Project Title: SMS Spam Detection using Naive Bayes

Objective: To classify short text messages as ham (legitimate) or spam using Natural Language Processing (NLP) and the highly efficient Multinomial Naive Bayes algorithm.

Import necessary libraries

This NLP environment by importing all necessary libraries. We bring in Pandas for data handling, NLTK (Natural Language Toolkit) for text processing, and Sklearn for machine learning models and metrics.

```
import pandas as pd
import re
import string
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
import kagglehub
import os
```

Data Collection and Loading

Source: The project uses a dataset from Kaggle titled "SMS Spam Collection Dataset"

Datasets: spam.csv

```
print("Loading dataset...")

# --- Load the dataset using kagglehub ---
try:
    from kagglehub import KaggleDatasetAdapter
    file_path = "spam.csv"
    # The dataset contains two columns: 'v1' (label) and 'v2' (text).
    df = kagglehub.load_dataset(
        KaggleDatasetAdapter.PANDAS,
        "uciml/sms-spam-collection-dataset",
        file_path,
        pandas_kwargs={'encoding': 'latin-1'}
    )
    # Rename columns for clarity
    df.rename(columns={'v1': 'label', 'v2': 'text'}, inplace=True)
    print("Dataset loaded successfully from KaggleHub.")
except Exception as e:
    print(f"Error loading from KaggleHub: {e}. Attempting to load from a local file.")
    try:
        df = pd.read_csv('spam.csv', encoding='latin-1')
        df.rename(columns={'v1': 'label', 'v2': 'text'}, inplace=True)
        print("Dataset loaded successfully from local file.")
    except FileNotFoundError:
        print("Error: 'spam.csv' not found. Please ensure the file is in the correct directory.")
        exit()
```

```
Loading dataset...
/tmp/ipython-input-989242744.py:8: DeprecationWarning: Use dataset_load() instead of load_dataset(). load_dataset() will b
  df = kagglehub.load_dataset(
Using Colab cache for faster access to the 'sms-spam-collection-dataset' dataset.
Dataset loaded successfully from KaggleHub.
```

Exploratory Data Analysis (EDA)

The output shows the dataset's structure, confirming five columns (we only need the label and text). The Class Distribution reveals that 13.41% of the messages are classified as spam. While this is an imbalance, it is less severe than the fraud detection task, which is one reason why the simpler Naive Bayes model is so effective here.

```
# --- Exploratory Data Analysis (EDA) ---
print("\n--- Dataset Overview ---")
print(df.info())
```

```
print("\nFirst 5 rows of the dataset:")
print(df.head())

# Check the class distribution
print("\n--- Class Distribution ---")
print(df['label'].value_counts())
print(f"\nSpam messages make up {round(df['label'].value_counts()['spam']/len(df) * 100, 2)}% of the dataset.")
```

```
--- Dataset Overview ---
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   label       5572 non-null   object
 1   text        5572 non-null   object
 2   Unnamed: 2  50 non-null     object
 3   Unnamed: 3  12 non-null     object
 4   Unnamed: 4  6 non-null      object
dtypes: object(5)
memory usage: 217.8+ KB
None

First 5 rows of the dataset:
  label                                               text Unnamed: 2  \
0   ham  Go until jurong point, crazy.. Available only ...        NaN
1   ham                      Ok lar... Joking wif u oni...        NaN
2  spam  Free entry in 2 a wkly comp to win FA Cup fina...        NaN
3   ham  U dun say so early hor... U c already then say...        NaN
4   ham  Nah I don't think he goes to usf, he lives aro...        NaN

  Unnamed: 3 Unnamed: 4
0        NaN        NaN
1        NaN        NaN
2        NaN        NaN
3        NaN        NaN
4        NaN        NaN

--- Class Distribution ---
label
ham     4825
spam     747
Name: count, dtype: int64

Spam messages make up 13.41% of the dataset.
```

Preprocessing and Text Cleaning

This is the text preprocessing phase, critical for all NLP projects. The custom clean_text function executes several steps using NLTK: it removes Punctuation, eliminates common words (or Stop Words), and uses Lemmatization to reduce words to their base form (e.g., 'running' to 'run'). This cleansing ensures the model focuses only on the meaningful, unique keywords in each SMS.

```
# --- Preprocessing ---
print("\n--- Preprocessing Data ---")
nltk.download('stopwords', quiet=True)
nltk.download('punkt', quiet=True)
nltk.download('punkt_tab', quiet=True)
nltk.download('wordnet', quiet=True)


def clean_text(text):
    text = text.lower()
    text = re.sub(r'\[.*?\]', '', text)
    translator = str.maketrans('', '', string.punctuation)
    text = text.translate(translator)
    tokens = word_tokenize(text)
    stop_words = set(stopwords.words('english'))
    lemmatizer = WordNetLemmatizer()
    lemmatized_tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words and len(word) > 1]
    return ' '.join(lemmatized_tokens)

df['cleaned_text'] = df['text'].apply(clean_text)
print("\nText data cleaned successfully.")
print("Original Text vs. Cleaned Text:")
print(df[['text', 'cleaned_text']].head())
```

```
--- Preprocessing Data ---

Text data cleaned successfully.
Original Text vs. Cleaned Text:
```

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

                                      cleaned_text
0  go jurong point crazy available bugis great wo...
1                             ok lar joking wif oni
2  free entry wkly comp win fa cup final tkts 21s...
3                      dun say early hor already say
4           nah dont think go usf life around though
```

Feature Representation (TF-IDF)

This step converts the cleaned text into a numerical format suitable for machine learning. First, the data is split into 80% for training and 20% for testing using stratify=y to maintain the spam/ham ratio. We then use TF-IDF Vectorization to assign numerical scores to words based on their importance, creating a feature matrix of size (4457, 5000) for training.

```
# --- Feature Representation (TF-IDF) ---
print("\n--- Feature Representation with TF-IDF ---")
X = df['cleaned_text']
y = df['label']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Create a TF-IDF Vectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=5000)
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

print(f"Shape of X_train_tfidf: {X_train_tfidf.shape}")
print(f"Shape of X_test_tfidf: {X_test_tfidf.shape}")
```

```
--- Feature Representation with TF-IDF ---
Shape of X_train_tfidf: (4457, 5000)
Shape of X_test_tfidf: (1115, 5000)
```

Model Training and Evaluation

We train the Multinomial Naive Bayes classifier, which is perfectly suited for text classification with TF-IDF features. This model is known for its speed and reliability in filtering spam.

```
# --- Model Training and Evaluation ---
print("\n--- Training and Evaluating Naive Bayes Model ---")
# Use the Naive Bayes Classifier, a highly effective and efficient model for this task.
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train_tfidf, y_train)

print("\n--- Evaluating Naive Bayes on the Test Set ---")
y_pred = nb_classifier.predict(X_test_tfidf)
```

```
--- Training and Evaluating Naive Bayes Model ---

--- Evaluating Naive Bayes on the Test Set ---
```

performance metrics:

The output displays the final metrics, including high Accuracy (0.9668) and an excellent breakdown of Precision and Recall in the Classification Report.

```
# Evaluate the model using a confusion matrix and classification report
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy:.4f}")
print("Classification Report:")
print(report)

print("\nTask complete. The Naive Bayes model has been trained and evaluated.")
```

```
Accuracy: 0.9668
Classification Report:
              precision    recall  f1-score   support

         ham       0.96      1.00      0.98       966
        spam       0.99      0.76      0.86       149

    accuracy                           0.97      1115
   macro avg       0.98      0.88      0.92      1115
weighted avg       0.97      0.97      0.96      1115
```

Task complete. The Naive Bayes model has been trained and evaluated.