```
!pip install --upgrade transformers
#Libraries
import numpy as np
import pandas as pd
import re

# Visualization tools
import matplotlib.pyplot as plt
import seaborn as sns

# Word Cloud library
from wordcloud import WordCloud, STOPWORDS

# Library used for data preprocessing
from datasets import load_dataset, DatasetDict, Dataset, ClassLabel
from transformers import AutoTokenizer

# Model selection libraries
from sklearn.model_selection import train_test_split

# Library used for ignore warnings
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.12/dist-packages (4.57.0)
Collecting transformers
  Downloading transformers-4.57.1-py3-none-any.whl.metadata (43 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 44.0/44.0 kB 2.9 MB/s eta 0:00:00
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from transformers) (3.20.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.34.0 in /usr/local/lib/python3.12/dist-packages (from transformers) (0.35.3)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.12/dist-packages (from transformers) (2.0.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from transformers) (25.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-packages (from transformers) (6.0.3)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.12/dist-packages (from transformers) (2024.11.6)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from transformers) (2.32.4)
Requirement already satisfied: tokenizers<=0.23.0,>=0.22.0 in /usr/local/lib/python3.12/dist-packages (from transformers) (0.22.1)
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.12/dist-packages (from transformers) (0.6.2)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.12/dist-packages (from transformers) (4.67.1)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub<1.0,>=0.34.0->trans
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub<1.0,>=0.3
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub<1.0,>=0.34.0->t
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests->transformers) (3.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests->transformers) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests->transformers) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests->transformers) (2025.10.
Downloading transformers-4.57.1-py3-none-any.whl (12.0 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.0/12.0 MB 52.1 MB/s eta 0:00:00
Installing collected packages: transformers
  Attempting uninstall: transformers
    Found existing installation: transformers 4.57.0
    Uninstalling transformers-4.57.0:
      Successfully uninstalled transformers-4.57.0
Successfully installed transformers-4.57.1
```

```
dataset = load_dataset("bourigue/data_email_spam")
print(dataset)
```

```
README.md: 100%  24.0/24.0 [00:00<00:00, 1.10kB/s]
spam_email_dataset.csv: 100%  112M/112M [00:02<00:00, 57.1MB/s]
Generating train split: 100%  85782/85782 [00:02<00:00, 33410.23 examples/s]
DatasetDict({
    train: Dataset({
        features: ['text_combined', 'label'],
        num_rows: 85782
    })
})
```

```
df = dataset['train'].to_pandas()
df.head()
```

	text_combined	label
0	fark rssfeedsspamassassintaintorg url httpwwwn...	0
1	alamac thanks painful got contract says 5 busi...	0
2	daily top 10 tanneryteunion104tnet daily top 1...	1
3	liza everett linpillardmetpillardde po dh ren ...	1
4	czanikhotmailcom font face3dverdana size3d3 co...	1

Next steps: Generate code with df | New interactive sheet

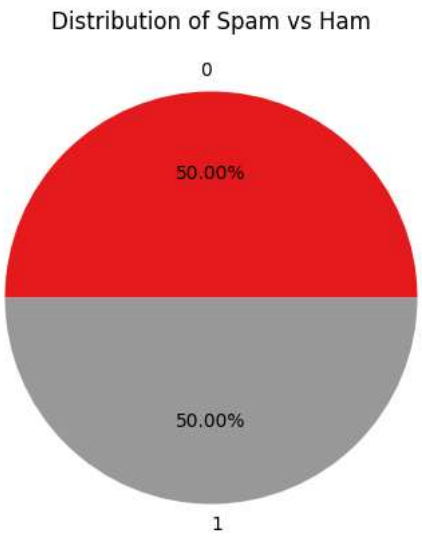
```
print(f"Dataset loaded: {len(df)} samples")
print(f"Class distribution:\n{df['label'].value_counts()}\n")
```

```
Dataset loaded: 85782 samples
Class distribution:
label
0    42891
1    42891
Name: count, dtype: int64
```

```
import matplotlib.pyplot as plt

# Label distribution
spread = df['label'].value_counts()

# Pie chart
plt.rcParams['figure.figsize'] = (5, 5)
spread.plot(kind='pie', autopct='%1.2f%%', cmap='Set1')
plt.title('Distribution of Spam vs Ham')
plt.ylabel('')
plt.show()
```

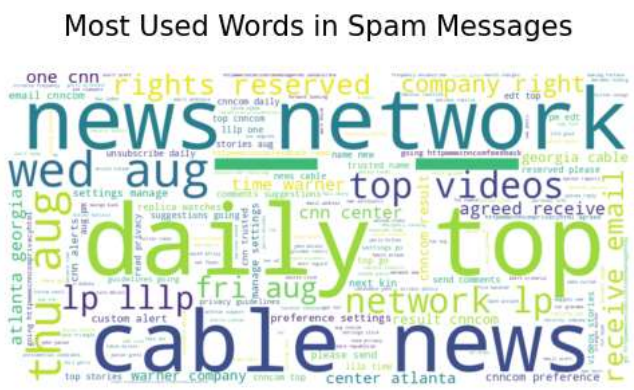



```
df_spam = df[df['label'] == 1]
stopwords = set(STOPWORDS)
text = " ".join(df_spam['text_combined'].astype(str).tolist()).lower()

wordcloud = WordCloud(
    width=1000,
    height=500,
    background_color='white',
    stopwords=stopwords,
    max_words=1000
).generate(text)

plt.figure(figsize=(6, 6))
plt.title("Most Used Words in Spam Messages", fontsize=15, pad=20)
plt.imshow(wordcloud, interpolation='bilinear')
```

```
plt.axis('off')
plt.show()
```




```
CONFIG = {
    'model_name': 'distilbert-base-uncased',
    'max_length': 128,
    'train_split': 0.7,
    'val_split': 0.1,
    'test_split': 0.2,
    'seed': 42
}

# Normalization:

def normalize_text(text):
    #Normalizing and cleaning email text while preserving spam indicators

    if pd.isna(text):
        return ""

    text = str(text)

    #Lowercase
    text = text.lower()

    #Normalize repeated characters (such as "oooo " or "eeeee")
    text = re.sub(r'(.)\1{2,}', r'\1\1', text)

    #Normalize numbers and replace with NUMBER token
    text = re.sub(r'\d+', ' NUMBER ', text)

    #Normalize currency symbols
    text = re.sub(r'[$£€¥₹]', ' CURRENCY ', text)

    #Replace URLs with URL token
    text = re.sub(r'http\S+|www\.\S+', ' URL ', text)

    #Replace email addresses with EMAIL token
    text = re.sub(r'\S+@\S+', ' EMAIL ', text)

    #Normalize excessive punctuation
    text = re.sub(r'!{2,}', ' EXCLAMATION ', text)
    text = re.sub(r'\?{2,}', ' QUESTION ', text)

    #Remove HTML tags
    text = re.sub(r'<[^>]+>', '', text)

    #Remove special characters
    text = re.sub(r'[^\w\s!?$]', ' ', text)

    #Normalize whitespace
    text = re.sub(r'\s+', ' ', text).strip()

    #Remove very short texts
    if len(text) < 10:
        return ""

    return text

df['text_combined'] = df['text_combined'].apply(normalize_text)
```

```
# Remove empty texts after normalization
df = df[df['text_combined'].str.len() > 0].reset_index(drop=True)
print(f"Normalization complete: {len(df)} samples remaining")
```

```
Normalization complete: 85777 samples remaining
```

```
#Test set:
train_val_df, test_df = train_test_split(
    df,
    test_size=CONFIG['test_split'],
    random_state=CONFIG['seed'],
    stratify=df['label']
)

#Validation set from training set:
val_size = CONFIG['val_split'] / (CONFIG['train_split'] + CONFIG['val_split'])
train_df, val_df = train_test_split(
    train_val_df,
    test_size=val_size,
    random_state=CONFIG['seed'],
    stratify=train_val_df['label']
)

print(f" Train set: {len(train_df)} samples")
print(f"  - Ham: {(train_df['label']==0).sum()}, Spam: {(train_df['label']==1).sum()}")
print(f" Validation set: {len(val_df)} samples")
print(f"  - Ham: {(val_df['label']==0).sum()}, Spam: {(val_df['label']==1).sum()}")
print(f" Test set: {len(test_df)} samples")
print(f"  - Ham: {(test_df['label']==0).sum()}, Spam: {(test_df['label']==1).sum()}")
```

```
 Train set: 60043 samples
  - Ham: 30023, Spam: 30020
 Validation set: 8578 samples
  - Ham: 4289, Spam: 4289
 Test set: 17156 samples
  - Ham: 8579, Spam: 8577
```

```
def tokenize_function(examples):

    tokens = tokenizer(
        examples['text_combined'],
        max_length=CONFIG['max_length'],
        padding='max_length',
        truncation=True,
        return_attention_mask=True
    )
    tokens['labels'] = examples['label']
    return tokens

tokenizer = AutoTokenizer.from_pretrained(CONFIG['model_name'])

train_dataset = Dataset.from_pandas(train_df[['text_combined', 'label']])
val_dataset = Dataset.from_pandas(val_df[['text_combined', 'label']])
test_dataset = Dataset.from_pandas(test_df[['text_combined', 'label']])

tokenized_train = train_dataset.map(tokenize_function, batched=True, remove_columns=['text_combined', 'label'])
tokenized_val = val_dataset.map(tokenize_function, batched=True, remove_columns=['text_combined', 'label'])
tokenized_test = test_dataset.map(tokenize_function, batched=True, remove_columns=['text_combined', 'label'])

#Final Dataset:
final_dataset = DatasetDict({'train': tokenized_train, 'validation': tokenized_val,'test': tokenized_test})

print(final_dataset)
```

```
tokenizer_config.json: 100%  48.0/48.0 [00:00<00:00, 3.03kB/s]
config.json: 100%  483/483 [00:00<00:00, 39.0kB/s]
vocab.txt: 100%  232k/232k [00:00<00:00, 3.90MB/s]
tokenizer.json: 100%  466k/466k [00:00<00:00, 6.74MB/s]
Map: 100%  60043/60043 [01:04<00:00, 1299.70 examples/s]
Map: 100%  8578/8578 [00:07<00:00, 1232.18 examples/s]
Map: 100%  17156/17156 [00:13<00:00, 1275.07 examples/s]
DatasetDict({
    train: Dataset({
        features: ['input_ids', 'attention_mask', 'labels'],
        num_rows: 8578
    })
    test: Dataset({
        features: ['input_ids', 'attention_mask', 'labels'],
        num_rows: 17156
    })
})
```

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification
model_name = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)
```

```
tokenizer_config.json: 100%  48.0/48.0 [00:00<00:00, 1.67kB/s]
config.json: 100%  570/570 [00:00<00:00, 41.5kB/s]
vocab.txt: 100%  232k/232k [00:00<00:00, 6.27MB/s]
tokenizer.json: 100%  466k/466k [00:00<00:00, 3.72MB/s]
model.safetensors: 100%  440M/440M [00:12<00:00, 23.9MB/s]
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly init
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
final_dataset.set_format("torch", columns=["input_ids", "attention_mask", "labels"])
```

```
from transformers import TrainingArguments

training_args = TrainingArguments(
    output_dir="./results",
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=3,
    logging_dir="./logs"
)
```

```
from transformers import AutoModelForSequenceClassification
model = AutoModelForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=2)
```

```
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly init
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
import numpy as np
from sklearn.metrics import accuracy_score, f1_score

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)
    return {
        "accuracy": accuracy_score(labels, predictions),
        "f1": f1_score(labels, predictions)
    }
```

```
from transformers import Trainer

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=final_dataset["train"],
    eval_dataset=final_dataset["test"],
    tokenizer=tokenizer,
    compute_metrics=compute_metrics
)
```

```
trainer.train()
```

```
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize?ref=models
wandb: Paste an API key from your profile and hit enter: ·····
wandb: WARNING If you're specifying your api key in code, ensure this code is not shared publicly.
wandb: WARNING Consider setting the WANDB_API_KEY environment variable, or running `wandb login` from the command line.
wandb: No netrc file found, creating one.
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
wandb: Currently logged in as: satya-gmsv (satya-gmsv-concordia-university) to https://api.wandb.ai. Use `wandb login --relogin` to
Tracking run with wandb version 0.22.2
Run data is saved locally in /content/wandb/run-20251017_145717-c12qa12q
Syncing run electric-thunder-1 to Weights & Biases (docs)
View project at https://wandb.ai/satya-gmsv-concordia-university/huggingface
View run at https://wandb.ai/satya-gmsv-concordia-university/huggingface/runs/c12qa12q
[11259/11259 1:16:46, Epoch 3/3]
```

Step	Training Loss
500	0.189900
1000	0.104500
1500	0.081900
2000	0.064900
2500	0.070000
3000	0.051600
3500	0.049400
4000	0.034700
4500	0.023500
5000	0.016000
5500	0.016500
6000	0.024900
6500	0.015800
7000	0.019400
7500	0.021500
8000	0.006500
8500	0.003400
9000	0.001900
9500	0.005600
10000	0.010900
10500	0.006800
11000	0.001400

```
TrainOutput(global_step=11259, training_loss=0.03655762196074202, metrics={'train_runtime': 4688.2496, 'train_samples_per_second':
38.421, 'train_steps_per_second': 2.402, 'total_flos': 1.184848282273536e+16, 'train_loss': 0.03655762196074202, 'epoch': 3.0})
```

```
results = trainer.evaluate()
print(results)
```

```
[1073/1073 05:53]
{'eval_loss': 0.041646894067525864, 'eval_accuracy': 0.9930636512007461, 'eval_f1': 0.9930680957651308, 'eval_runtime': 116.7256, 'ev
```

```
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Generate predictions
predictions = trainer.predict(final_dataset["test"])

# Extract predicted and true labels
predicted_labels = np.argmax(predictions.predictions, axis=1)
true_labels = predictions.label_ids
```

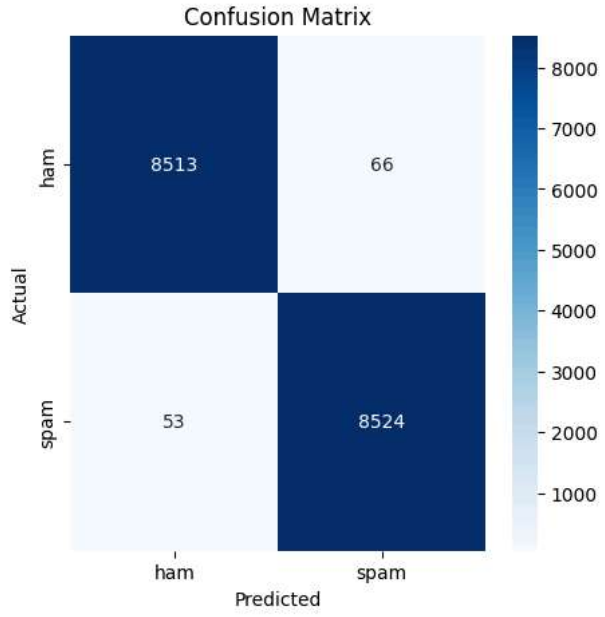
```
# Classification report
print(classification_report(true_labels, predicted_labels, target_names=["ham", "spam"]))

# Confusion matrix
conf_matrix = confusion_matrix(true_labels, predicted_labels)
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=["ham", "spam"], yticklabels=["ham", "spam"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

```
              precision    recall  f1-score   support

         ham       0.99      0.99      0.99      8579
        spam       0.99      0.99      0.99      8577

    accuracy                           0.99     17156
   macro avg       0.99      0.99      0.99     17156
weighted avg       0.99      0.99      0.99     17156
```




```
model.save_pretrained("./bert")
tokenizer.save_pretrained("./tokenizer")
```

```
('./tokenizer/tokenizer_config.json',
 './tokenizer/special_tokens_map.json',
 './tokenizer/vocab.txt',
 './tokenizer/added_tokens.json',
 './tokenizer/tokenizer.json')
```

```
from transformers import AutoModelForSequenceClassification, AutoTokenizer

model_name = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)
```

```
model.safetensors: 100%  268M/268M [00:11<00:00, 23.2MB/s]
Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-uncased and ar
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
final_dataset.set_format("torch", columns=["input_ids", "attention_mask", "labels"])
```

```
from transformers import TrainingArguments

training_args = TrainingArguments(
    output_dir="./distilbert_results",
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=3,
```

```
    logging_dir="./distilbert_logs"
)
```

```
import numpy as np
from sklearn.metrics import accuracy_score, f1_score

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)
    return {
        "accuracy": accuracy_score(labels, predictions),
        "f1": f1_score(labels, predictions)
    }
```

```
from transformers import Trainer

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=final_dataset["train"],
    eval_dataset=final_dataset["test"],
    tokenizer=tokenizer,
    compute_metrics=compute_metrics
)

trainer.train()
```

Step	Training Loss
500	0.174300
1000	0.092600
1500	0.060400
2000	0.059100
2500	0.054100
3000	0.039200
3500	0.044700
4000	0.026400
4500	0.014000
5000	0.021900
5500	0.011700
6000	0.016000
6500	0.018900
7000	0.014500
7500	0.015400

[11259/11259 37:40, Epoch 3/3]

Step	Training Loss
500	0.174300
1000	0.092600
1500	0.060400
2000	0.059100
2500	0.054100
3000	0.039200
3500	0.044700
4000	0.026400
4500	0.014000
5000	0.021900
5500	0.011700
6000	0.016000
6500	0.018900
7000	0.014500
7500	0.015400
8000	0.004500
8500	0.000200
9000	0.000700
9500	0.005300
10000	0.004300
10500	0.006100
11000	0.002700

```
TrainOutput(global_step=11259, training_loss=0.030536711697061104, metrics={'train_runtime': 2260.9335,
'train_samples_per_second': 79.67, 'train_steps_per_second': 4.98, 'total_flos': 5965305013126656.0, 'train_loss':
0.030536711697061104, 'epoch': 3.0})
```

```
results = trainer.evaluate()
print("Evaluation Results:", results)
```

```
model.save_pretrained("./distilbert_model")
tokenizer.save_pretrained("./distilbert_model_tokenizer")
```

```
('./distilbert_model_tokenizer/tokenizer_config.json',
 './distilbert_model_tokenizer/special_tokens_map.json',
 './distilbert_model_tokenizer/vocab.txt',
 './distilbert_model_tokenizer/added_tokens.json',
 './distilbert_model_tokenizer/tokenizer.json')
```

```
import numpy as np
import pandas as pd

predictions = trainer.predict(final_dataset["test"])
predicted_labels = np.argmax(predictions.predictions, axis=1)
true_labels = predictions.label_ids

df_predictions = pd.DataFrame({
    "true_label": true_labels,
    "predicted_label": predicted_labels
})
df_predictions.to_csv("distilbert_predictions.csv", index=False)
```

```
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

print(classification_report(true_labels, predicted_labels, target_names=["ham", "spam"]))

conf_matrix = confusion_matrix(true_labels, predicted_labels)
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=["ham", "spam"], yticklabels=["ham", "spam"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - DistilBERT Spam Classification")
```