Install required packages

```
!pip install sdv --quiet
!pip install openai --quiet
!pip install datasets --quiet
!pip install mimesis --quiet
!pip install nltk --quiet
!pip install wordcloud --quiet
!pip install mlflow --quiet
```

Show hidden output

NLTK download packages

```
import nltk
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

Import mimesis for fake data generation

```
from mimesis import Person
from mimesis import Generic
from mimesis import Fieldset
from mimesis.locales import Locale
from mimesis.enums import Gender
```

Import all the necessary library

```
import numpy as np
import openai
import os
import csv
import json
from openai import OpenAI
import pandas as pd
import random
import string
from datetime import datetime, timedelta
from datasets import load_dataset
import matplotlib.pyplot as plt
%matplotlib inline
from wordcloud import WordCloud, STOPWORDS
# Set a random seed for reproducibility
np.random.seed(42)
# Load  API key from an environment variable or secret management service
openai.api_key = ""
client = OpenAI(api_key="")
```

```
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifie
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from xgboost import XGBClassifier
```

```
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
```

Synthetic data Generation

Generate a fake person detail

```
def get_person_detail():
  """
    Generate random person details including full name, email, and occupation.

    Returns:
        Tuple[str, str, str]: A tuple containing the generated full name, email, and occupation.
  """
  # List of locales for generating person details
  locations = [Locale.CS,Locale.EN,Locale.DE,Locale.ES,Locale.FA,Locale.SV,Locale.FI,Locale.HU]
  # List of genders for generating person details
  genders = [Gender.FEMALE,Gender.MALE]

  # Randomly choose a location and gender
  location = random.choice(locations)
  gender = random.choice(genders)
  # Create a Generic instance with the chosen locatio
  data = Generic(location)
  return data.person.full_name(gender=gender),data.person.email(),data.person.occupation()
```

Download a huggingface spam/ham dataset for additional data

```
dataset = load_dataset("SetFit/enron_spam")
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/repocard.py:105: UserWarning: Repo card metadata block was not found. Setti
  warnings.warn("Repo card metadata block was not found. Setting CardData to empty.")
```

Get train,eval datasets

```
train_dataset = dataset['train']
eval_dataset = dataset['test']
```

Define suffix,company categories, prefix for phishing email generation

Show code

```
def get_string(input_data, probability=1):
    """
    Generates a random string from the given input data with a specified probability.

    Args:
        input_data (list): List of strings to choose from.
        probability (float, optional): Probability of generating a string from the input data. Defaults to 1.

    Returns:
        str or None: The randomly chosen string or None if the random float exceeds the probability.
    """
    # Generate a random float between 0 and 1
    random_float = random.random()

    # Equal weights for all input data elements
    weights = [1] * len(input_data)

    # Check if the random float is less than the specified probability
    if random_float < probability:
        # Choose a random string from the input data based on the weights
        data = random.choices(input_data, weights=weights)[0]
        return data

    return None
```

```
def get_phishing_suffix(probability=0.3):
    """
    Generates a random phishing suffix with a specified probability.

    Args:
        probability (float, optional): Probability of generating a phishing suffix. Defaults to 0.3.

    Returns:
        str or None: The randomly chosen phishing suffix or None if probability condition is not met.
    """
    # Call get_string function with phishing_suffixes as input data
    return get_string(phishing_suffixes, probability=probability)

def get_phishing_prefix(probability=1):
    """
    Generates a random phishing prefix.

    Args:
        probability (float, optional): Probability of generating a phishing prefix. Defaults to 1.

    Returns:
        str or None: The randomly chosen phishing prefix.
    """
    # Call get_string function with suspicious_prefixes as input data
    return get_string(suspicious_prefixes, probability=probability)
```

Methods for generating the phishing mail body and subject based on template

Show code

Method to generate a similar looking string(Phishing email ids have in common)

Show code

```
Input string: password
Similar string: password
```

Create a phishing mail-id,subject,body

```
def generate_phishing_mail_org():
  name,_,_ = get_person_detail()
  category_name = random.choices(list(company_categories.keys()), weights=category_weight)[0]
  company_name = get_string(company_categories[category_name],probability=1)
  company_name = company_name.lower().replace(' ', '-')
  phishing_name = generate_similar_string(company_name,0.3)
  phishing_prefix_org = get_phishing_prefix(probability=1)
  prefix2suffix = prefix_suffix_mapping[phishing_prefix_org]
  domain_suffix = get_string(domain_suffixes,probability=1)
  phishing_suffix_prob = 0.78
  phishing_prefix_prob = 0.6
  if not company_name == phishing_name:
    phishing_suffix_prob = 0.25
    phishing_prefix_prob = 0.18
  phishing_prefix = generate_similar_string(phishing_prefix_org,phishing_prefix_prob)
  phishing_suffix = get_string(prefix2suffix,probability=phishing_suffix_prob)
  if phishing_suffix:
    phishing_name = f'{phishing_name}-{phishing_suffix}'
  phishing_domain = f'{phishing_name}{domain_suffix}'
  phishing_mail = f'{phishing_prefix}@{phishing_domain}'
  phishing_subject = get_phishing_subject(phishing_prefix_org)
  phishing_body = get_phishing_body(phishing_prefix_org,name)
  return 'spam',phishing_body,phishing_subject,phishing_mail
```

Generate n number of fake random email

```
def generate_fake_email(n_samples=1):
    """
    Method to generate a list of fake emails based on n_sample
    """
```

Code block to get a sample of data from huggingface dataset

```
def get_hf_data(dataset, n_samples, class_name='spam'):
    """
    Generate a sample dataset with a specified number of samples and class name.

    Args:
        dataset (Dataset): Hugging Face dataset containing the original data.
        n_samples (int): Number of samples to generate.
        class_name (str, optional): Name of the class to filter. Defaults to 'spam'.

    Returns:
        pandas.DataFrame: A DataFrame containing the generated samples with the specified class label.
    """
    # Columns to remove from the dataset
    columns_to_remove = ["message_id", "text", "label", "date"]

    # Remove the specified columns from the dataset
    temp_dataset = dataset.remove_columns(columns_to_remove)

    # Filter the dataset to get inputs with the specific label
    filtered_dataset = temp_dataset.filter(lambda example, idx: example['label_text'] == class_name, with_indices=True)

    # Convert the filtered dataset to a pandas DataFrame
    filtered_df = filtered_dataset.to_pandas()

    # Reset the index of the DataFrame
    filtered_df = filtered_df.reset_index(drop=True)

    # Sample n_samples rows from the filtered DataFrame
    filtered_df = filtered_df.sample(n=n_samples)

    # Generate fake email addresses for the sampled rows
    e_mail = generate_fake_email(n_samples=n_samples)

    # Add the generated email addresses as a new column 'sender'
    filtered_df['sender'] = e_mail

    # Reset the index of the DataFrame
    return filtered_df.reset_index(drop=True)
```

Generate spam email data

```
def generate_spam_data(n=10):
    """
    Generate spam data by combining phishing emails from an organization and samples from a Hugging Face dataset.

    Args:
        n (int, optional): Number of samples to generate. Defaults to 10.

    Returns:
        pandas.DataFrame: DataFrame containing the generated spam data.
    """
    # Calculate the number of samples needed from each source
    samples = n // 2

    # Generate phishing emails from an organization
    org_phishing_data = [generate_phishing_mail_org() for idx in range(samples)]

    # Convert the phishing data to a DataFrame
    org_phishing_df = pd.DataFrame(org_phishing_data)

    # Calculate the remaining number of samples needed
    remaining_mail = n - samples

    # Get phishing data from a Hugging Face dataset
    hf_phishing_data = get_hf_data(train_dataset, remaining_mail, class_name="spam")

    # Concatenate the two DataFrames to create the final dataset
    result_df = pd.concat([hf_phishing_data, org_phishing_df], ignore_index=True)

    return result_df
```

Generate synthetic data using LLM (openai api)

```
def generate_data_open_ai(n=500,file_name="ham_data.json"):
  response = client.chat.completions.create(
  model="gpt-3.5-turbo-0125",
  # model="gpt-4-0125-preview",
  response_format={ "type": "json_object" },
  messages=[
    {"role": "system", "content": "You are a helpful assistant designed to output JSON."},
    {"role": "user", "content": f" write me {n} complete  email like business email,job offers, linkedin, Finance, social media, mail b
  ]
  )
  data = json.loads(response.choices[0].message.content)
  with open(file_name,'w') as json_file:
    json.dump(data,json_file)
  ham_df = pd.DataFrame(data['emails'])
  ham_df = ham_df[['label', 'subject', 'message','sender']]
  ham_df = ham_df.rename(columns={'label': 'label_text'})
  return ham_df
```

Generate LLM data in samples of 20 due to max token limitation in open ai api

```
def get_llm_data(samples):
  final_ham_df = pd.DataFrame()
  for i in range(0,samples,20):
    ham_df = generate_data_open_ai(5,file_name=f"data_{str(i)}.json")
    ham_df.to_csv(f'ham_data_{str(i)}.csv', index=False)
    final_ham_df = pd.concat([final_ham_df,ham_df],ignore_index=True)
  return final_ham_df
```

Method to generate legitimate email using synthetic data + hugging face dataset

```
def generate_ham_data(n=10):
  samples = 1000
  try:
    org_ham_df = get_llm_data(samples)
    remaining_mail = n - samples
  except:
    org_ham_df = pd.DataFrame()
    remaining_mail = n
  hf_ham_data = get_hf_data(train_dataset,remaining_mail,class_name="ham")
  result_df = pd.concat([hf_ham_data, org_ham_df], ignore_index=True)
  return result_df
```

Generate SPAM, Legitimate emails and store in a csv file

```
spam_df = generate_spam_data(n=6000)
# Save the DataFrame as a CSV file
spam_df.to_csv('spam_data.csv', index=False)
ham_df = generate_ham_data(n=6500)
# Save the DataFrame as a CSV file
ham_df.to_csv('ham_data.csv', index=False)
# Concatenate the DataFrames
train_df = pd.concat([spam_df, ham_df], ignore_index=True)
# Save the DataFrame as a CSV file
train_df.to_csv('train_synthetic_data.csv', index=False)
```

Fix the row indexing issue in dataframe contatination

```
def reorder_row(row):
    return row[row.notnull()].tolist() + row[row.isnull()].tolist()
updated_train_df = train_df.sample(frac=1, replace=False, random_state=42)
updated_train_df = updated_train_df.apply(reorder_row, axis=1, result_type='expand')
updated_train_df.columns = [str(col) for col in train_df.columns]
updated_train_df = updated_train_df.drop(columns=['0','1','2','3'])
```

View the sample data

```
updated_train_df.head()
```

	label_text	subject	message	sender
6930	ham	rt / volume management meeting	the mt . hood conference room is reserved for ...	usb1887@protonmail.com
8384	ham	re : 7 / 14 - - crude oil and nat gas	fyi\n1 . george hopefully will get us some pea...	road1973@example.com
7889	ham	re : hello team	ken :\nwe are very excited about our alp at en...	cgi1872@example.com
357	spam	' the antidote '	hello ,\nwe have an alternative to drugs & ant...	fought1918@live.com
3753	spam	Dear Ferdinand Kahanek,\n\nWe regret to inform...	Your payment status: Verify your account detai...	payment@mićrosoft.com

```
updated_train_df.to_csv('formated_train_data.csv', index=False,quoting=csv.QUOTE_ALL)
```

```
updated_train_df.shape
```

```
(11938, 4)
```

Replace the data_path with path of formated_train_data.csv

```
data_path = "/content/drive/MyDrive/EMAIL_ML/"
```

```
from sdv.datasets.local import load_csvs
from sdv.datasets.demo import download_demo

# This is the default folder name that the GOogle Colab notebook uses.
# Change this if you have your own folder with CSV files.
FOLDER_NAME = data_path
# data = load_csvs(folder_name=FOLDER_NAME)
try:
  data = load_csvs(folder_name=FOLDER_NAME)
except ValueError:
  print('You have not uploaded any csv files. Using some demo data instead.')
```

```
email_table = data['formated_train_data']

# use the head method to inspect the first few rows of the data
email_table.tail(3)
```

	label_text	subject	message	sender
11935	spam	Dear مرداس مددی,\n\nWe recently received an en...	Urgent enquiry: Verify your account details to...	enquiry@naukři.io
11936	spam	do not walk behind me , for i may not lead . d...	you have 2 options here ,\noption 1 - you can ...	marks2075@example.org
11937	ham	enron / hpl actuals for january 2 , 2001	teco tap 30 . 000 / enron\nls hpl lsk ic 30	glow2010@example.org

Show code

Show code

```
Auto detected data:
```

label_text : categorical
subject : unknown
message : unknown
sender : unknown

Show code

Show code

Exploratory Data Analysis

Check columns for null values

```
email_table.isna().sum()
```

```
label_text     0
subject       41
message       67
sender         0
dtype: int64
```

Drop rows which do not contain the subject and email body

```
rows_to_remove = email_table[email_table['subject'].isna()&(email_table['message'].isna())].index
email_table.drop(rows_to_remove, inplace=True)
```

Analysis to check where subject is missing, where email body is missing

```
spam_na_subject = email_table[email_table['subject'].isna()& (email_table['label_text'] == 'spam')]
ham_na_subject = email_table[email_table['subject'].isna()& (email_table['label_text'] == 'ham')]
spam_na_message = email_table[email_table['message'].isna()& (email_table['label_text'] == 'spam')]
ham_na_message = email_table[email_table['message'].isna()& (email_table['label_text'] == 'ham')]
```

```
ham_na_message.head(3)
```

	label_text	subject	message	sender
1220	ham	fyi - piece on enron	NaN	pump1937@example.com
2647	ham	recommendations based on job group , mid - yea...	NaN	alumni1992@outlook.com
3136	ham	fyi	NaN	kills1979@yahoo.com

Fix the missing email body with their subject

```
email_table.loc[email_table['message'].isna(), 'message']= email_table.loc[email_table['message'].isna(), 'subject']
```

Fill the missing email subject with random phishing email subject

```
# Iterate through rows in na_rows_except_value
for index, row in spam_na_subject.iterrows():
    # Randomly choose one of the 5 values
    random_value = np.random.choice(['Urgent Action required!', 'You have 1 day left ', 'Security alert- account access denied'])
    # Set the randomly chosen value in the specific column for the current row
    email_table.at[index, 'subject'] = random_value
```

Verify there is no null columns in dataframe

```
email_table.isna().sum()
```

```
label_text    0
subject       0
message       0
sender        0
dtype: int64
```

```
email_table.head()
```

	label_text	subject	message	sender
0	ham	rt / volume management meeting	the mt . hood conference room is reserved for ...	usb1887@protonmail.com
1	ham	re : 7 / 14 - - crude oil and nat gas	fyi\n1 . george hopefully will get us some pea...	road1973@example.com
2	ham	re : hello team	ken :\nwe are very excited about our alp at en...	cgi1872@example.com
3	spam	' the antidote '	hello ,\nwe have an alternative to drugs & ant...	fought1918@live.com
4	spam	Dear Ferdinand Kahanek,\n\nWe regret to inform...	Your payment status: Verify your account detai...	payment@mićrosoft.com

Check if there is any duplication in synthetic data generation

```
email_table.duplicated().sum()
```

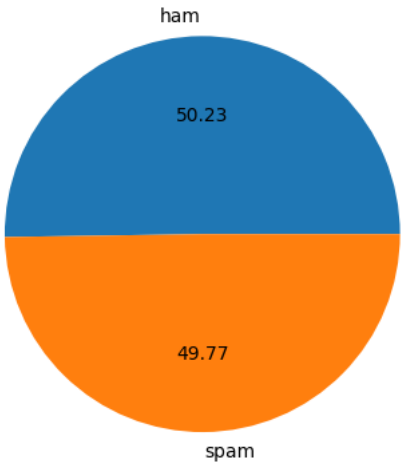
```
0
```

```
email_table['label_text'].value_counts()
```

```
spam    5994
ham     5938
Name: label_text, dtype: int64
```

Visualiztion of class distribution

```
plt.pie(email_table['label_text'].value_counts(), labels=['ham','spam'],autopct="%0.2f")
plt.show()
```




```
email_table.groupby('label_text').describe()
```

	subject				message				sender			
	count	unique	top	freq	count	unique	top	freq	count	unique	top	freq
label_text												
ham	5938	5151	schedule crawler : hourahead failure	64	5938	5846	---------- ...	4	5938	5757	finance@bank.com	16
			Subject: Urgent:				YOUR ACCOUNT					

Generate point cloud to visualize most occuring words in dataset

```
def plot_wordcloud(text, mask=None, max_words=200, max_font_size=100, figure_size=(24.0,16.0)
                   title = None, title_size=40, image_color=False):
    stopwords = set(STOPWORDS)
    wordcloud = WordCloud(background_color='black',
                    stopwords = stopwords,
                    max_words = max_words,
                    max_font_size = max_font_size,
                    random_state = 42,
```

```
                    width=800,
                    height=400,
                    mask = mask)
    wordcloud.generate(str(text))

    plt.figure(figsize=figure_size)
    plt.imshow(wordcloud)
    plt.title(title, fontdict={'size': title_size, 'color': 'black',
                               'verticalalignment': 'bottom'})
    plt.axis('off');

messages_text = ' '.join(email_table['message'].dropna())

plot_wordcloud(messages_text)
```


```
email_table['label_text'].unique()
```

```
array([0, 1])
```

Label encode the target columns with numeric values

```
email_table['label_text'] = email_table['label_text'].replace({'ham': 0, 'spam': 1})
```

Create a new column in Dataframe contatnating both subject and body

```
email_table['input_no_sender'] = email_table['subject'] + ' ' + email_table['message']
```

Create a new column in Dataframe contatnating sender mail, subject and body

```
email_table['input_with_sender'] = email_table['sender'] + ' ' + email_table['subject'] + ' ' + email_table['message']

from nltk.tokenize import word_tokenize
import re
```

Perprocess raw text to remove sensitive data, stop words, convert words to their root form

```
def preprocess_text(text):
    # Convert to lowercase
    text = text.lower()

    # Remove URLs
    text = re.sub(r'http\S+', '', text)

    # Remove punctuation and digits
    text = re.sub(r'[^a-zA-Z\s]', '', text)

    # Tokenization
    tokens = word_tokenize(text)

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words]

    # Lemmatization
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(word) for word in tokens]

    # Join tokens back into text
    preprocessed_text = ' '.join(tokens)

    return preprocessed_text
```

Apply the pre-processing transformation on input columns

```
# Preprocess text data
email_table['input_no_sender'] = email_table['input_no_sender'].apply(preprocess_text)
email_table['input_with_sender'] = email_table['input_with_sender'].apply(preprocess_text)
```

```
backup_email_table = email_table.copy()
```

Store the pre-processed input features in a csv file

```
email_table.to_csv('preprocessed_email_data.csv', index=False,quoting=csv.QUOTE_ALL)
```

```
email_table.head(1)
```

	label_text	subject	message	sender	input_no_sender	input_with_sender
0	0	rt / volume management meeting	the mt . hood conference room is reserved for ...	usb1887@protonmail.com	rt volume management meeting mt hood conferenc...	usbprotonmailcom rt volume management meeting ...

```
# Extract features and labels
X = email_table['input_no_sender']
y = email_table['label_text']
# Split the dataset in a stratified way
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2, random_state=42)

# Converting text to TF-IDF features
tfidf_vectorizer = TfidfVectorizer(max_features=1000)
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)
```

```
print(y_train.value_counts())
print('Test/Val split',y_test.value_counts())
```

```
1    4795
0    4750
Name: label_text, dtype: int64
Test/Val split 1    1199
0    1188
Name: label_text, dtype: int64
```

References for synthetic data creation, domain understanding, dataset samples

https://www.linkedin.com/pulse/text-classification-exploration-spam-detection-news-varghese-chacko/

https://www.hindawi.com/journals/scn/2022/1862888/

https://archive.ics.uci.edu/dataset/228/sms+spam+collection

https://www.getcybersafe.gc.ca/en/resources/real-examples-fake-emails