```
import pandas as pd

data = pd.read_csv("enron_spam_data.csv", engine="python", on_bad_lines="skip")
data = data[['Message', 'Spam/Ham']]
data.columns = ['text', 'label']
data.dropna(inplace=True)
```

```
def remap_label(row):
    text = row['text']
    if row['label'] == 'spam':
        return 'Spam'
    if 'offer' in text or 'discount' in text or 'sale' in text:
        return 'Promotions'
    if 'support' in text or 'issue' in text or 'help' in text:
        return 'Support'
    return 'Personal'

data['label'] = data.apply(remap_label, axis=1)
```

```
import re

def clean_text(text):
    text = text.lower()
    text = re.sub(r'[^a-zA-Z ]', '', text)
    return text

data['text'] = data['text'].apply(clean_text)
```

```
data['text'] = data['text'].astype(str)
data = data[data['text'].str.strip() != '']
data.dropna(inplace=True)
```

```
X = data['text']
y = data['label']

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
)
```

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(max_features=5000, ngram_range=(1,2))
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)
```

```
from sklearn.linear_model import LogisticRegression

tfidf_model = LogisticRegression(max_iter=1000)
tfidf_model.fit(X_train_tfidf, y_train)
```

```
LogisticRegression
LogisticRegression(max_iter=1000)
```

```
from sklearn.metrics import confusion_matrix, classification_report

y_pred_tfidf = tfidf_model.predict(X_test_tfidf)
confusion_matrix(y_test, y_pred_tfidf)
classification_report(y_test, y_pred_tfidf)
```

```
'              precision    recall  f1-score   support\n\n    Personal       0.86      0.99      0.92      1175\n  Promotions
0.93      0.54      0.68       236\n        Spam       1.00      1.00      1.00       596\n     Support       0.85      0.65
0.74       339\n\n    accuracy                           0.90      2346\n   macro avg       0.91      0.79      0.83      2346
```

```
pip install fasttext
```

```
Collecting fasttext
  Downloading fasttext-0.9.3.tar.gz (73 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 73.4/73.4 kB 1.6 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Collecting pybind11>=2.2 (from fasttext)
  Using cached pybind11-3.0.1-py3-none-any.whl.metadata (10.0 kB)
Requirement already satisfied: setuptools>=0.7.0 in /usr/local/lib/python3.12/dist-packages (from fasttext) (75.2.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (from fasttext) (2.0.2)
Using cached pybind11-3.0.1-py3-none-any.whl (293 kB)
Building wheels for collected packages: fasttext
  Building wheel for fasttext (pyproject.toml) ... done
  Created wheel for fasttext: filename=fasttext-0.9.3-cp312-cp312-linux_x86_64.whl size=4498211 sha256=e0cb53dc3ccd1c1e79ee81ec4
  Stored in directory: /root/.cache/pip/wheels/20/27/95/a7baf1b435f1cbde017cabdf1e9688526d2b0e929255a359c6
Successfully built fasttext
Installing collected packages: pybind11, fasttext
Successfully installed fasttext-0.9.3 pybind11-3.0.1
```

```
import fasttext

with open("emails.txt", "w") as f:
    for t in X:
        f.write(t + "\n")

ft_model = fasttext.train_unsupervised("emails.txt", model="skipgram")
```

```
import numpy as np

def ft_vector(text):
    return ft_model.get_sentence_vector(text)

X_train_ft = np.array([ft_vector(t) for t in X_train])
X_test_ft = np.array([ft_vector(t) for t in X_test])
```

```
from sklearn.ensemble import RandomForestClassifier

ft_model_clf = RandomForestClassifier(n_estimators=200)
ft_model_clf.fit(X_train_ft, y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier(n_estimators=200)
```

```
y_pred_ft = ft_model_clf.predict(X_test_ft)
confusion_matrix(y_test, y_pred_ft)
classification_report(y_test, y_pred_ft)
```

```
'              precision    recall  f1-score   support\n\n    Personal       0.78      0.98      0.87      1175\n  Promotions
0.84      0.39      0.53       236\n        Spam       1.00      1.00      1.00       596\n     Support       0.78      0.37
```

```
from transformers import AutoTokenizer, AutoModel
import torch

tokenizer = AutoTokenizer.from_pretrained("sentence-transformers/all-MiniLM-L6-v2")
model = AutoModel.from_pretrained("sentence-transformers/all-MiniLM-L6-v2")
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
config.json: 100%                                                 612/612 [00:00<00:00, 54.6kB/s]

tokenizer_config.json: 100%                                       350/350 [00:00<00:00, 33.1kB/s]

vocab.txt:  232k/? [00:00<00:00, 8.26MB/s]

tokenizer.json:  466k/? [00:00<00:00, 18.3MB/s]

special_tokens_map.json: 100%                                     112/112 [00:00<00:00, 9.85kB/s]

model.safetensors: 100%                                           90.9M/90.9M [00:01<00:00, 110MB/s]

Loading weights: 100%                                             103/103 [00:00<00:00, 373.33it/s, Materializing param=pooler.dense.weight]
BertModel LOAD REPORT from: sentence-transformers/all-MiniLM-L6-v2
Key                     | Status     |  |
------------------------+------------+--+-
embeddings.position_ids | UNEXPECTED |  |

Notes:
- UNEXPECTED    :can be ignored when loading from different task/architecture; not ok if you expect identical arch.
```

```
def transformer_embedding(text):
    inputs = tokenizer(text, return_tensors="pt", truncation=True, padding=True)
    with torch.no_grad():
        output = model(**inputs)
    return output.last_hidden_state.mean(dim=1).squeeze().numpy()
```

```
X_train_emb = np.array([transformer_embedding(t) for t in X_train])
X_test_emb = np.array([transformer_embedding(t) for t in X_test])
```

```
genai_model = LogisticRegression(max_iter=1000)
genai_model.fit(X_train_emb, y_train)
```

```
▾ LogisticRegression
LogisticRegression(max_iter=1000)
```

```
y_pred_genai = genai_model.predict(X_test_emb)
confusion_matrix(y_test, y_pred_genai)
classification_report(y_test, y_pred_genai)
```

```
'              precision    recall  f1-score   support\n\n    Personal       0.81      0.90      0.85      1175\n  Promotions
0.70      0.52      0.59       236\n        Spam       1.00      1.00      1.00       596\n     Support       0.58      0.47
0.52       339\n\n    accuracy                           0.82      2346\n   macro avg       0.77      0.72      0.74      2346
```

```
def predict_email(email):
    email = clean_text(email)
    emb = transformer_embedding(email).reshape(1, -1)
    return genai_model.predict(emb)[0]

predict_email("Get 50 percent discount now")
```

```
'Promotions'
```

```
Start coding or generate with AI.
```