```
print("NLP Miniproject")
print("Title: Movie Genre Classifier Recommender")
print("Created by Dhwaj, Kaustav, Hadi & Eric")
```

```
NLP Miniproject
Title: Movie Genre Classifier Recommender
Created by Dhwaj, Kaustav, Hadi & Eric
```

```
# Install required libraries (run this cell first)
!pip install nltk scikit-learn pandas numpy matplotlib seaborn

# Import all necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# Download NLTK data
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('punkt')

# Scikit-learn imports
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report
from sklearn.metrics.pairwise import cosine_similarity

import warnings
warnings.filterwarnings('ignore')

print("✅ All libraries imported successfully!")
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.12/dist-packages (3.9.1)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.12/dist-packages (1.6.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages (2.2.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (2.0.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (3.10.0)
Requirement already satisfied: seaborn in /usr/local/lib/python3.12/dist-packages (0.13.2)
Requirement already satisfied: click in /usr/local/lib/python3.12/dist-packages (from nltk) (8.3.0)
Requirement already satisfied: joblib in /usr/local/lib/python3.12/dist-packages (from nltk) (1.5.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.12/dist-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from nltk) (4.67.1)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.16.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (4.60.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.4.9)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (25.0)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (3.2.5)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
✅ All libraries imported successfully!
```

```
# Load the dataset
df = pd.read_csv('imdb_top_1000.csv')  # Replace with your filename
```

```
# Display basic info
print(f"Dataset shape: {df.shape}")
print(f"\nColumn names: {df.columns.tolist()}")
df.head()
```

```
Dataset shape: (1000, 16)

Column names: ['Poster_Link', 'Series_Title', 'Released_Year', 'Certificate', 'Runtime', 'Genre', 'IMDB_Rating', 'Overview', 'Me
```

	Poster_Link	Series_Title	Released_Year	Certificate	Runtime	Genre	IMDB_Rating	Overview	Meta_s
0	https://m.media-amazon.com/images/M/MV5BMDFkYT...	The Shawshank Redemption	1994	A	142 min	Drama	9.3	Two imprisoned men bond over a number of years...	
1	https://m.media-amazon.com/images/M/MV5BM2MyNj...	The Godfather	1972	A	175 min	Crime, Drama	9.2	An organized crime dynasty's aging patriarch t...	1
2	https://m.media-amazon.com/images/M/MV5BMTMxNT...	The Dark Knight	2008	UA	152 min	Action, Crime, Drama	9.0	When the menace known as the Joker wreaks havo...	
3	https://m.media-amazon.com/images/M/MV5BMWMwMG...	The Godfather: Part II	1974	A	202 min	Crime, Drama	9.0	The early life and career of Vito Corleone in ...	
4	https://m.media-amazon.com/images/M/MV5BMWU4N2...	12 Angry Men	1957	U	96 min	Crime, Drama	9.0	A jury holdout attempts to prevent a miscarria...	

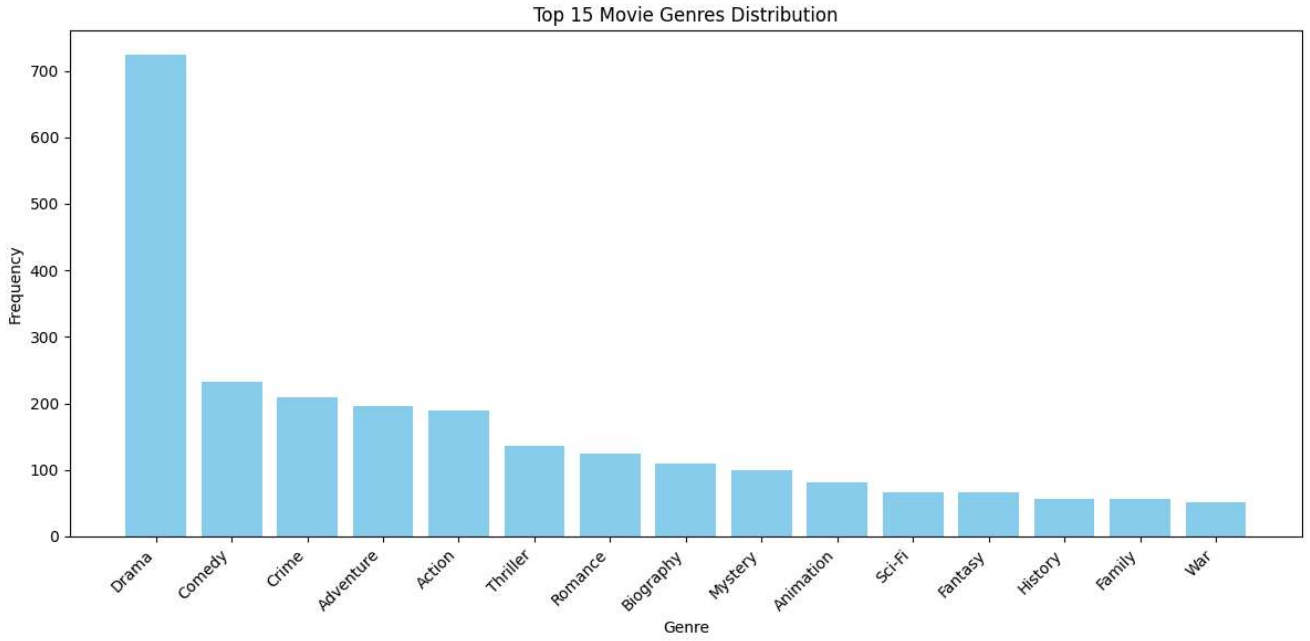
```
# Check dataset structure
print("="*50)
print("DATASET OVERVIEW")
print("="*50)
print(f"Total movies: {len(df)}")
print(f"\nColumns: {df.columns.tolist()}")
print(f"\nData types:\n{df.dtypes}")
print(f"\nMissing values:\n{df.isnull().sum()}")

# Display sample rows
print("\n" + "="*50)
print("SAMPLE DATA")
print("="*50)
df.head(10)
```

```
==================================================
DATASET OVERVIEW
==================================================
Total movies: 1000

Columns: ['Poster_Link', 'Series_Title', 'Released_Year', 'Certificate', 'Runtime', 'Genre', 'IMDB_Rating', 'Overview', 'Meta_

Data types:
Poster_Link       object
Series_Title      object
Released_Year     object
Certificate       object
Runtime           object
Genre             object
IMDB_Rating      float64
Overview          object
Meta_score       float64
Director          object
Star1             object
Star2             object
Star3             object
Star4             object
No_of_Votes        int64
Gross             object
dtype: object

Missing values:
Poster_Link        0
Series_Title       0
Released_Year      0
Certificate      101
Runtime            0
Genre              0
IMDB_Rating        0
Overview           0
Meta_score       157
Director           0
```

```
# Assuming your dataset has columns: 'title', 'plot', 'genres'
# Adjust column names based on your actual dataset

# Check unique genres
print("="*50)
print("GENRE ANALYSIS")
print("="*50)

# If genres are in list format [Genre1, Genre2, ...]
# Parse genres
all_genres = []
for genres in df['Genre']:
    if isinstance(genres, str):
        # If stored as string representation of list
        import ast
        try:
            genre_list = ast.literal_eval(genres)
            all_genres.extend(genre_list)
        except:
            genre_list = genres.split(',')
            all_genres.extend([g.strip() for g in genre_list])
    else:
        all_genres.extend(genres)

# Count genre frequency
from collections import Counter
genre_counts = Counter(all_genres)
print(f"Total unique genres: {len(genre_counts)}")
print(f"\nTop 15 genres:")
for genre, count in genre_counts.most_common(15):
    print(f"  {genre}: {count}")

# Visualize genre distribution
plt.figure(figsize=(12, 6))
top_genres = dict(genre_counts.most_common(15))
plt.bar(top_genres.keys(), top_genres.values(), color='skyblue')
plt.xlabel('Genre')
plt.ylabel('Frequency')
plt.title('Top 15 Movie Genres Distribution')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

```
==================================================
GENRE ANALYSIS
==================================================
Total unique genres: 21
Top 15 genres:
  Drama: 724
  Comedy: 233
  Crime: 209
  Adventure: 196
  Action: 189
  Thriller: 137
  Romance: 125
  Biography: 109
  Mystery: 99
  Animation: 82
  Sci-Fi: 67
  Fantasy: 66
  History: 56
  Family: 56
  War: 51
```

5	https://m.media-amazon.com/images/M/MV5BNzA5ZD...	The Lord of the Rings: The Return of the King	2003	U	201 min	Action, Adventure, Drama	8.9	Gandalf and Aragorn lead the World of Men agai...
6	https://m.media-amazon.com/images/M/MV5BNGNhMD...	Pulp Fiction	1994	A	154 min	Crime, Drama	8.9	The lives of two mob hitmen, a boxer, a gangst...
7	https://m.media-amazon.com/images/M/MV5BNDE4OT...	Schindler's List	1993	A	195 min	Biography, Drama, History	8.9	In German-occupied Poland during World War II,...




```
# Create a clean dataframe with required columns
# Adjust column names based on your dataset
df_clean = df[['Series_Title', 'Overview', 'Genre']].copy()

# Remove rows with missing plot or genres
df_clean = df_clean.dropna(subset=['Overview', 'Genre'])

# Remove duplicates
df_clean = df_clean.drop_duplicates(subset=['Overview'])

print(f"✅ Clean dataset shape: {df_clean.shape}")
df_clean.head()
```

```
✅ Clean dataset shape: (1000, 3)
```

	Series_Title	Overview	Genre
0	The Shawshank Redemption	Two imprisoned men bond over a number of years...	Drama
1	The Godfather	An organized crime dynasty's aging patriarch t...	Crime, Drama
2	The Dark Knight	When the menace known as the Joker wreaks havo...	Action, Crime, Drama
3	The Godfather: Part II	The early life and career of Vito Corleone in ...	Crime, Drama
4	12 Angry Men	A jury holdout attempts to prevent a miscarria...	Crime, Drama

```
# Initialize lemmatizer
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    """
    Comprehensive text preprocessing function
    """
    if not isinstance(text, str):
        return ""

    # Convert to lowercase
    text = text.lower()

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

    # Tokenize
    tokens = text.split()

    # Remove stopwords and lemmatize
    tokens = [lemmatizer.lemmatize(word) for word in tokens
              if word not in stop_words and len(word) > 2]

    # Join back to string
    return ' '.join(tokens)

# Test the function
sample_plot = df_clean['Overview'].iloc[0]
print("Original plot:")
print(sample_plot[:200])
print("\n" + "="*50)
print("\nPreprocessed plot:")
print(preprocess_text(sample_plot)[:200])
```

```
Original plot:
Two imprisoned men bond over a number of years, finding solace and eventual redemption through acts of common decency.

==================================================

Preprocessed plot:
two imprisoned men bond number year finding solace eventual redemption act common decency
```

```
# Apply preprocessing to all plots
print("🔄 Preprocessing all plots... This may take a few minutes.")
df_clean['plot_processed'] = df_clean['Overview'].apply(preprocess_text)

# Remove empty processed plots
df_clean = df_clean[df_clean['plot_processed'].str.len() > 0]

print(f"✅ Preprocessing complete!")
print(f"Final dataset shape: {df_clean.shape}")

# Display sample
df_clean[['Series_Title', 'plot_processed', 'Genre']].head()
```

```
🔄 Preprocessing all plots... This may take a few minutes.
✅ Preprocessing complete!
Final dataset shape: (1000, 4)
```

	Series_Title	plot_processed	Genre
0	The Shawshank Redemption	two imprisoned men bond number year finding so...	Drama
1	The Godfather	organized crime dynasty aging patriarch transf...	Crime, Drama
2	The Dark Knight	menace known joker wreaks havoc chaos people g...	Action, Crime, Drama
3	The Godfather: Part II	early life career vito corleone new york city ...	Crime, Drama
4	12 Angry Men	jury holdout attempt prevent miscarriage justi...	Crime, Drama

```
# Parse genres into lists
def parse_genres(genre_str):
    """Convert genre string to list"""
    if isinstance(genre_str, str):
        try:
            # If it's a string representation of a list
            import ast
```

```
            return ast.literal_eval(genre_str)
        except:
            # If it's comma-separated
            return [g.strip() for g in genre_str.split(',')]
    return genre_str

df_clean['genres_list'] = df_clean['Genre'].apply(parse_genres)

# Multi-label binarization
mlb = MultiLabelBinarizer()
genre_encoded = mlb.fit_transform(df_clean['genres_list'])

print(f"✅ Genre encoding complete!")
print(f"Total unique genres: {len(mlb.classes_)}")
print(f"Genre classes: {mlb.classes_}")
print(f"\nEncoded shape: {genre_encoded.shape}")

# Create a dataframe of encoded genres
genre_df = pd.DataFrame(genre_encoded, columns=mlb.classes_)
print("\nSample encoded genres:")
genre_df.head()
```

```
✅ Genre encoding complete!
Total unique genres: 21
Genre classes: ['Action' 'Adventure' 'Animation' 'Biography' 'Comedy' 'Crime' 'Drama'
 'Family' 'Fantasy' 'Film-Noir' 'History' 'Horror' 'Music' 'Musical'
 'Mystery' 'Romance' 'Sci-Fi' 'Sport' 'Thriller' 'War' 'Western']

Encoded shape: (1000, 21)

Sample encoded genres:
```

	Action	Adventure	Animation	Biography	Comedy	Crime	Drama	Family	Fantasy	Film-Noir	...	Horror	Music	Musical	Mystery	Ro
0	0	0	0	0	0	0	1	0	0	0	...	0	0	0	0	
1	0	0	0	0	0	1	1	0	0	0	...	0	0	0	0	
2	1	0	0	0	0	1	1	0	0	0	...	0	0	0	0	
3	0	0	0	0	0	1	1	0	0	0	...	0	0	0	0	
4	0	0	0	0	0	1	1	0	0	0	...	0	0	0	0	

5 rows × 21 columns

```
# Create TF-IDF vectors from processed plots
print("🔄 Creating TF-IDF vectors...")

# Initialize TF-IDF vectorizer
tfidf = TfidfVectorizer(
    max_features=5000,      # Limit to top 5000 features
    min_df=2,               # Ignore terms that appear in less than 2 documents
    max_df=0.8,             # Ignore terms that appear in more than 80% of documents
    ngram_range=(1, 2)      # Use unigrams and bigrams
)

# Fit and transform the processed plots
X = tfidf.fit_transform(df_clean['plot_processed'])
y = genre_encoded

print(f"✅ TF-IDF vectorization complete!")
print(f"Feature matrix shape: {X.shape}")
print(f"Label matrix shape: {y.shape}")
print(f"Total features: {len(tfidf.get_feature_names_out())}")
```

```
🔄 Creating TF-IDF vectors...
✅ TF-IDF vectorization complete!
Feature matrix shape: (1000, 2430)
Label matrix shape: (1000, 21)
Total features: 2430
```

```
# 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
)
```

```
print(f"✅ Data split complete!")
print(f"Training set: {X_train.shape}")
print(f"Testing set: {X_test.shape}")
```

```
✅ Data split complete!
Training set: (800, 2430)
Testing set: (200, 2430)
```

```
# Dictionary to store models and results
models = {}
results = {}

print("="*50)
print("TRAINING MODELS")
print("="*50)

# 1. Logistic Regression
print("\n1️⃣ Training Logistic Regression...")
lr_model = OneVsRestClassifier(LogisticRegression(max_iter=1000, random_state=42))
lr_model.fit(X_train, y_train)
models['Logistic Regression'] = lr_model
print("✅ Logistic Regression trained!")

# 2. Linear SVM
print("\n2️⃣ Training Linear SVM...")
svm_model = OneVsRestClassifier(LinearSVC(random_state=42))
svm_model.fit(X_train, y_train)
models['Linear SVM'] = svm_model
print("✅ Linear SVM trained!")

# 3. Naive Bayes
print("\n3️⃣ Training Multinomial Naive Bayes...")
nb_model = OneVsRestClassifier(MultinomialNB())
nb_model.fit(X_train, y_train)
models['Naive Bayes'] = nb_model
print("✅ Naive Bayes trained!")

print("\n" + "="*50)
print("✅ ALL MODELS TRAINED SUCCESSFULLY!")
print("="*50)
```

```
==================================================
TRAINING MODELS
==================================================

1️⃣ Training Logistic Regression...
✅ Logistic Regression trained!

2️⃣ Training Linear SVM...
✅ Linear SVM trained!

3️⃣ Training Multinomial Naive Bayes...
✅ Naive Bayes trained!

==================================================
✅ ALL MODELS TRAINED SUCCESSFULLY!
==================================================
```

```
# Evaluate each model
print("="*50)
print("MODEL EVALUATION")
print("="*50)

for name, model in models.items():
    print(f"\n📊 {name}")
    print("-" * 40)

    # Make predictions
    y_pred = model.predict(X_test)

    # Calculate metrics
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='micro')
    recall = recall_score(y_test, y_pred, average='micro')
    f1 = f1_score(y_test, y_pred, average='micro')

    # Store results
```

```
        results[name] = {
            'accuracy': accuracy,
            'precision': precision,
            'recall': recall,
            'f1_score': f1
        }

        print(f"Accuracy:  {accuracy:.4f}")
        print(f"Precision: {precision:.4f}")
        print(f"Recall:    {recall:.4f}")
        print(f"F1 Score:  {f1:.4f}")

    # Compare results
    print("\n" + "="*50)
    print("MODEL COMPARISON")
    print("="*50)
    results_df = pd.DataFrame(results).T
    print(results_df)

    # Visualize comparison
    results_df.plot(kind='bar', figsize=(12, 6))
    plt.title('Model Performance Comparison')
    plt.xlabel('Model')
    plt.ylabel('Score')
    plt.legend(loc='lower right')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

```
==================================================
MODEL EVALUATION
==================================================

📊 Logistic Regression
----------------------------------------
Accuracy:  0.0750
Precision: 0.7313
Recall:    0.2888
F1 Score:  0.4161
```

Predict Genre for a New Movie Plot

```
# Get user input for a movie plot
new_movie_plot = input("Enter the plot of the movie: ")

# Preprocess the new plot
processed_new_plot = preprocess_text(new_movie_plot)

# Transform the processed plot using the fitted TF-IDF vectorizer
# We need to reshape for a single sample
new_plot_vector = tfidf.transform([processed_new_plot])

print("\n✅ Preprocessing and vectorization complete for the new plot.")
```

```
Enter the plot of the movie: THERE WAS A boy @@ 1947 he was lonely he went to HAUNTED HOUSE it was dark he saw ghosts HE GOT &&

✅ Preprocessing and vectorization complete for the new plot.
```

```
==================================================
```

```
print("Preprocessed user input plot:")
print(processed_new_plot)
```

```
Preprocessed user input plot:
boy lonely went haunted house dark saw ghost got possessed
```

```
Logistic Regression     0.075   0.725000  0.284872  0.409027
Linear SVM              0.080   0.711297  0.333988  0.454545
Naive Bayes             0.070   0.731343  0.288802  0.414085

Model Performance Comparison
```

```
# Make predictions using the trained models
print("="*50)
print("PREDICTED GENRES")
print("="*50)

for name, model in models.items():
    # Predict the genre probabilities/scores
    # Some models have predict_proba, others just predict
    try:
        predictions_proba = model.predict_proba(new_plot_vector)
        # Get the top predicted genres (adjust threshold if needed)
        predicted_genres_indices = (predictions_proba > 0.1).nonzero()[1] # Example threshold
        predicted_genres = mlb.classes_[predicted_genres_indices]
    except AttributeError:
        # For models without predict_proba (like LinearSVC)
        predictions = model.predict(new_plot_vector)
        predicted_genres_indices = predictions.nonzero()[1]
        predicted_genres = mlb.classes_[predicted_genres_indices]


    print(f"\n📊 {name}:")
    if len(predicted_genres) > 0:
        print(f"  Predicted Genres: {', '.join(predicted_genres)}")
    else:
        print("  No specific genres predicted above the threshold.")
```

```
==================================================
PREDICTED GENRES
==================================================

📊 Logistic Regression:
  Predicted Genres: Action, Adventure, Animation, Biography, Comedy, Crime, Drama, Thriller

📊 Linear SVM:
  Predicted Genres: Drama

📊 Naive Bayes:
  Predicted Genres: Action, Adventure, Comedy, Crime, Drama
```