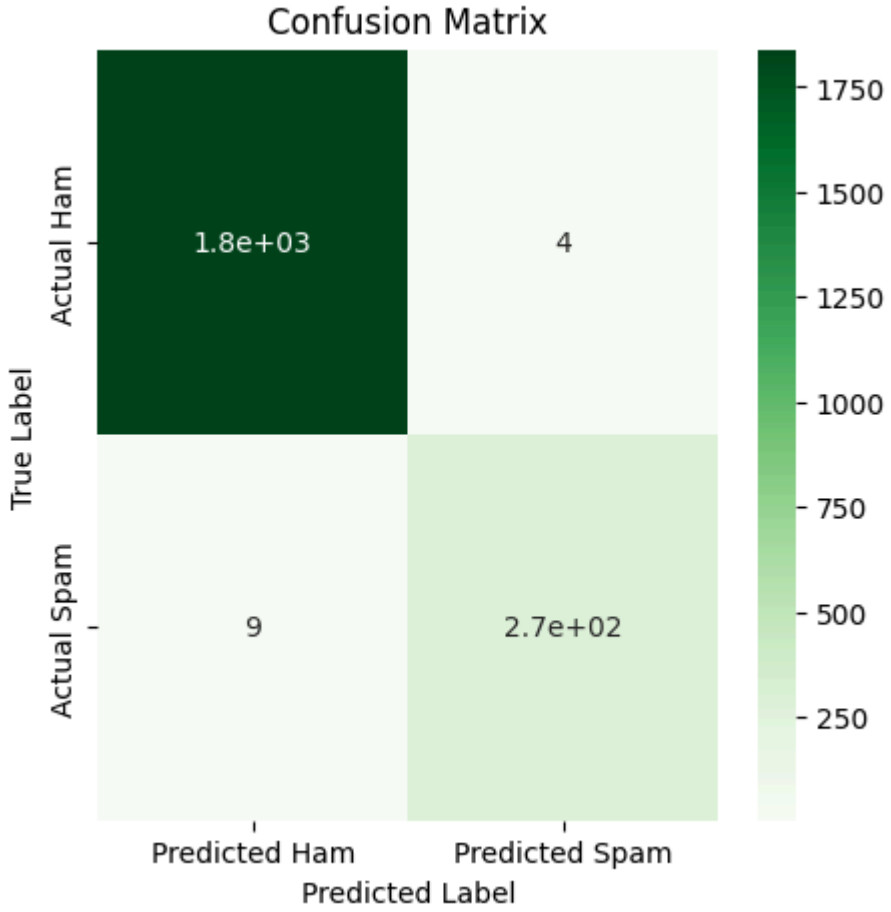
```
    # Create table in descending order
    result = pd.Series(counts, index=words)
    result = result.sort_values(ascending=False).head(n)

    # Plot a bar graph
    plt.figure(figsize=(5, 5))
    sns.barplot(x=result.values, y=result.index)
    if email_type == 1:
      plt.title(f"Top {n} Most Frequent Words in Spam Emails (Excluding '
    else:
      plt.title(f"Top {n} Most Frequent Words in Ham Emails (Excluding 'S
    plt.xlabel("Frequency")
    plt.ylabel("Word")
    plt.show()

n = 20
# Visualize n most frequent spam words
plot_frequent_words(dataset, n, 1)

# Visualize n most frequent ham words
print("\n")
plot_frequent_words(dataset, n, 0)
```





Task 2. Plot Confusion Matrix

In [85]:
```
def plot_confusion_matrix(dataset, model, vectorizer):
  # Vectorize test data and predcit email type
  X_test_vector = vectorizer.transform(dataset.X_test)
  y_pred = model.predict(X_test_vector)

  # Create confusion matrix
  cm = confusion_matrix(dataset.y_test, y_pred)
  cm_label = pd.DataFrame(cm, index=['Actual Ham', 'Actual Spam'], column

  # Plot confusion matrix
  plt.figure(figsize=(5, 5))
  sns.heatmap(cm_label, annot=True, cmap='Greens')
  plt.title("Confusion Matrix")
  plt.ylabel("True Label")
  plt.xlabel("Predicted Label")
  plt.show()

  # Display accuracy score
  accuracy = accuracy_score(dataset.y_test, y_pred) * 100
  print(f"\nAccuracy Score: {accuracy:.2f}%")

plot_confusion_matrix(dataset, sgd, vectorizer)
```

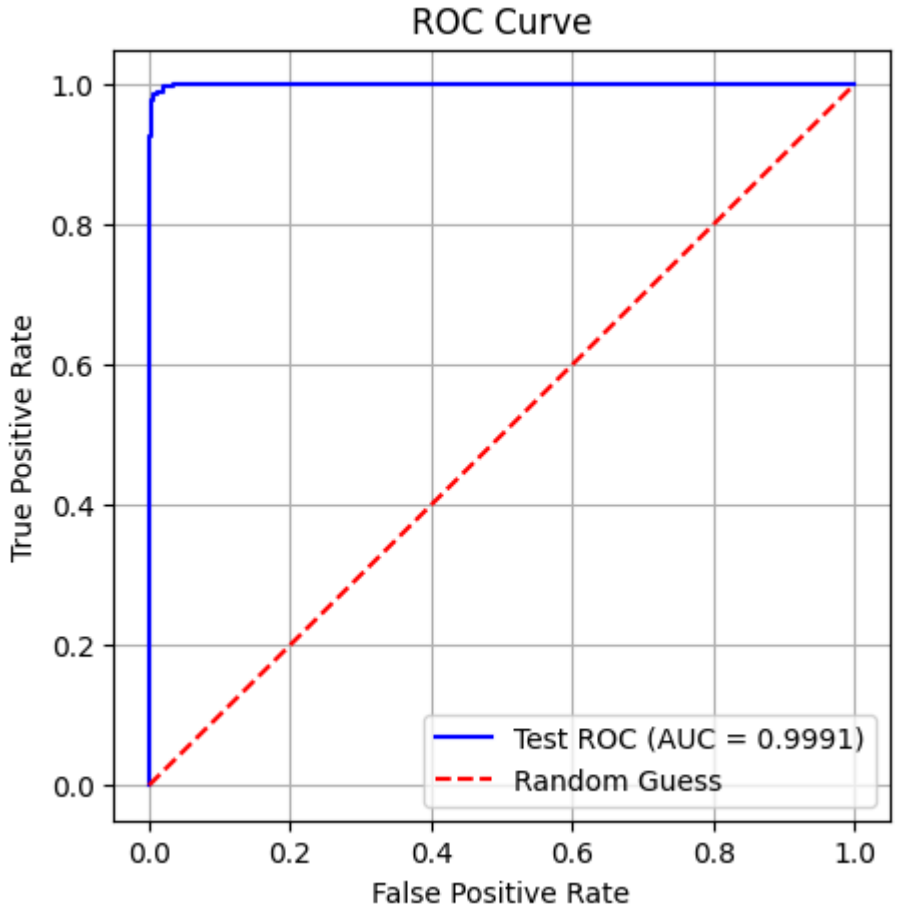



```
Accuracy Score: 99.39%
```

Task 3. ROC Curve

In [86]:
```
def plot_roc_curve(dataset, model, vectorizer):
  # Vectorize X data for test
  X_test_vector = vectorizer.transform(dataset.X_test)

  # Predict email type of vectorized X data
  y_decision_score = model.decision_function(X_test_vector)

  # Calculate ROC and AUC
  fprs_test, tprs_test, _ = roc_curve(dataset.y_test, y_decision_score)
  auc_test = auc(fprs_test, tprs_test)

  # Plot ROC curves
  plt.figure(figsize=(5, 5))
  plt.plot(fprs_test, tprs_test, label=f'Test ROC (AUC = {auc_test:.4f})'
  plt.plot([0, 1], [0, 1], 'k--', label='Random Guess', color='red')
  plt.xlabel("False Positive Rate")
  plt.ylabel("True Positive Rate")
  plt.title("ROC Curve")
  plt.legend(loc="lower right")
  plt.grid(True)
  plt.show()

plot_roc_curve(dataset, sgd, vectorizer)
```

```
<ipython-input-86-2769b9c52fc4>:15: UserWarning: color is redundantly defi
ned by the 'color' keyword argument and the fmt string "k--" (-> color
='k'). The keyword argument will take precedence.
  plt.plot([0, 1], [0, 1], 'k--', label='Random Guess', color='red')
```

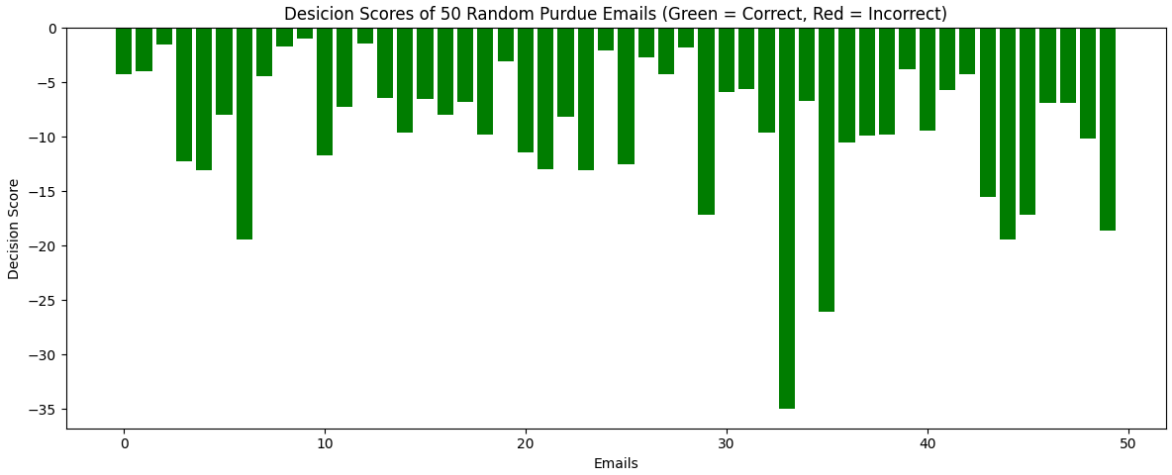


Task 4. Test Model with 50 Random Purdue Emails by Plotting Model Confidence Scores with Correct / Incorrect Colored-Predictions

A visualization of all predictions of 50 random purdue emails with decision scores, highlighting correct and incorrect classifications with color.

In [87]:
```
def test_random_purdue_emails(dataset, model, vectorizer):
  # Vectorize and predict random purdue email data
  X_test_vector = vectorizer.transform(dataset.X_random)
  y_decision_score = model.decision_function(X_test_vector)
  predicted = model.predict(X_test_vector)

  # Create table containing results of each email
  results = pd.DataFrame({'score': y_decision_score, 'correct': predicted

  # Plot a graph
  colors = results['correct'].map({True: 'green', False: 'red'})
  plt.figure(figsize=(12, 5))
  plt.bar(range(len(results)), results['score'], color=colors)
  plt.title("Desicion Scores of 50 Random Purdue Emails (Green = Correct,
  plt.xlabel("Emails")
  plt.ylabel("Decision Score")
  plt.tight_layout()
```

```
  plt.show()

test_random_purdue_emails(dataset, sgd, vectorizer)
```



Step 4: Classify Actual Emails Sent to Purdue Student Account

Implement a function that classifies actual emails sent to Purdue student account (Outlook) whether they are ham or spam and prints the probability of the prediction correctness.

In [92]:
```
def display_email(email_text, email_type, classification_result):
    text = "\n".join(textwrap.wrap(email_text, width=55))

    if (email_type == 1):
      text = "<Acutal Purdue Spam Email>\n\n" + text
    else:
      text = "<Acutal Purdue Ham Email>\n\n" + text

    text = text + "\n\n-> " + classification_result

    plt.figure(figsize=(5, 1))
    plt.axis('off')
    plt.text(0.05, 0.5, text, fontsize=12)
    plt.show()

def sigmoid(decision_score):
  return 1 / (1 + np.exp(-decision_score))

def classify_email(email_text, email_type, model, vectorizer):
  # Vectorize and predict type of received email
  text_vector = vectorizer.transform([email_text])
  decision_score = model.decision_function(text_vector)[0]
  prediction = model.predict(text_vector)[0]

  # Calculate correctness probability using sigmoid
  proba_spam = sigmoid(decision_score)
  proba_ham = 1 - proba_spam

  if (prediction == 0):
    label = "Ham"
```

```
    prob = proba_ham * 100
  else:
    label = "Spam"
    prob = proba_spam * 100

  classification_result = f"The email was classified as {label} with {pro
  display_email(email_text, email_type, classification_result)

# Test each spam and ham email sent to Purdue email account
with open("purdue_spam_email.txt", "r", encoding="utf-8") as f:
  purdue_spam_email = f.read()
  classify_email(purdue_spam_email, 1, sgd, vectorizer)
  print("\n")

with open("purdue_ham_email.txt", "r", encoding="utf-8") as f:
  purdue_ham_email = f.read()
  classify_email(purdue_ham_email, 0, sgd, vectorizer)
```

<Acutal Purdue Spam Email>

Greetings,The Federal Deposit Insurance Corporation (FDIC) has Approves Interim Final Rule with the Board of Education to assist/compensate each student/Teachers and Staffs due to the coronavirus disease 2019 (COVID-19) global pandemic with $1,980 .The following information is more important than ever during these challenging times.

-> The email was classified as Spam with 99.90% probability.

<Acutal Purdue Ham Email>

This email is to confirm that your submission to assignment folder PD Activity 3: Research Conference Presentation Submission was successful.

-> The email was classified as Ham with 98.99% probability.