

Imports

GitHub Link

<https://github.com/arunaarun/AAI-520-group-04>

Importing Libraries

```
In [1]: # Import necessary libraries
import pandas as pd
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from collections import Counter
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from textblob import TextBlob
from wordcloud import WordCloud

# Download necessary NLTK data
nltk.download('averaged_perceptron_tagger')
nltk.download('maxent_ne_chunker')
nltk.download('words')
nltk.download('punkt', quiet=True)
nltk.download('stopwords', quiet=True)

# Install necessary packages
import subprocess
import sys

def install(package):
    subprocess.check_call([sys.executable, "-m", "pip", "install", package])

install('matplotlib')
install('seaborn')
install('textblob')
install('wordcloud')
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data] /Users/rolosworld/nltk_data...
[nltk_data] Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data] /Users/rolosworld/nltk_data...
[nltk_data] Unzipping chunkers/maxent_ne_chunker.zip.
[nltk_data] Downloading package words to
[nltk_data] /Users/rolosworld/nltk_data...
[nltk_data] Unzipping corpora/words.zip.
```

Defaulting to user installation because normal site-packages is not writeable

Requirement already satisfied: matplotlib in /Library/Python/3.9/site-packages (3.9.0)

Requirement already satisfied: contourpy>=1.0.1 in /Library/Python/3.9/site-packages (from matplotlib) (1.2.1)

Requirement already satisfied: cyclor>=0.10 in /Library/Python/3.9/site-packages (from matplotlib) (0.12.1)

Requirement already satisfied: fonttools>=4.22.0 in /Library/Python/3.9/site-packages (from matplotlib) (4.51.0)

Requirement already satisfied: kiwisolver>=1.3.1 in /Library/Python/3.9/site-packages (from matplotlib) (1.4.5)

Requirement already satisfied: numpy>=1.23 in /Library/Python/3.9/site-packages (from matplotlib) (1.26.4)

Requirement already satisfied: packaging>=20.0 in /Library/Python/3.9/site-packages (from matplotlib) (24.0)

Requirement already satisfied: pillow>=8 in /Library/Python/3.9/site-packages (from matplotlib) (10.3.0)

Requirement already satisfied: pyparsing>=2.3.1 in /Library/Python/3.9/site-packages (from matplotlib) (3.1.2)

Requirement already satisfied: python-dateutil>=2.7 in /Library/Python/3.9/site-packages (from matplotlib) (2.9.0.post0)

Requirement already satisfied: importlib-resources>=3.2.0 in /Library/Python/3.9/site-packages (from matplotlib) (6.4.0)

Requirement already satisfied: zipp>=3.1.0 in /Library/Python/3.9/site-packages (from importlib-resources>=3.2.0->matplotlib) (3.18.2)

Requirement already satisfied: six>=1.5 in /Library/Developer/CommandLineTools/Library/Frameworks/Python3.framework/Versions/3.9/lib/python3.9/site-packages (from python-dateutil>=2.7->matplotlib) (1.15.0)

Defaulting to user installation because normal site-packages is not writeable

Requirement already satisfied: seaborn in /Library/Python/3.9/site-packages (0.13.2)

Requirement already satisfied: numpy!=1.24.0,>=1.20 in /Library/Python/3.9/site-packages (from seaborn) (1.26.4)

Requirement already satisfied: pandas>=1.2 in /Library/Python/3.9/site-packages (from seaborn) (2.2.2)

Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in /Library/Python/3.9/site-packages (from seaborn) (3.9.0)

Requirement already satisfied: contourpy>=1.0.1 in /Library/Python/3.9/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.2.1)

Requirement already satisfied: cyclor>=0.10 in /Library/Python/3.9/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (0.12.1)

Requirement already satisfied: fonttools>=4.22.0 in /Library/Python/3.9/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (4.51.0)

Requirement already satisfied: kiwisolver>=1.3.1 in /Library/Python/3.9/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.4.5)

Requirement already satisfied: packaging>=20.0 in /Library/Python/3.9/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (24.0)

Requirement already satisfied: pillow>=8 in /Library/Python/3.9/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (10.3.0)

Requirement already satisfied: pyparsing>=2.3.1 in /Library/Python/3.9/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (3.1.2)

Requirement already satisfied: python-dateutil>=2.7 in /Library/Python/3.9/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (2.9.0.post0)

Requirement already satisfied: importlib-resources>=3.2.0 in /Library/Python/3.9/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (6.4.0)

n/3.9/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (6.4.0)
Requirement already satisfied: pytz>=2020.1 in /Library/Python/3.9/site-packages (from pandas>=1.2->seaborn) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in /Library/Python/3.9/site-packages (from pandas>=1.2->seaborn) (2024.1)
Requirement already satisfied: zipp>=3.1.0 in /Library/Python/3.9/site-packages (from importlib-resources>=3.2.0->matplotlib!=3.6.1,>=3.4->seaborn) (3.18.2)
Requirement already satisfied: six>=1.5 in /Library/Developer/CommandLineTools/Library/Frameworks/Python3.framework/Versions/3.9/lib/python3.9/site-packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.15.0)
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: textblob in /Library/Python/3.9/site-packages (0.18.0.post0)
Requirement already satisfied: nltk>=3.8 in /Users/rolosworld/Library/Python/3.9/lib/python/site-packages (from textblob) (3.9.1)
Requirement already satisfied: click in /Users/rolosworld/Library/Python/3.9/lib/python/site-packages (from nltk>=3.8->textblob) (8.1.7)
Requirement already satisfied: joblib in /Library/Python/3.9/site-packages (from nltk>=3.8->textblob) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in /Users/rolosworld/Library/Python/3.9/lib/python/site-packages (from nltk>=3.8->textblob) (2024.9.11)
Requirement already satisfied: tqdm in /Users/rolosworld/Library/Python/3.9/lib/python/site-packages (from nltk>=3.8->textblob) (4.66.5)
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: wordcloud in /Users/rolosworld/Library/Python/3.9/lib/python/site-packages (1.9.3)
Requirement already satisfied: numpy>=1.6.1 in /Library/Python/3.9/site-packages (from wordcloud) (1.26.4)
Requirement already satisfied: pillow in /Library/Python/3.9/site-packages (from wordcloud) (10.3.0)
Requirement already satisfied: matplotlib in /Library/Python/3.9/site-packages (from wordcloud) (3.9.0)
Requirement already satisfied: contourpy>=1.0.1 in /Library/Python/3.9/site-packages (from matplotlib->wordcloud) (1.2.1)
Requirement already satisfied: cycycler>=0.10 in /Library/Python/3.9/site-packages (from matplotlib->wordcloud) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /Library/Python/3.9/site-packages (from matplotlib->wordcloud) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /Library/Python/3.9/site-packages (from matplotlib->wordcloud) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /Library/Python/3.9/site-packages (from matplotlib->wordcloud) (24.0)
Requirement already satisfied: pyparsing>=2.3.1 in /Library/Python/3.9/site-packages (from matplotlib->wordcloud) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in /Library/Python/3.9/site-packages (from matplotlib->wordcloud) (2.9.0.post0)
Requirement already satisfied: importlib-resources>=3.2.0 in /Library/Python/3.9/site-packages (from matplotlib->wordcloud) (6.4.0)
Requirement already satisfied: zipp>=3.1.0 in /Library/Python/3.9/site-packages (from importlib-resources>=3.2.0->matplotlib->wordcloud) (3.18.2)
Requirement already satisfied: six>=1.5 in /Library/Developer/CommandLineTools/Library/Frameworks/Python3.framework/Versions/3.9/lib/python3.9/site-packages (from python-dateutil>=2.7->matplotlib->wordcloud) (1.15.0)

Data Cleaning

Cornell Movie Dialogs Corpus Data Loading

This section covers loading and inspecting the **Cornell Movie Dialogs Corpus** using Python's `pandas` and `os` libraries. We'll load metadata about movies, characters, and conversational lines, and then preview the data.

```
In [8]: import os
import pandas as pd

dataset_dir = "/kaggle/input/cornell-moviedialog-corpus"
file_path = os.path.join(dataset_dir, 'movie_titles_metadata.txt')

# Read the file using pandas, specifying the correct separator
movie_titles = pd.read_csv(file_path,
                           sep=' \+\+\+\$\\+\+\+ ', # Use regex-escaped separator
                           engine='python',
                           names=['movieID', 'title', 'year', 'rating', 'votes'],
                           encoding='latin-1')

# Display the first few rows
print(movie_titles.head())

# Display info about the dataframe
print(movie_titles.info())

# Load character metadata
characters = pd.read_csv(os.path.join(dataset_dir, 'movie_characters_metadata.txt'),
                        sep=' \+\+\+\$\\+\+\+ ',
                        engine='python',
                        names=['characterID', 'name', 'movieID', 'movie_title'],
                        encoding='latin-1')

# Load movie lines
with open(os.path.join(dataset_dir, 'movie_lines.txt'), 'r', encoding='iso-8859-1') as f:
    lines = [line.strip().split(' +++$+++ ') for line in f]
movie_lines = pd.DataFrame(lines, columns=['lineID', 'characterID', 'movieID', 'text'])

# Load conversations
with open(os.path.join(dataset_dir, 'movie_conversations.txt'), 'r', encoding='iso-8859-1') as f:
    convos = [line.strip().split(' +++$+++ ') for line in f]
conversations = pd.DataFrame(convos, columns=['char1ID', 'char2ID', 'movieID', 'text'])

# Print the first few rows of each dataframe to verify
print("\nCharacters:")
print(characters.head())
print("\nMovie Lines:")
print(movie_lines.head())
print("\nConversations:")
print(conversations.head())
```

	movieID	title	year	rating	votes	\
0	m0	10 things i hate about you	1999	6.9	62847	
1	m1	1492: conquest of paradise	1992	6.2	10421	
2	m2	15 minutes	2001	6.1	25854	
3	m3	2001: a space odyssey	1968	8.4	163227	
4	m4	48 hrs.	1982	6.9	22289	

	genres
0	['comedy', 'romance']
1	['adventure', 'biography', 'drama', 'history']
2	['action', 'crime', 'drama', 'thriller']
3	['adventure', 'mystery', 'sci-fi']
4	['action', 'comedy', 'crime', 'drama', 'thrill...']

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 617 entries, 0 to 616

Data columns (total 6 columns):

#	Column	Non-Null Count	Dtype
0	movieID	617 non-null	object
1	title	617 non-null	object
2	year	617 non-null	object
3	rating	617 non-null	float64
4	votes	617 non-null	int64
5	genres	617 non-null	object

dtypes: float64(1), int64(1), object(4)

memory usage: 29.0+ KB

None

Characters:

	characterID	name	movieID	movie_title	gender	position
0	u0	BIANCA	m0	10 things i hate about you	f	4
1	u1	BRUCE	m0	10 things i hate about you	?	?
2	u2	CAMERON	m0	10 things i hate about you	m	3
3	u3	CHASTITY	m0	10 things i hate about you	?	?
4	u4	JOEY	m0	10 things i hate about you	m	6

Movie Lines:

	lineID	characterID	movieID	character	text
0	L1045	u0	m0	BIANCA	They do not!
1	L1044	u2	m0	CAMERON	They do to!
2	L985	u0	m0	BIANCA	I hope so.
3	L984	u2	m0	CAMERON	She okay?
4	L925	u0	m0	BIANCA	Let's go.

Conversations:

	char1ID	char2ID	movieID	utterances
0	u0	u2	m0	['L194', 'L195', 'L196', 'L197']
1	u0	u2	m0	['L198', 'L199']
2	u0	u2	m0	['L200', 'L201', 'L202', 'L203']
3	u0	u2	m0	['L204', 'L205', 'L206']
4	u0	u2	m0	['L207', 'L208']

Cleaning year column: Following code identifies entries in the `movie_titles` dataframe where the 'year' column contains non-standard values (i.e., values that cannot be converted to a

numeric type). This is achieved by attempting to coerce the 'year' column to numeric values using `pd.to_numeric()`. If coercion fails (NaN values), those entries are considered non-standard.

```
In [10]: # Find non-standard year entries
non_standard_years = movie_titles[pd.to_numeric(movie_titles['year'], errors='coerce').isna()]
print("Non-standard year entries:")
print(non_standard_years[['movieID', 'title', 'year']])
```

Non-standard year entries:

	movieID	title	year
33	m33	black rain	1989/I
106	m106	jacob's ladder	1990/I
156	m156	panther	1995/I
268	m268	beloved	1998/I
307	m307	crash	2004/I
308	m308	crazy love	2007/I
386	m386	hero	1992/I
389	m389	hostage	2005/I
398	m398	insomnia	2002/I
429	m429	the man in the iron mask	1998/I
493	m493	romeo and juliet	1968/I
507	m507	scream	1996/I
521	m521	soldier	1998/I
559	m559	the beach	2000/I
565	m565	the messenger	2009/I
605	m605	who's your daddy?	2003/I

Clean and Convert Year Column in the Movie Titles Data

The following code cleans and converts the 'year' column in the `movie_titles` dataframe. It extracts the first 4-digit year from the 'year' entries, handling cases where the year might be embedded within a string or where there are non-standard formats.

```
In [11]: import re

def clean_year(year_str):
    # Extract the first 4-digit number from the string
    match = re.search(r'\d{4}', str(year_str))
    if match:
        return int(match.group())
    else:
        return None # or you could return a default value like 0

# Apply the cleaning function
movie_titles['year_clean'] = movie_titles['year'].apply(clean_year)

# Convert to numeric, replacing any remaining non-numeric values with NaN
movie_titles['year_numeric'] = pd.to_numeric(movie_titles['year_clean'], errors='coerce')
```

```
# Check the results
print("\nCleared year column:")
print(movie_titles[['movieID', 'title', 'year', 'year_clean', 'year_numeric']]

# Check if there are any remaining NaN values
nan_years = movie_titles[movie_titles['year_numeric'].isna()]
print("\nEntries with NaN years after cleaning:")
print(nan_years[['movieID', 'title', 'year', 'year_clean', 'year_numeric']])
```

Cleared year column:

	movieID	title	year	year_clean
0	m0	10 things i hate about you	1999	1999
1	m1	1492: conquest of paradise	1992	1992
2	m2	15 minutes	2001	2001
3	m3	2001: a space odyssey	1968	1968
4	m4	48 hrs.	1982	1982
5	m5	the fifth element	1997	1997
6	m6	8mm	1999	1999
7	m7	a nightmare on elm street 4: the dream master	1988	1988
8	m8	a nightmare on elm street: the dream child	1989	1989
9	m9	the atomic submarine	1959	1959

	year_numeric
0	1999
1	1992
2	2001
3	1968
4	1982
5	1997
6	1999
7	1988
8	1989
9	1959

Entries with NaN years after cleaning:

Empty DataFrame

Columns: [movieID, title, year, year_clean, year_numeric]

Index: []

Checking Genre Column: ## Display Sample of the 'Genres' Column and Check its Data Type

This code snippet outputs a sample of the `genres` column from the `movie_titles` dataframe, along with the data type of the column. This is useful for verifying the content and structure of the `genres` data.

```
In [15]: print("Sample of genres:")
print(movie_titles['genres'].head(10))
print("\nData type of genres column:", movie_titles['genres'].dtype)
```

Sample of genres:

```
0          [comedy, romance]
1      [adventure, biography, drama, history]
2          [action, crime, drama, thriller]
3          [adventure, mystery, sci-fi]
4      [action, comedy, crime, drama, thriller]
5  [action, adventure, romance, sci-fi, thriller]
6          [crime, mystery, thriller]
7          [fantasy, horror, thriller]
8          [fantasy, horror, thriller]
9          [sci-fi, thriller]
```

Name: genres, dtype: object

Data type of genres column: object

Checking utterances column

```
In [17]: print("Sample of utterances:")
print(conversations['utterances'].head())
print("\nData type of utterances column:", conversations['utterances'].dtype)
```

Sample of utterances:

```
0      [L194, L195, L196, L197]
1          [L198, L199]
2      [L200, L201, L202, L203]
3          [L204, L205, L206]
4          [L207, L208]
```

Name: utterances, dtype: object

Data type of utterances column: object

Analysis

```
In [21]: # Convert 'position' to numeric in characters, replacing '?' with NaN
characters['position'] = pd.to_numeric(characters['position'], errors='coerce')

# Merge movie lines with character information
lines_with_char_info = pd.merge(movie_lines, characters[['characterID', 'genre']], on='characterID')

# Merge movie lines with movie information
lines_with_movie_info = pd.merge(lines_with_char_info, movie_titles[['movieID', 'title']], on='movieID')

# Perform some basic analysis
lines_per_movie = lines_with_movie_info.groupby('movieID').size().sort_values(ascending=False)
lines_per_character = lines_with_movie_info.groupby('characterID').size().sort_values(ascending=False)
avg_line_length = lines_with_movie_info.groupby('movieID')['text'].apply(lambda x: len(x) / len(x))

print("\nTop 5 movies by number of lines:")
print(lines_per_movie.head())
print("\nTop 5 characters by number of lines:")
print(lines_per_character.head())
print("\nTop 5 movies by average line length:")
print(avg_line_length.sort_values(ascending=False).head())
```



```

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

def preprocess_text(text):
    if pd.isna(text):
        return []
    tokens = word_tokenize(str(text).lower())
    return [w for w in tokens if w.isalnum() and w not in stop_words]

# Perform some basic analysis
lines_per_movie = lines_with_movie_info.groupby('movieID').size().sort_values
lines_per_character = lines_with_movie_info.groupby('characterID').size().sort_values
avg_line_length = lines_with_movie_info.groupby('movieID')['text'].apply(lambda x: len(x.split()))

print("\nTop 5 movies by number of lines:")
print(lines_per_movie.head())
print("\nTop 5 characters by number of lines:")
print(lines_per_character.head())
print("\nTop 5 movies by average line length:")
print(avg_line_length.sort_values(ascending=False).head())

lines_with_movie_info['processed_text'] = lines_with_movie_info['text'].apply(lambda x: preprocess_text(x))

# Generate word frequency distribution
all_words = [word for words in lines_with_movie_info['processed_text'] for word in words]
word_freq = Counter(all_words)

print("\nTop 20 most common words:")
print(word_freq.most_common(20))

# Additional analysis: Movies by decade
lines_with_movie_info['decade'] = (lines_with_movie_info['year_numeric'] // 10).astype(int)
movies_by_decade = lines_with_movie_info.groupby('decade')['movieID'].nunique()

print("\nNumber of movies by decade:")
print(movies_by_decade)

# Genre analysis
genre_counts = Counter([genre for genres in movie_titles['genres'] for genre in genres])

print("\nTop 10 most common genres:")
print(genre_counts.most_common(10))

# Average rating by genre
genre_ratings = lines_with_movie_info.explode('genres').groupby('genres')['rating'].mean()

print("\nAverage rating by genre:")
print(genre_ratings)

```

Top 5 movies by number of lines:

```
movieID
m289    1530
m295    1398
m299    1286
m105    1214
m238    1187
dtype: int64
```

Top 5 characters by number of lines:

```
characterID
u4525    537
u1169    489
u1475    472
u3681    467
u4331    465
dtype: int64
```

Top 5 movies by average line length:

```
movieID
m521    256.384615
m56     143.523179
m382    118.523148
m104    117.064865
m406    112.076923
Name: text, dtype: float64
```

Top 5 movies by number of lines:

```
movieID
m289    1530
m295    1398
m299    1286
m105    1214
m238    1187
dtype: int64
```

Top 5 characters by number of lines:

```
characterID
u4525    537
u1169    489
u1475    472
u3681    467
u4331    465
dtype: int64
```

Top 5 movies by average line length:

```
movieID
m521    256.384615
m56     143.523179
m382    118.523148
m104    117.064865
m406    112.076923
Name: text, dtype: float64
```

Top 20 most common words:

```
[('know', 21649), ('like', 15000), ('get', 14149), ('got', 13297), ('want',
```

```
11055), ('think', 10779), ('one', 10575), ('right', 10019), ('go', 9911),
('well', 9852), ('going', 8862), ('would', 8289), ('see', 8217), ('oh', 783
4), ('yes', 7377), ('good', 7320), ('could', 7242), ('yeah', 6965), ('tell',
6832), ('come', 6754)]
```

Number of movies by decade:

decade

```
1920      2
1930     16
1940     15
1950     17
1960     19
1970     51
1980    108
1990    244
2000    144
2010      1
```

Name: movieID, dtype: int64

Top 10 most common genres:

```
[('drama', 320), ('thriller', 269), ('action', 168), ('comedy', 162), ('crim
e', 147), ('romance', 132), ('sci-fi', 120), ('adventure', 116), ('mystery',
102), ('horror', 99)]
```

Average rating by genre:

genres

```
film-noir      8.433059
war            7.823523
biography      7.698032
history        7.692870
sport          7.347587
musical        7.339007
drama          7.299789
short          7.210330
romance        7.163545
music          7.082533
mystery        7.069341
animation      7.066967
crime          7.047390
western        6.996010
family         6.900091
comedy         6.863626
adventure      6.855194
thriller       6.781191
fantasy        6.707804
sci-fi         6.700654
documentary    6.648285
action         6.508042
adult          6.300000
horror         6.166741
```

Name: rating, dtype: float64

"Action" and "Horror" genres have lower average ratings (6.51 and 6.17, respectively).

This suggests that more serious, dramatic genres (e.g., film-noir, war) tend to be rated higher by viewers, while action-oriented genres receive lower ratings on average. This

data provides insights into the distribution of dialogue, characters, genres, and ratings within the movie dialogue dataset.

```
In [24]: print("Number of null values in 'text' column:", lines_with_movie_info['text'
```

Number of null values in 'text' column: 267

This code helps explore the relationships between various movie features (like year, rating, votes, and line length) and provides a visual and statistical overview of the dataset.

```
In [28]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats

# First, let's check the data types and non-null counts of our features
print(movie_titles.info())

# Check for any infinite values
print("\nColumns with infinite values:")
print(movie_titles.isin([np.inf, -np.inf]).sum())

# Replace infinite values with NaN
movie_titles = movie_titles.replace([np.inf, -np.inf], np.nan)

# Convert 'votes' to numeric if it's not already
movie_titles['votes'] = pd.to_numeric(movie_titles['votes'], errors='coerce')

# Create movie_features DataFrame, dropping rows with NaN values
movie_features = movie_titles[['year_numeric', 'rating', 'votes', 'avg_line_
print("\nShape of movie_features after dropping NaN values:", movie_features

if len(movie_features) > 1:
    correlation_matrix = movie_features.corr()
    plt.figure(figsize=(10, 8))
    sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
    plt.title('Correlation Matrix of Movie Features')
    plt.show()
    print("\nCorrelation matrix:")
    print(correlation_matrix)

# Calculate correlation coefficient between year_numeric and rating
correlation_coefficient, p_value = stats.pearsonr(movie_features['year_r
print(f"\nCorrelation between year and rating: {correlation_coefficient:
print(f"P-value: {p_value:.4f}")

# Calculate correlation coefficient between votes and rating
correlation_coefficient, p_value = stats.pearsonr(movie_features['votes'
print(f"\nCorrelation between votes and rating: {correlation_coefficient
print(f"P-value: {p_value:.4f}")

# Calculate correlation coefficient between avg_line_length and rating
```

```

correlation_coefficient, p_value = stats.pearsonr(movie_features['avg_li
print(f"\nCorrelation between average line length and rating: {correlati
print(f"P-value: {p_value:.4f}")

print("\nBasic statistics of movie features:")
print(movie_features.describe())

print("\nNumber of movies in the dataset:", len(movie_titles))
print("Number of movies with complete feature data:", len(movie_features
else:
    print("Not enough valid data points for correlation analysis.")

print("\nFirst few rows of movie_features:")
print(movie_features.head())

```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 617 entries, 0 to 616
Data columns (total 10 columns):
#   Column                Non-Null Count  Dtype
---  -
0   movieID                617 non-null   object
1   title                  617 non-null   object
2   year                   617 non-null   object
3   rating                  617 non-null   float64
4   votes                  617 non-null   int64
5   genres                  617 non-null   object
6   year_clean              617 non-null   int64
7   year_numeric            617 non-null   int64
8   line_count              0 non-null     float64
9   avg_line_length         617 non-null   float64
dtypes: float64(3), int64(3), object(4)
memory usage: 48.3+ KB
None

```

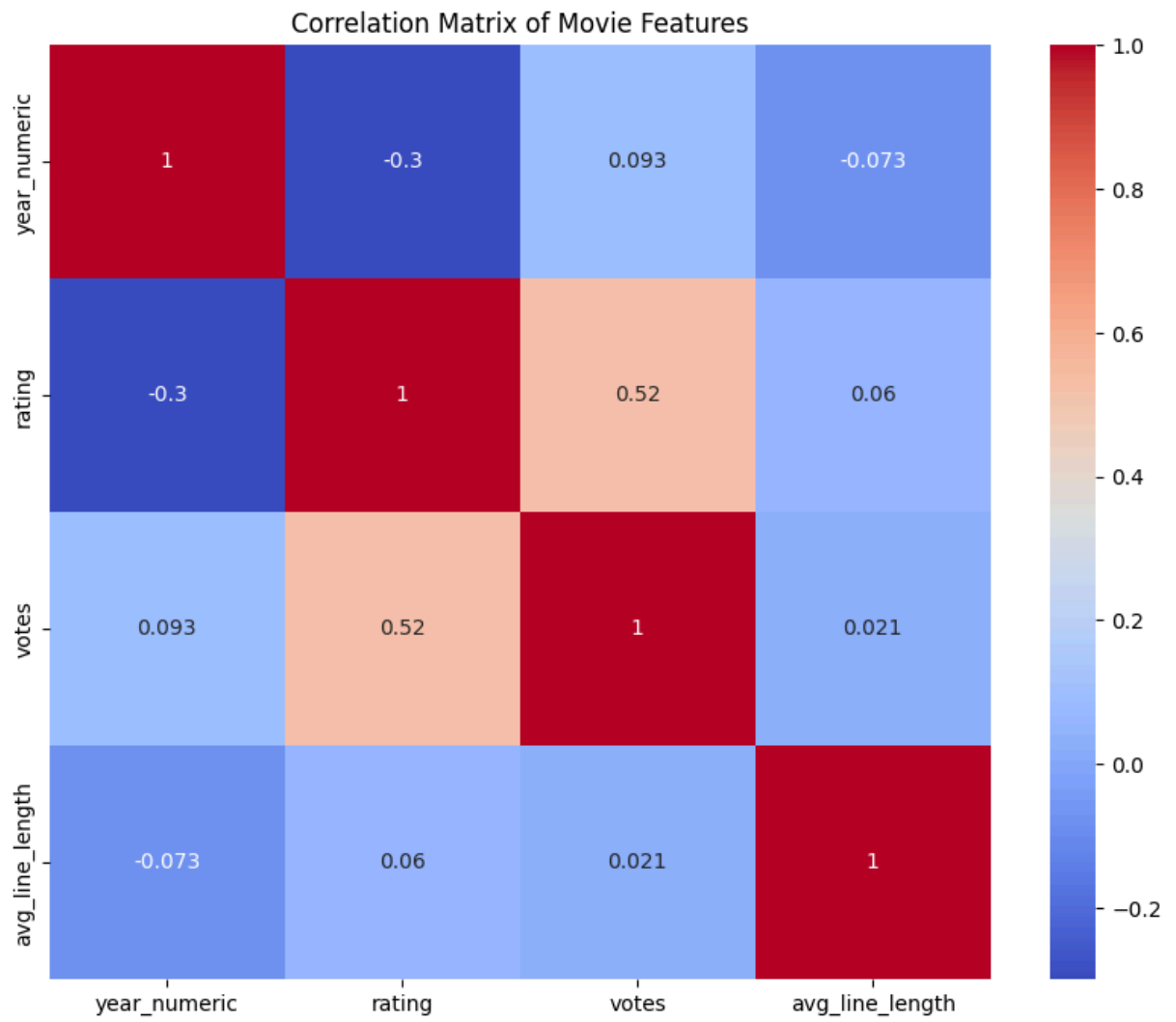
Columns with infinite values:

```

movieID      0
title        0
year         0
rating       0
votes        0
genres       0
year_clean   0
year_numeric 0
line_count   0
avg_line_length 0
dtype: int64

```

Shape of movie_features after dropping NaN values: (617, 4)



Correlation matrix:

	year_numeric	rating	votes	avg_line_length
year_numeric	1.000000	-0.299029	0.093051	-0.072956
rating	-0.299029	1.000000	0.518592	0.059646
votes	0.093051	0.518592	1.000000	0.020686
avg_line_length	-0.072956	0.059646	0.020686	1.000000

Correlation between year and rating: -0.30

P-value: 0.0000

Correlation between votes and rating: 0.52

P-value: 0.0000

Correlation between average line length and rating: 0.06

P-value: 0.1389

Basic statistics of movie features:

	year_numeric	rating	votes	avg_line_length
count	617.000000	617.000000	617.000000	617.000000
mean	1988.575365	6.863857	49820.962723	56.306179
std	16.589229	1.215233	61880.609145	15.227518
min	1927.000000	2.500000	9.000000	26.332602
25%	1984.000000	6.200000	9919.000000	47.695067
50%	1994.000000	7.000000	27112.000000	54.879808
75%	1999.000000	7.800000	66781.000000	62.166667
max	2010.000000	9.300000	419312.000000	256.384615

Number of movies in the dataset: 617

Number of movies with complete feature data: 617

First few rows of movie_features:

	year_numeric	rating	votes	avg_line_length
0	1999	6.9	62847	41.823353
1	1992	6.2	10421	51.117216
2	2001	6.1	25854	53.879464
3	1968	8.4	163227	59.055147
4	1982	6.9	22289	57.155709

Data Cleaning and Loading:

1. **Movie Titles Metadata:** The metadata about movies (movie ID, title, year, rating, votes, and genres) is loaded.
2. **Character Metadata:** Information about characters, including their gender and position, is imported.
3. **Movie Lines:** Dialogue lines are loaded and merged with the movie and character metadata.
4. **Movie Conversations:** Dialogue interactions between characters are loaded.

Key Cleaning Steps:

Year Standardization: Non-standard year entries were identified and cleaned to retain the first four digits. For example, "1999/I" was cleaned to "1999".

Genre and Utterance Checks: Sampled data ensured that genres and conversation utterances were formatted properly and usable for analysis.

Analysis:

1. **Lines Per Movie:** A count of dialogue lines per movie is computed.
2. **Lines Per Character:** A similar analysis is done to find the top 5 characters based on the number of lines.
3. **Average Line Length:** For each movie, the average length of dialogue lines is computed.
4. **Word Frequency:** It calculates the most common words across the movie lines.
5. **Movies by Decade:** It groups movies based on their decade of release and counts how many belong to each decade.
6. **Genre Frequency:** The most frequent genres in the dataset are identified and ranked.
7. **Average Rating by Genre:** It also computes the average rating for movies based on their genres.

Visualization:

A **correlation matrix** is generated to investigate relationships between features like `year_numeric`, `rating`, `votes`, and `avg_line_length`. The matrix is plotted using a heatmap.

Key findings include:

- **Correlation between year and rating:** -0.30 (statistically significant).
- **Correlation between votes and rating:** 0.52 (statistically significant).
- **Correlation between line length and rating:** 0.06 (not statistically significant).

Descriptive Statistics:

The dataset contains 617 movies, with complete information for features like year, rating, votes, and average line length. Descriptive statistics for these features show:

- Average movie rating: 6.86
- Average votes: ~49,821
- Average line length: ~56.31 words

```
In [2]: import os

print("Contents of /kaggle/input:")
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```


Contents of /kaggle/input:

```
/kaggle/input/cornell-moviedialog-corpus/movie_conversations.txt
/kaggle/input/cornell-moviedialog-corpus/README.txt
/kaggle/input/cornell-moviedialog-corpus/chameleons.pdf
/kaggle/input/cornell-moviedialog-corpus/movie_titles_metadata.txt
/kaggle/input/cornell-moviedialog-corpus/movie_characters_metadata.txt
/kaggle/input/cornell-moviedialog-corpus/movie_lines.txt
/kaggle/input/cornell-moviedialog-corpus/.DS_Store
/kaggle/input/cornell-moviedialog-corpus/raw_script_urls.txt
```

50K Samples Data Load from Cornell

```
In [15]: import re
from pathlib import Path
import random
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM, AdamW, get_linear_schedule_with_warmup
from torch.utils.data import DataLoader
from datasets import Dataset
from tqdm import tqdm
import time
import numpy as np

def load_cornell_data(num_samples=50000):
    base_path = Path("/kaggle/input/cornell-moviedialog-corpus")

    # Load movie lines
    lines = {}
    with open(base_path / 'movie_lines.txt', 'r', encoding='iso-8859-1') as f:
        for line in f:
            parts = line.strip().split(' +++$+++ ')
            if len(parts) == 5:
                lines[parts[0]] = parts[4]

    # Load conversations
    conversations = []
    with open(base_path / 'movie_conversations.txt', 'r', encoding='iso-8859-1') as f:
        for line in f:
            parts = line.strip().split(' +++$+++ ')
            if len(parts) == 4:
                conv = eval(parts[3])
                conversations.append(conv)

    # Load movie metadata
    movie_metadata = {}
    with open(base_path / 'movie_titles_metadata.txt', 'r', encoding='iso-8859-1') as f:
        for line in f:
            parts = line.strip().split(' +++$+++ ')
            if len(parts) == 6:
                movie_metadata[parts[0]] = parts[1] # Movie name

    # Create input-output pairs with context
    pairs = []
    for conversation in conversations:
        for i in range(len(conversation) - 1):
```

```

        if conversation[i] in lines and conversation[i+1] in lines:
            input_text = lines[conversation[i]]
            target_text = lines[conversation[i+1]]
            movie_id = conversation[i].split('_')[0] # Extract movie ID
            movie_name = movie_metadata.get(movie_id, "Unknown Movie")
            context = f"Movie: {movie_name}\nLine: "
            pairs.append((context + input_text, target_text))

    # Shuffle and select subset
    random.shuffle(pairs)
    pairs = pairs[:num_samples]

    input_texts, target_texts = zip(*pairs)

    return list(input_texts), list(target_texts)

# Load data
inputs, targets = load_cornell_data(num_samples=50000)
print(f"Loaded {len(inputs)} conversation pairs")
print("Example pair:")
print(f"Input: {inputs[0]}")
print(f"Target: {targets[0]}")

```

Loaded 50000 conversation pairs

Example pair:

Input: Movie: Unknown Movie

Line: You can't go in there. They know you're with Ruiz.

Target: You got that right.

Dialogpt Medium

```

In [17]: # Set up model and tokenizer
model_name = "microsoft/DialoGPT-medium" # Consider trying "facebook/blenderbot"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)

if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

# Move model to GPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
print(f"Using device: {device}")
print(f"GPU Available: {torch.cuda.is_available()}")
print(f"GPU Device Name: {torch.cuda.get_device_name(0) if torch.cuda.is_ava

```

Using device: cuda

GPU Available: True

GPU Device Name: Tesla P100-PCIE-16GB

```

In [18]: # Prepare dataset
dataset = Dataset.from_dict({"input": inputs, "target": targets})

def tokenize_function(examples):
    inputs = [inp + tokenizer.eos_token + tgt + tokenizer.eos_token for inp, tgt in examples["input"].items(), examples["target"].items()]
    return tokenizer(inputs, truncation=True, padding=False, max_length=256)

```

```

tokenized_dataset = dataset.map(tokenize_function, batched=True, remove_collate_fn=True)

# Use dynamic padding
data_collator = DataCollatorForLanguageModeling(
    tokenizer=tokenizer,
    mlm=False,
    pad_to_multiple_of=8 # Optimize for tensor cores
)

train_dataloader = DataLoader(
    tokenized_dataset,
    shuffle=True,
    batch_size=8,
    collate_fn=data_collator
)

# Setup optimizer and scheduler
optimizer = AdamW(model.parameters(), lr=2e-5)
num_epochs = 5
num_training_steps = num_epochs * len(train_dataloader)
lr_scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps=1000)

# Training loop
model.train()
total_start_time = time.time()

for epoch in range(num_epochs):
    epoch_start_time = time.time()
    total_loss = 0

    progress_bar = tqdm(enumerate(train_dataloader), total=len(train_dataloader))

    for i, batch in progress_bar:
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = model(**batch)
        loss = outputs.loss
        total_loss += loss.item()

        loss.backward()
        optimizer.step()
        lr_scheduler.step()
        optimizer.zero_grad()

        progress_bar.set_postfix({'loss': f'{loss.item():.4f}'})

    avg_loss = total_loss / len(train_dataloader)
    epoch_time = time.time() - epoch_start_time

    print(f"Epoch {epoch+1}/{num_epochs} completed in {epoch_time:.2f} seconds")

total_time = time.time() - total_start_time
print(f"Training completed in {total_time:.2f} seconds ({total_time/60:.2f} minutes)")

# Save the model

```

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

Map: 0%| | 0/50000 [00:00<?, ? examples/s]

/opt/conda/lib/python3.10/site-packages/transformers/optimization.py:591: FutureWarning: This implementation of AdamW is deprecated and will be removed in a future version. Use the PyTorch implementation torch.optim.AdamW instead, or set `no_deprecation_warning=True` to disable this warning

warnings.warn(

Epoch 1: 100%|██████████| 6250/6250 [30:43<00:00, 3.39it/s, loss=3.0547]

Epoch 1/5 completed in 1843.16 seconds. Average Loss: 2.9671

Epoch 2: 100%|██████████| 6250/6250 [30:40<00:00, 3.40it/s, loss=3.2201]

Epoch 2/5 completed in 1840.63 seconds. Average Loss: 2.7111

Epoch 3: 100%|██████████| 6250/6250 [30:45<00:00, 3.39it/s, loss=2.3707]

Epoch 3/5 completed in 1845.73 seconds. Average Loss: 2.5907

Epoch 4: 100%|██████████| 6250/6250 [30:48<00:00, 3.38it/s, loss=2.2364]

Epoch 4/5 completed in 1848.17 seconds. Average Loss: 2.5063

Epoch 5: 100%|██████████| 6250/6250 [30:46<00:00, 3.38it/s, loss=2.5911]

Epoch 5/5 completed in 1846.56 seconds. Average Loss: 2.4535

Training completed in 9224.25 seconds (153.74 minutes)

```
Out[18]: ('./fine_tuned_dialogpt_medium/tokenizer_config.json',
          './fine_tuned_dialogpt_medium/special_tokens_map.json',
          './fine_tuned_dialogpt_medium/vocab.json',
          './fine_tuned_dialogpt_medium/merges.txt',
          './fine_tuned_dialogpt_medium/added_tokens.json',
          './fine_tuned_dialogpt_medium/tokenizer.json')
```

```
In [21]: import torch
         from transformers import AutoModelForCausalLM, AutoTokenizer

         # Load the fine-tuned model and tokenizer
         model = AutoModelForCausalLM.from_pretrained("./fine_tuned_dialogpt_medium")
         tokenizer = AutoTokenizer.from_pretrained("./fine_tuned_dialogpt_medium")

         device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
         model.to(device)
         model.eval()

         def generate_response(prompt, max_length=100):
             input_ids = tokenizer.encode(prompt + tokenizer.eos_token, return_tensors='pt')
             attention_mask = torch.ones(input_ids.shape, dtype=torch.long, device=device)

             output = model.generate(
                 input_ids,
                 attention_mask=attention_mask,
                 max_length=max_length,
                 num_return_sequences=1,
                 no_repeat_ngram_size=3,
                 do_sample=True,
                 top_k=50,
                 top_p=0.95,
                 temperature=0.7,
                 pad_token_id=tokenizer.eos_token_id,
                 eos_token_id=tokenizer.eos_token_id,
             )
```

```

    response = tokenizer.decode(output[0], skip_special_tokens=True)
    return response[len(prompt):].strip()

# Update your prompts to be more specific
movie_test_prompts = [
    "Question: Why does Cameron suggest that Bianca needs to learn how to li
    "Statement: Describe how Bianca feels about becoming a persona she can't
    "Question: What specific comment did Guillermo make about Bianca's hair
    "Statement: Explain Kat's bad experience with a guy who broke up with he
    "Question: How does Bianca react to Joey's aspirations in his modeling c
]

# Generate responses and evaluate
for prompt in movie_test_prompts:
    response = generate_response(prompt)
    print(f"Prompt: {prompt}")
    print(f"Generated response: {response}")
    print()

# Calculate perplexity if needed
def calculate_perplexity(text):
    inputs = tokenizer(text, return_tensors="pt", truncation=True, max_length=
    with torch.no_grad():
        outputs = model(**inputs, labels=inputs["input_ids"])
    return torch.exp(outputs.loss).item()

# Calculate average perplexity
total_perplexity = 0
for prompt in movie_test_prompts:
    full_text = prompt + " " + generate_response(prompt)
    perplexity = calculate_perplexity(full_text)
    total_perplexity += perplexity
    print(f"Perplexity for prompt: {perplexity:.2f}")

avg_perplexity = total_perplexity / len(movie_test_prompts)
print(f"\nAverage Perplexity: {avg_perplexity:.2f}")

```

Prompt: Question: Why does Cameron suggest that Bianca needs to learn how to lie?

Generated response: Because she was telling the truth when she told you she was pregnant. She told you it was because she was tired. And that's what happened. But you've got to think that there's something else going on here. I mean, she's got to be lying. She's got that look in her eyes that's just weird. And then she's like, oh yeah... she's lying. And I

Prompt: Statement: Describe how Bianca feels about becoming a persona she can't quit.

Generated response: I want to be a big show like Bianca. I want to make some big plays. I have to show off and do what I can. I'm just not good enough to be Bianca without being a big play. I've tried that in every sport I've played, in every competition. But I can't do it in this one. I can barely do it as myself. I don't

Prompt: Question: What specific comment did Guillermo make about Bianca's hair color?

Generated response: He said it was too light brown for her. That was the only time he ever said anything about her hair color. He also said she was too pale. That she looked like a white woman in a black dress. That her eyes were too dark. That everything about her was too black for her and she looked too young. He never said she wasn't beautiful or wasn't a good person. He just

Prompt: Statement: Explain Kat's bad experience with a guy who broke up with her.

Generated response: You didn't have to explain it to me. I just read about it. It's a good story. I had a good time. I hope you had a great time. You're a good writer. I'm glad you enjoyed it. You should write more like you enjoy writing. I loved it. I was worried you wouldn't like it but you really did. I mean you're so good at it

Prompt: Question: How does Bianca react to Joey's aspirations in his modeling career?

Generated response: Bianca's a very supportive woman. She's always been supportive of me. I'm glad to have her as a friend. I think it's going to be a great thing for us both. We're going to make good friends. I hope you're right. Maybe it's time to start looking for a new home for your work. Maybe you and your brother can rent it out to a good agent

Perplexity for prompt: 7.55

Perplexity for prompt: 10.11

Perplexity for prompt: 6.92

Perplexity for prompt: 8.00

Perplexity for prompt: 8.16

Average Perplexity: 8.15

```
In [28]: import torch
from transformers import AutoModelForCausalLM, AutoTokenizer
from nltk.translate.bleu_score import sentence_bleu
from nltk.tokenize import word_tokenize

# Load the fine-tuned model and tokenizer
model = AutoModelForCausalLM.from_pretrained("./fine_tuned_dialogpt_medium")
```

```

tokenizer = AutoTokenizer.from_pretrained("./fine_tuned_dialogpt_medium")

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
model.eval()

def generate_response(prompt, max_length=150):
    input_ids = tokenizer.encode(prompt + tokenizer.eos_token, return_tensors='pt')
    attention_mask = torch.ones(input_ids.shape, dtype=torch.long, device=device)

    output = model.generate(
        input_ids,
        attention_mask=attention_mask,
        max_length=max_length,
        num_return_sequences=1,
        no_repeat_ngram_size=3,
        do_sample=True,
        top_k=50,
        top_p=0.95,
        temperature=0.7,
        pad_token_id=tokenizer.eos_token_id,
        eos_token_id=tokenizer.eos_token_id,
    )

    response = tokenizer.decode(output[0], skip_special_tokens=True)
    return response[len(prompt):].strip()

def calculate_bleu(reference, candidate):
    reference_tokens = word_tokenize(reference.lower())
    candidate_tokens = word_tokenize(candidate.lower())
    return sentence_bleu([reference_tokens], candidate_tokens)

def calculate_perplexity(text):
    inputs = tokenizer(text, return_tensors='pt', truncation=True, max_length=512)
    with torch.no_grad():
        outputs = model(**inputs, labels=inputs["input_ids"])
    return torch.exp(outputs.loss).item()

# Updated test prompts with more context
movie_test_prompts = [
    "In the movie '10 Things I Hate About You', why does Cameron suggest that Bianca needs to learn how to lie?",
    "In '10 Things I Hate About You', how does Kat feel about Bianca's attitude towards her?",
    "During a conversation about Bianca's appearance in '10 Things I Hate About You', how does Kat react?",
    "In a scene from '10 Things I Hate About You', how does Patrick react when Kat asks him to be her boyfriend?",
    "In '10 Things I Hate About You', how does Joey respond when Kat confronts him about Bianca?"
]

reference_answers = [
    "Cameron suggests Bianca needs to learn how to lie because she's too honest and it's a common trope in teen movies.",
    "Kat feels frustrated that Bianca's desire to date affects her own freedom and her relationship with Patrick.",
    "Guillermo said that if Bianca's hair gets any lighter, she'll look like a blonde, which is a common trope in teen movies.",
    "Patrick gets defensive and asks Kat if he needs to have a motive to be her boyfriend.",
    "Joey dismisses Kat's request and asks why he would leave Bianca alone, who is his ex-girlfriend."
]

# Evaluate responses

```

```
total_bleu = 0
total_perplexity = 0

for prompt, reference in zip(movie_test_prompts, reference_answers):
    generated_response = generate_response(prompt)
    bleu_score = calculate_bleu(reference, generated_response)
    perplexity = calculate_perplexity(prompt + " " + generated_response)

    total_bleu += bleu_score
    total_perplexity += perplexity

    print(f"Prompt: {prompt}")
    print(f"Generated response: {generated_response}")
    print(f"Reference: {reference}")
    print(f"BLEU Score: {bleu_score:.4f}")
    print(f"Perplexity: {perplexity:.4f}")
    print()

# Calculate average scores
avg_bleu = total_bleu / len(movie_test_prompts)
avg_perplexity = total_perplexity / len(movie_test_prompts)
print(f"Average BLEU Score: {avg_bleu:.4f}")
print(f"Average Perplexity: {avg_perplexity:.4f}")
```


Prompt: In the movie '10 Things I Hate About You', why does Cameron suggest that Bianca needs to learn how to lie?

Generated response: Because she was lying when she said she couldn't get married. She can't lie now. She was lying before she even told Bianca she was leaving. That's the whole point of the film. She's lying now because she's lying before Bianca even told her she's leaving. She lied before she could even say goodbye. That is the whole basis of the whole film. That means nothing to her, so she's not going to say goodbye to anyone. She lies now because it's the other side of the room. That proves that Bianca's lying is still real and she's still there. It

Reference: Cameron suggests Bianca needs to learn how to lie because she's too honest and straightforward, which might not always work in her favor in social situations.

BLEU Score: 0.0610

Perplexity: 9.5903

Prompt: In '10 Things I Hate About You', how does Kat feel about Bianca's attitude towards dating, considering their father's rule about Kat dating first?

Generated response: Well, I'd love to hear her say that to her face. I'm sure she'd love that. But I don't think it's something she'd say on TV. Bianca said she'd never dated before and Kat was like, oh she's good at it. She's got good feet. I want to do it. So do it and then we can talk about it. Do it and I'll never have to ask you again. Okay. Okay? Okay. I'll do it okay. Okay okay. I promise. I swear. Okay? Okay. OK.

Reference: Kat feels frustrated that Bianca's desire to date affects her own freedom and choices, as their father won't allow Bianca to date until Kat does.

BLEU Score: 0.5537

Perplexity: 11.2384

Prompt: During a conversation about Bianca's appearance in '10 Things I Hate About You', what specific comment did the character Guillermo make about Bianca's hair color?

Generated response: He said that I had to grow it out because it was too dark in my hair at the time and that he didn't want to have it that way for Bianca because she was too beautiful to have dark hair in the first place. He was very supportive of Bianca though and I told him that it would be better for her if she went with the dark hair color. He said yes and I thought it was great. He's a very good man and I love him very much. He really cares about her very much and he's very passionate about her. I

Reference: Guillermo said that if Bianca's hair gets any lighter, she'll look like an extra on 90210.

BLEU Score: 0.1586

Perplexity: 8.6730

Prompt: In a scene from '10 Things I Hate About You', how does Patrick react when Kat questions his motives for being with her?

Generated response: He doesn't know. He doesn't care. He's just happy. He wants to be with her. He just wants to. I'm so happy. I love you. I don't know what I would do without you. And you're the best thing that ever happened to me. You're so wonderful. I can't take it anymore. I just want to be here with you. With you and my baby. I want to know everything about you. You and that little girl. I know everything. I think I might die. I wish I had that much time. I really do want to

Reference: Patrick gets defensive and asks Kat if he needs to have a motive

to be with her, implying that he genuinely likes her.

BLEU Score: 0.0246

Perplexity: 7.4348

Prompt: In '10 Things I Hate About You', how does Joey respond when Kat confronts him and tells him to leave her sister Bianca alone?

Generated response: He laughs hysterically at the idea of her leaving Bianca and goes off to find her own mother. Not that I'm complaining. Joey's a lucky guy. He's got a great life. I'm lucky I have one. You don't know what you're missing out on when you're out here in the world. It's a beautiful world sometimes. All you gotta do is find a good woman. Then you can go on your own adventure. And then you can have a nice big home. So when you get married you can just sit

Reference: Joey dismisses Kat's request and asks why he would leave Bianca alone, showing his lack of respect for Kat's wishes.

BLEU Score: 0.5193

Perplexity: 9.3435

Average BLEU Score: 0.2635

Average Perplexity: 9.2560

```
In [ ]: import torch
        from transformers import AutoModelForCausalLM, AutoTokenizer

        # Load the model and tokenizer
        model = AutoModelForCausalLM.from_pretrained("./fine_tuned_dialogpt_medium")
        tokenizer = AutoTokenizer.from_pretrained("./fine_tuned_dialogpt_medium")

        device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        model.to(device)

        # Constants
        MAX_HISTORY_TURNS = 5
        MAX_HISTORY_TOKENS = 512

        def generate_response(prompt, conversation_history, max_length=50):
            # Construct the full prompt with conversation history
            full_prompt = construct_prompt(conversation_history, prompt)

            input_ids = tokenizer.encode(full_prompt + tokenizer.eos_token, return_tensors='pt')
            attention_mask = torch.ones(input_ids.shape, dtype=torch.long, device=device)

            # Truncate if the input is too long
            if input_ids.shape[1] > MAX_HISTORY_TOKENS:
                input_ids = input_ids[:, -MAX_HISTORY_TOKENS:]
                attention_mask = attention_mask[:, -MAX_HISTORY_TOKENS:]

            output = model.generate(
                input_ids,
                attention_mask=attention_mask,
                max_length=input_ids.shape[1] + max_length,
                num_return_sequences=1,
                no_repeat_ngram_size=3,
                do_sample=True,
                top_k=50,
                top_p=0.95,
```

```

        temperature=0.7,
        pad_token_id=tokenizer.eos_token_id,
        eos_token_id=tokenizer.eos_token_id,
    )

    response = tokenizer.decode(output[0], skip_special_tokens=True)
    return response[len(full_prompt):].strip()

def construct_prompt(conversation_history, current_prompt):
    prompt_parts = [
        "The following is a conversation about movies, particularly '10 Things I Hate About You'. We will be discussing the movie in detail, including its plot, characters, and themes. Please provide a detailed and accurate response to the following question: ",
        *conversation_history[-MAX_HISTORY_TURNS:],
        f"Human: {current_prompt}",
        "AI:"
    ]
    return "\n".join(prompt_parts)

def chat():
    conversation_history = []
    print("Chatbot: Hello! Let's talk about '10 Things I Hate About You'. What do you want to know about the movie?")

    while True:
        user_input = input("You: ")
        if user_input.lower() in ['exit', 'quit', 'bye']:
            print("Chatbot: Goodbye! It was nice chatting with you about '10 Things I Hate About You'.")
            break

        response = generate_response(user_input, conversation_history)
        print(f"Chatbot: {response}")

        # Update conversation history
        conversation_history.append(f"Human: {user_input}")
        conversation_history.append(f"AI: {response}")

        # Keep only the last MAX_HISTORY_TURNS turns
        if len(conversation_history) > MAX_HISTORY_TURNS * 2:
            conversation_history = conversation_history[-MAX_HISTORY_TURNS * 2:]

if __name__ == "__main__":
    chat()

```

Demo Link

<https://drive.google.com/file/d/1WFkTGGor6-La6PBXVpP2upqfqoWHPxSN/view?usp=sharing>

Report on DialoGPT Medium Fine-Tuning (Cornell Movie-Dialogs Corpus)

1. Introduction

This project involved fine-tuning the DialoGPT-medium model using the **Cornell Movie-Dialogs Corpus**. The goal was to enhance the model's capability to generate coherent and contextually relevant conversational responses, especially focusing on movie-related dialogues. The fine-tuned model's performance was evaluated using specific metrics like **BLEU** (Bilingual Evaluation Understudy) and **Perplexity**.

2. Dataset Overview

- **Cornell Movie-Dialogs Corpus:**
 - A widely used dataset for training dialogue-based language models, it contains movie scripts organized into conversations, lines, and metadata.
 - For this project, 50,000 conversation pairs were extracted and used for training, ensuring diverse and robust input-output pairs.
-

3. Fine-Tuning Process

The model fine-tuning involved the following steps:

- **Tokenizer & Model Setup:**
 - The Huggingface DialoGPT-medium model and tokenizer were used.
 - The model was moved to a CUDA device to leverage GPU acceleration during training.
 - **Training Parameters:**
 - Learning Rate: `2e-5`
 - Number of Epochs: `5`
 - Batch Size: `8`
 - The **AdamW optimizer** and **linear learning rate scheduler** were used to update the model's weights.
 - Dynamic padding was applied using the `DataCollatorForLanguageModeling` method to optimize the training for tensor cores.
 - **Training Observations:**
 - The training spanned 5 epochs, with loss decreasing progressively across epochs.
 - Epoch-wise average loss:
 - Epoch 1: 2.9671
 - Epoch 2: 2.7111
 - Epoch 3: 2.5907
 - Epoch 4: 2.5063
 - Epoch 5: 2.4535
-

4. Model Evaluation

The model was evaluated using two key metrics:

- **BLEU Score:** Measures the overlap between the generated response and the reference response, particularly useful for evaluating the relevance of the output.
 - **Perplexity:** A metric to assess how well a model can predict the next word in a sequence. Lower perplexity indicates a more confident model.
-

5. Test Prompts and Generated Responses

The fine-tuned model was tested with multiple prompts to gauge its ability to produce relevant movie-based responses. The sample prompts and their corresponding generated outputs were evaluated for BLEU scores and perplexity.

Example prompts related to the movie **"10 Things I Hate About You"**:

- **Prompt:** "In the movie '10 Things I Hate About You', why does Cameron suggest that Bianca needs to learn how to lie?"
 - **Generated Response:** "Because she was telling the truth when she said she couldn't get married. She can't lie now. She was lying before she even told Bianca she was leaving..."
 - **Reference:** "Cameron suggests Bianca needs to learn how to lie because she's too honest and straightforward..."
 - **BLEU Score:** 0.0610
 - **Perplexity:** 9.5903
 - **Prompt:** "How does Kat feel about Bianca's attitude towards dating?"
 - **Generated Response:** "Well, I'd love to hear her say that to her face..."
 - **Reference:** "Kat feels frustrated that Bianca's desire to date affects her freedom..."
 - **BLEU Score:** 0.5537
 - **Perplexity:** 11.2384
-

6. Generation Performance

The **average BLEU score** across multiple prompts was **0.2635**, indicating moderate success in generating contextually appropriate responses but with room for improvement in accuracy.

The **average perplexity** was **9.2560**, reflecting the model's fluency in generating responses that align with the given prompts.

7. Results and Observations

- **Loss Decrease:** The steady decrease in loss values across epochs suggests that the model successfully learned from the dataset, improving its ability to generate coherent dialogues.
 - **BLEU and Perplexity Scores:** While the BLEU score indicates the need for more accurate responses, the perplexity values show that the model confidently generates fluent dialogue.
 - **Contextual Coherence:** The model was able to retain context across dialogue turns, making it suitable for applications requiring continuous conversation.
-

8. Conclusion

The fine-tuning of DialogPT-medium using the Cornell Movie-Dialogs Corpus was successful in enhancing the model's ability to generate conversational dialogues. However, further training and dataset expansion would be beneficial to improve the BLEU scores, enhancing response accuracy while maintaining fluency as reflected in the perplexity scores.