# Mahati Murthy Akella 22070521027

1. import lib
2. load data
3. preprocess to text: -convert to lowercase -remove punctutation, no.s, stopwords aplly stemming or lemmatization
4. convert text to numeric using tf-idf
5. train using naive bayes(MultinomialNB0)
6. predict spam/ham for test data
7. Evaluate performance - accuracy, precission, recall & F1- score

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
```

```
df = pd.read_csv('spam.csv', encoding='latin1')
df.head()
```

| | v1 | v2 | Unnamed: 2 | Unnamed: 3 | Unnamed: 4 |
|---|---|---|---|---|---|
| 0 | ham | Go until jurong point, crazy.. Available only ... | NaN | NaN | NaN |
| 1 | ham | Ok lar... Joking wif u oni... | NaN | NaN | NaN |
| 2 | spam | Free entry in 2 a wkly comp to win FA Cup fina... | NaN | NaN | NaN |
| 3 | ham | U dun say so early hor... U c already then say... | NaN | NaN | NaN |
| 4 | ham | Nah I don't think he goes to usf, he lives aro... | NaN | NaN | NaN |

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.drop(columns=['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], inplace=True)
df.head()
```

| | v1 | v2 |
|---|---|---|
| 0 | ham | Go until jurong point, crazy.. Available only ... |
| 1 | ham | Ok lar... Joking wif u oni... |
| 2 | spam | Free entry in 2 a wkly comp to win FA Cup fina... |
| 3 | ham | U dun say so early hor... U c already then say... |
| 4 | ham | Nah I don't think he goes to usf, he lives aro... |

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
# Converting to lowercase
df['v2'] = df['v2'].str.lower()
df.head()
```

| | v1 | v2 |
|---|---|---|
| 0 | ham | go until jurong point, crazy.. available only ... |
| 1 | ham | ok lar... joking wif u oni... |
| 2 | spam | free entry in 2 a wkly comp to win fa cup fina... |
| 3 | ham | u dun say so early hor... u c already then say... |
| 4 | ham | nah i don't think he goes to usf, he lives aro... |

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
import re
import nltk
from nltk.corpus import stopwords

# Download stopwords (only first time)
nltk.download('stopwords')
```

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

def clean_text(text):
    text = re.sub(r'[^a-zA-Z]', ' ', text)  # Remove numbers and punctuation
    words = text.split()
    words = [word for word in words if word.lower() not in stop_words]  # Remove stopwords
    return ' '.join(words)

df['v2'] = df['v2'].apply(clean_text)
df.head()
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

| | v1 | v2 |
|---|---|---|
| 0 | ham | go jurong point crazy available bugis n great ... |
| 1 | ham | ok lar joking wif u oni |
| 2 | spam | free entry wkly comp win fa cup final tkts st ... |
| 3 | ham | u dun say early hor u c already say |
| 4 | ham | nah think goes usf lives around though |

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
# Applying stemming or lemmatization
from nltk.stem import PorterStemmer

stemmer = PorterStemmer()

def stem_text(text):
    words = text.split()
    stemmed_words = [stemmer.stem(word) for word in words]
    return ' '.join(stemmed_words)

df['v2'] = df['v2'].apply(stem_text)
df.head()
```

| | v1 | v2 |
|---|---|---|
| 0 | ham | go jurong point crazi avail bugi n great world... |
| 1 | ham | ok lar joke wif u oni |
| 2 | spam | free entri wkli comp win fa cup final tkt st m... |
| 3 | ham | u dun say earli hor u c alreadi say |
| 4 | ham | nah think goe usf live around though |

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
# Converting text into numeric using TF-IDF Vectorization
tfidf = TfidfVectorizer()
X = tfidf.fit_transform(df['v2'])
y = df['v1']
print(X)
```

```
<Compressed Sparse Row sparse matrix of dtype 'float64'
        with 44826 stored elements and shape (5572, 6221)>
  Coords        Values
  (0, 2148)     0.14084052842905104
  (0, 2794)     0.352275555327129
  (0, 4046)     0.24054119706179242
  (0, 1162)     0.2728131680559814
  (0, 377)      0.2634906267537017
  (0, 732)      0.29760381268143565
  (0, 2222)     0.19459721085856557
  (0, 6060)     0.23615475543085504
  (0, 2898)     0.2850448490727193
  (0, 730)      0.3362850956787249
  (0, 957)      0.29760381268143565
  (0, 2185)     0.16514812015268623
  (0, 188)      0.352275555327129
  (0, 5886)     0.19459721085856557
  (1, 3718)     0.2811632882742994
  (1, 2926)     0.4218684931830353
  (1, 2761)     0.47451057922863127
  (1, 5982)     0.4459451111953121
  (1, 3743)     0.5647537939557097
  (2, 1990)     0.12870765808799112
  (2, 1659)     0.3979792962044152
```

```
  (2, 6026)     0.2134950996997142
  (2, 1051)     0.2179115387053519
  (2, 5993)     0.16032634537022528
  (2, 1777)     0.5263325120342057
  :     :
  (5567, 3328)  0.2776026280950185
  (5567, 1560)  0.3003266961726054
  (5568, 2148)  0.29632963790124456
  (5568, 2431)  0.37453093229027423
  (5568, 1979)  0.5740011643413078
  (5568, 1690)  0.6651601234243666
  (5569, 5177)  0.5590776449483026
  (5569, 3402)  0.5301411472422174
  (5569, 3999)  0.6374814122151056
  (5570, 1990)  0.19245011423612288
  (5570, 5919)  0.2213846473559941
  (5570, 3007)  0.19108989254519423
  (5570, 3574)  0.254492372236279
  (5570, 5736)  0.25300362901092216
  (5570, 4927)  0.24880555093147316
  (5570, 757)   0.24748697443578171
  (5570, 1617)  0.2956554393937655
  (5570, 2078)  0.3335412598897019
  (5570, 2267)  0.25374199736641695
  (5570, 2640)  0.3162502671484596
  (5570, 52)    0.36992623143153675
  (5570, 573)   0.3543990344396484
  (5571, 3499)  0.48340924495090487
  (5571, 5592)  0.5294963215537312
  (5571, 4488)  0.6971005288744686
```

```
feature_names = tfidf.get_feature_names_out()
print(feature_names[2148])  # This will tell you which word index 2148 refers to
```

```
go
```

```
dense_matrix = X.todense()
print(dense_matrix)
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

```
import numpy as np

doc_id = 0
row = X[doc_id].toarray().flatten()
top_indices = row.argsort()[::-1][:10]
top_words = [(feature_names[i], row[i]) for i in top_indices]
print(top_words)
```

```
[('jurong', np.float64(0.352275555327129)), ('amor', np.float64(0.352275555327129)), ('buffet', np.float64(0.33628509567
```

```
from sklearn.feature_selection import chi2
import numpy as np

chi2_scores, _ = chi2(X, y)
top_indices = np.argsort(chi2_scores)[::-1][:50]
top_words = [tfidf.get_feature_names_out()[i] for i in top_indices]
print(top_words)
```

```
['txt', 'claim', 'prize', 'mobil', 'free', 'tone', 'www', 'servic', 'award', 'call', 'uk', 'nokia', 'guarante', 'urgent'
```

```
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)
```

```
# Training a Naive Bayes classifier
model = MultinomialNB()
model.fit(X_train, y_train)
```

```
▾ MultinomialNB ⓘ ?
MultinomialNB()
```

```
# Predict spam/ham for the test data
y_pred = model.predict(X_test)
print(y_pred)
```

```
['ham' 'ham' 'ham' ... 'ham' 'ham' 'spam']
```

```
# Evaluating model performance
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report, confusion_matrix
print("--- Model Evaluation ---")
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, pos_label='spam')
recall = recall_score(y_test, y_pred, pos_label='spam')
f1 = f1_score(y_test, y_pred, pos_label='spam')

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}  (Of all messages predicted as spam, what percentage actually were?)")
print(f"Recall: {recall:.4f}     (Of all actual spam messages, what percentage did we catch?)")
print(f"F1-Score: {f1:.4f}    (A weighted average of Precision and Recall)")

print("\n--- Classification Report ---")
print(classification_report(y_test, y_pred))

print("\n--- Confusion Matrix ---")
print("             Predicted")
print("            Ham | Spam")
print(confusion_matrix(y_test, y_pred))
```

```
--- Model Evaluation ---
Accuracy: 0.9659
Precision: 1.0000  (Of all messages predicted as spam, what percentage actually were?)
Recall: 0.7467     (Of all actual spam messages, what percentage did we catch?)
F1-Score: 0.8550    (A weighted average of Precision and Recall)

--- Classification Report ---
              precision    recall  f1-score   support

         ham       0.96      1.00      0.98       965
        spam       1.00      0.75      0.85       150

    accuracy                           0.97      1115
   macro avg       0.98      0.87      0.92      1115
weighted avg       0.97      0.97      0.96      1115


--- Confusion Matrix ---
             Predicted
            Ham | Spam
[[965   0]
 [ 38 112]]
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split

# --- Training Part (do this once) ---
X_text = df['v2']   # cleaned/preprocessed messages
y = df['v1']        # labels (ham/spam)

tfidf_vectorizer = TfidfVectorizer()
X_tfidf = tfidf_vectorizer.fit_transform(X_text)   # ✅ Fit here

X_train, X_test, y_train, y_test = train_test_split(X_tfidf, y, test_size=0.2, random_state=42)

model = MultinomialNB()
model.fit(X_train, y_train)   # ✅ Train model
```

```
▾ MultinomialNB ⓘ ?
MultinomialNB()
```

```
# --- Prediction on User Input ---
new_message = input("Enter a message to classify: ")
preprocessed_message = preprocess_text(new_message)
print(f"Cleaned message: '{preprocessed_message}'")

# ✅ Use the already fitted vectorizer
vectorized_message = tfidf_vectorizer.transform([preprocessed_message])

# ✅ Predict with the trained model
prediction = model.predict(vectorized_message)
```

```
print("\n--- Result ---")
print(f"The model predicts that this message is: '{prediction[0].upper()}'")
```

```
Enter a message to classify: Enter a message to classify: Congratulations! You have won a free lottery ticket worth $100
Cleaned message: 'enter messag classifi congratul free lotteri ticket worth call claim prize'

--- Result ---
The model predicts that this message is: 'SPAM'
```