In [4]:
```python
# 1. Importing the Necessary Modules

import numpy as np
import nltk
from collections import Counter, defaultdict
from nltk.corpus import stopwords as nltk_stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
```

In [5]:
```python
# 2. Download resources

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt_tab')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

Out[5]: True

In [6]:
```python
# 3. Sample Document

text = """
I am learning natural language processing
Natural language processing is the important module of subject artificial intell
This domain has seen many recent advancements in terms of its execution
"""
```

In [7]:
```python
# 4. Tokenization

tokens = word_tokenize(text.lower())
stop_words = set(nltk_stopwords.words('english'))
lemmatizer = WordNetLemmatizer()
```

In [8]:
```python
# 5. Preprocessing

filtered_tokens = [lemmatizer.lemmatize(word) for word in tokens if word.isalnum
```

In [9]:
```python
# 6. Generate bigrams

def generate_n_grams(tokens, n):
    return [tuple(tokens[i:i + n]) for i in range(len(tokens) - n + 1)]

bigrams = generate_n_grams(filtered_tokens, 2)
```

In [10]:
```python
# 7. Train model (bigram)

def train_grams(n_grams):
    model = defaultdict(Counter)
    for ngram in n_grams:
```

```
        prefix = ngram[:-1]
        next_word = ngram[-1]
        model[prefix][next_word] += 1
    return model

model = train_grams(bigrams)
```

In [11]:
```
# 8. Predict with probabilities

def predict_next_word(model, prefix_words):
    prefix = tuple(prefix_words.split())
    if prefix in model:
        total = sum(model[prefix].values())
        return [(word, round(count / total, 3)) for word, count in model[prefix]
    else:
        return "No Prediction"
```

In [12]:
```
# 9. Predictions for various seed phrases

seeds = ["natural", "language", "artificial", "processing", "subject"]
print("\nPredictions:")
print(f"{'Seed Phrase':<15}{'Predicted Word':<20}{'Probability'}")
print("-" * 50)

for seed in seeds:
    prediction = predict_next_word(model, seed)
    if prediction != "No Prediction":
        for word, prob in prediction:
            print(f"{seed:<15}{word:<20}{prob}")
    else:
        print(f"{seed:<15}{'No Prediction':<20}{'-'}")
```

```
Predictions:
Seed Phrase    Predicted Word      Probability
--------------------------------------------------
natural        language            1.0
language       processing          1.0
artificial     intelligence        1.0
processing     natural             0.5
processing     important           0.5
subject        artificial          1.0
```

```
In [ ]: # 1. Importing the Necessary Modules

import numpy as np
import pandas as pd
from collections import Counter
from sklearn.metrics.pairwise import cosine_similarity
import math
import seaborn as sns
import matplotlib.pyplot as plt
```

```
In [ ]: # 2. Defining the Sample Documents

documents = [
    "Data science is an interdisciplinary field",
    "Machine learning is a part of data science",
    "Deep learning is a branch of machine learning"
]

print("Documents:")
for i, doc in enumerate(documents, 1):
    print(f"{i}. {doc}")
```

```
Documents:
1. Data science is an interdisciplinary field
2. Machine learning is a part of data science
3. Deep learning is a branch of machine learning
```

BoW Vectorization

```
In [ ]: bow_vectors = []

for doc in tokenized_docs:
    word_count = Counter(doc)
    bow_vectors.append([word_count[word] for word in vocab])

bow_df = pd.DataFrame(bow_vectors, columns=vocab)
print("\nBag of Words (BoW) Vectorization:")
display(bow_df)
```

```
Bag of Words (BoW) Vectorization:
```

	a	an	branch	data	deep	field	interdisciplinary	is	learning	machine	of	part	s
0	0	1	0	1	0	1	1	1	0	0	0	0	
1	1	0	0	1	0	0	0	1	1	1	1	1	
2	1	0	1	0	1	0	0	1	2	1	1	0	

```
In [ ]: # Cosine Similarity (BoW)

cosine_bow = cosine_similarity(bow_df)
cosine_bow_df = pd.DataFrame(cosine_bow, columns=["Doc1", "Doc2", "Doc3"], index

print("\nCosine Similarity (BoW):")
display(cosine_bow_df)
```

Cosine Similarity (BoW):

	Doc1	Doc2	Doc3
Doc1	1.000000	0.433013	0.129099
Doc2	0.433013	1.000000	0.670820
Doc3	0.129099	0.670820	1.000000

TF Vectorization

In [ ]:
```
tf_vectors = []

for doc in tokenized_docs:
    word_count = Counter(doc)
    total_words = len(doc)
    tf_vectors.append([word_count[word]/total_words for word in vocab])

tf_df = pd.DataFrame(tf_vectors, columns=vocab)
print("\nTerm Frequency (TF) Vectorization:")
display(tf_df)
```

Term Frequency (TF) Vectorization:

	a	an	branch	data	deep	field	interdisciplinary	is	learning
0	0.000	0.166667	0.000	0.166667	0.000	0.166667	0.166667	0.166667	0.00(
1	0.125	0.000000	0.000	0.125000	0.000	0.000000	0.000000	0.125000	0.12!
2	0.125	0.000000	0.125	0.000000	0.125	0.000000	0.000000	0.125000	0.25(

In [ ]:
```
# Cosine Similarity (TF)

cosine_tf = cosine_similarity(tf_df)
cosine_tf_df = pd.DataFrame(cosine_tf, columns=["Doc1", "Doc2", "Doc3"], index=[

print("\nCosine Similarity (TF):")
display(cosine_tf_df)
```

Cosine Similarity (TF):

	Doc1	Doc2	Doc3
Doc1	1.000000	0.433013	0.129099
Doc2	0.433013	1.000000	0.670820
Doc3	0.129099	0.670820	1.000000

TF-IDF Vectorization

In [ ]:
```
idf_vector = []
N = len(documents)

for word in vocab:
    df = sum([1 for doc in tokenized_docs if word in doc])
```

```
    idf = math.log(N / (1 + df))
    idf_vector.append(idf)

tf_idf_matrix = np.array(tf_vectors) * np.array(idf_vector)
tfidf_df = pd.DataFrame(tf_idf_matrix, columns=vocab)

print("\nTF-IDF Vectorization:")
display(tfidf_df)
```

```
TF-IDF Vectorization:
```

	a	an	branch	data	deep	field	interdisciplinary	is	learning
0	0.0	0.067578	0.000000	0.0	0.000000	0.067578	0.067578	-0.047947	0.0
1	0.0	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	-0.035960	0.0
2	0.0	0.000000	0.050683	0.0	0.050683	0.000000	0.000000	-0.035960	0.0

In [ ]:

```
# Cosine Similarity (TF-IDF)

cosine_tfidf = cosine_similarity(tfidf_df)
cosine_tfidf_df = pd.DataFrame(cosine_tfidf, columns=["Doc1", "Doc2", "Doc3"], i

print("\nCosine Similarity (TF-IDF):")
display(cosine_tfidf_df)
```

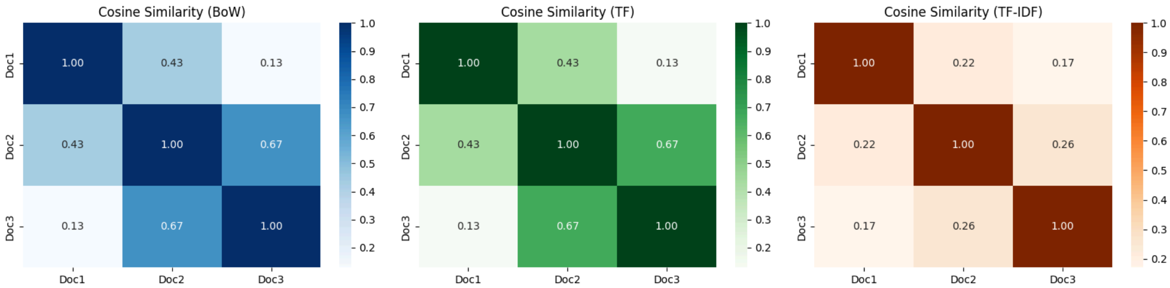
```
Cosine Similarity (TF-IDF):
```

	Doc1	Doc2	Doc3
Doc1	1.000000	0.219349	0.169984
Doc2	0.219349	1.000000	0.259487
Doc3	0.169984	0.259487	1.000000

Plot Heatmaps

In [ ]:

```
plt.figure(figsize=(16, 4))

plt.subplot(1, 3, 1) # Specify subplot index as 1
sns.heatmap(cosine_bow_df, annot=True, cmap="Blues", fmt=".2f")
plt.title("Cosine Similarity (BoW)")

plt.subplot(1, 3, 2) # Specify subplot index as 2
sns.heatmap(cosine_tf_df, annot=True, cmap="Greens", fmt=".2f")
plt.title("Cosine Similarity (TF)")

plt.subplot(1, 3, 3) # Specify subplot index as 3
sns.heatmap(cosine_tfidf_df, annot=True, cmap="Oranges", fmt=".2f")
plt.title("Cosine Similarity (TF-IDF)")

plt.tight_layout()
plt.show()
```


Cosine Similarity (BoW)
Cosine Similarity (TF)
Cosine Similarity (TF-IDF)
Doc1
Doc2
Doc3
1.00
0.43
0.13
0.67
0.22
0.17
0.26

In [2]:
```
# 1. Importing the Necessar Modules

import nltk
import spacy
import numpy as np
import networkx as nx
from sklearn.metrics import jaccard_score
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import MultiLabelBinarizer
```

In [16]:
```
# 2. Downloading the Necessary Libraries and Modules

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('punkt_tab')

from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

In [5]:
```
# 3. Load spaCy model for vectorization

nlp = spacy.load("en_core_web_sm")
```

In [6]:
```
# 4. Tokenize documents into sentences

def tokenize_sentences(text):
    return sent_tokenize(text)
```

In [7]:
```
# 5. Preprocess each sentence

def preprocess_sentence(sentence):
    stop_words = set(stopwords.words('english'))
    words = word_tokenize(sentence.lower())
    return [word for word in words if word.isalnum() and word not in stop_words]
```

In [8]:
```
# 6. Extract key phrases using CountVectorizer

def extract_key_phrases(sentences):
    preprocessed_sentences = [' '.join(preprocess_sentence(s)) for s in sentence
    vectorizer = CountVectorizer().fit(preprocessed_sentences)
    key_phrases = vectorizer.get_feature_names_out()
    return key_phrases
```

In [11]:
```
# 7. Jaccard Similarity Matrix between sentences and key phrases

def build_similarity_matrix(sentences, key_phrases):
    binarizer = MultiLabelBinarizer(classes=key_phrases)
    sentence_sets = [set(preprocess_sentence(s)) for s in sentences]
    binary_matrix = binarizer.fit_transform(sentence_sets)
```

```python
    n = len(sentences)
    similarity_matrix = np.zeros((n, n))

    for i in range(n):
        for j in range(n):
            if i != j:
                similarity_matrix[i][j] = jaccard_score(binary_matrix[i], binary

    return similarity_matrix
```

In [12]:

```python
# 8. Rank Sentences

def rank_sentences(similarity_matrix):
    graph = nx.from_numpy_array(similarity_matrix)
    scores = nx.pagerank(graph)
    return scores
```

In [13]:

```python
# 9. Get summary

def textrank_summarize(text, summary_ratio=0.3):
    sentences = tokenize_sentences(text)
    key_phrases = extract_key_phrases(sentences)
    similarity_matrix = build_similarity_matrix(sentences, key_phrases)
    scores = rank_sentences(similarity_matrix)

    ranked_sentences = sorted(((scores[i], s) for i, s in enumerate(sentences)),

    top_n = int(len(sentences) * summary_ratio)
    summary = ' '.join([sent for _, sent in ranked_sentences[:top_n]])

    return summary
```

In [17]:

```python
# 10. Implementing  the Model

text = """
Natural Language Processing (NLP) is a sub-field of artificial intelligence.
It involves understanding and generating human language.
One of the most interesting tasks in NLP is text summarization.
There are two main approaches to summarization: extractive and abstractive.
TextRank is an extractive summarization algorithm.
It is inspired by the PageRank algorithm used by Google.
TextRank builds a graph of sentences based on similarity.
Then, it ranks the sentences to pick the most important ones for the summary.
"""
print("Summary of the document :- ")
print(textrank_summarize(text))
```

```
Summary of the document :-
TextRank is an extractive summarization algorithm. There are two main approaches
to summarization: extractive and abstractive.
```

In [18]:
```python
# 1. Importing the Necessar Modules

import nltk
import spacy
import numpy as np
import networkx as nx
from sklearn.metrics import jaccard_score
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import MultiLabelBinarizer
import warnings

warnings.filterwarnings("ignore")
```

In [3]:
```python
# 2. Downloading the Necessary Libraries and Modules

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('punkt_tab')

from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
```

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

In [4]:
```python
# 3. Load spaCy model for vectorization

nlp = spacy.load("en_core_web_sm")
```

In [5]:
```python
# 4. Tokenize documents into sentences

def tokenize_sentences(text):
    return sent_tokenize(text)
```

In [6]:
```python
# 5. Preprocess each sentence

def preprocess_sentence(sentence):
    stop_words = set(stopwords.words('english'))
    words = word_tokenize(sentence.lower())
    return [word for word in words if word.isalnum() and word not in stop_words]
```

In [7]:
```python
# 6. Extract key phrases using CountVectorizer

def extract_key_phrases(sentences):
    preprocessed_sentences = [' '.join(preprocess_sentence(s)) for s in sentence
    vectorizer = CountVectorizer().fit(preprocessed_sentences)
    key_phrases = vectorizer.get_feature_names_out()
    return key_phrases
```

In [8]:
```python
# 7. Jaccard Similarity Matrix between sentences and key phrases

def build_similarity_matrix(sentences, key_phrases):
    binarizer = MultiLabelBinarizer(classes=key_phrases)
```

```
    sentence_sets = [set(preprocess_sentence(s)) for s in sentences]
    binary_matrix = binarizer.fit_transform(sentence_sets)

    n = len(sentences)
    similarity_matrix = np.zeros((n, n))

    for i in range(n):
        for j in range(n):
            if i != j:
                similarity_matrix[i][j] = jaccard_score(binary_matrix[i], binary

    return similarity_matrix
```

In [9]:
```
# 8. Rank Sentences

def rank_sentences(similarity_matrix):
    graph = nx.from_numpy_array(similarity_matrix)
    scores = nx.pagerank(graph)
    return scores
```

In [10]:
```
# 9. Get summary

def textrank_summarize(text, summary_ratio=0.3):
    sentences = tokenize_sentences(text)
    key_phrases = extract_key_phrases(sentences)
    similarity_matrix = build_similarity_matrix(sentences, key_phrases)
    scores = rank_sentences(similarity_matrix)

    ranked_sentences = sorted(((scores[i], s) for i, s in enumerate(sentences)),

    top_n = int(len(sentences) * summary_ratio)
    summary = ' '.join([sent for _, sent in ranked_sentences[:top_n]])

    return summary
```

In [11]:
```
# 10. Creating a Portfolio.

portfolio = """
I am Aryan Langhanoja
A Student of Semester 6 In Department of Information and Communication Technolog
I am Serving as a  Deputy Convenor of Competitive Programming Club.
I had done many projects in HTML CSS JS PHP Flutter React Node Express MongoDB P
I am trying to imporoving my problem solving skills by practicing DSA.
"""
```

In [19]:
```
# 11. Summary of the 50% of the size of my portfolio

print("Summary of the 50% of the size of my portfolio  :- ")
print(textrank_summarize(portfolio , 0.5))
```

```
Summary of the 50% of the size of my portfolio  :-
I had done many projects in HTML CSS JS PHP Flutter React Node Express MongoDB Po
stgreSQL etc. I am trying to imporoving my problem solving skills by practicing D
SA.
```

In [20]:
```
# 12. Summary of the 25% of the size of my portfolio

print("Summary of the 25% of the size of my portfolio  :- ")
print(textrank_summarize(portfolio , 0.25))
```

Summary of the 25% of the size of my portfolio :-
I had done many projects in HTML CSS JS PHP Flutter React Node Express MongoDB PostgreSQL etc.

In [1]:
```
# 1: Import Required Libraries

import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

In [3]:
```
# 2: Create the Dataset

data = pd.DataFrame({
    'id': [1, 2, 3, 4, 5],
    'description': [
        'Virat Kohli is a good cricketer and a sport person, he plays cricket we
        'Cricket is a famous sport in India and people likes to play it',
        'AI is changing the world and is now working as a human',
        'Natural Language Processing is an important module of AI',
        'AI is a very huge domain and it is the future'
    ]
})
```

In [4]:
```
# 3: Vectorize Text using TF-IDF

# Convert descriptions into TF-IDF vectors
tfidf = TfidfVectorizer(stop_words='english')
tfidf_vectors = tfidf.fit_transform(data["description"])
```

In [5]:
```
# 4: Compute Cosine Similarity Matrix

# Compute cosine similarity between all sentences
cosine_sim = cosine_similarity(tfidf_vectors, tfidf_vectors)
```

In [6]:
```
# 5: Choose a Sentence to Recommend From

# Choose the sentence index to find recommendations for
recommend_from = 3  # 0-based index
```

In [7]:
```
# 6: Get Similarity Scores and Sort

# Get similarity scores for the selected sentence
sim_scores = list(enumerate(cosine_sim[recommend_from]))

# Sort scores in descending order of similarity
sorted_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
```

In [9]:
```
# 7: Extract Top N Recommendations

# Top N recommendations (excluding the sentence itself)
top_n = 2
top_recommendations = sorted_scores[1:top_n + 1]  # skip self match at index 0
```

In [10]:
```
# 8: Display Recommended Sentences

# Collect recommended sentences
recommended_sentences = []
for item in top_recommendations:
    index = item[0]
    recommended_sentences.append((index, data['description'][index]))
```

```python
# Display as DataFrame
recommend_df = pd.DataFrame(recommended_sentences, columns=["Index", "Recommende
recommend_df
```

Out[10]:

	Index	Recommended Sentence
0	4	AI is a very huge domain and it is the future
1	2	AI is changing the world and is now working as...

In [1]:
```
# 1: Import Required Libraries

import warnings
import kagglehub as kg
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

warnings.filterwarnings("ignore")
```

In [2]:
```
# 2: Importing the Dataset

# Download from kaggle
path = kg.dataset_download("harshitshankhdhar/imdb-dataset-of-top-1000-movies-an

#Making the dataframe
series_df = pd.read_csv(path + '/imdb_top_1000.csv')
series_df.head()
```

Out[2]:

	Poster_Link	Series_Title	Released_Year	Certificate	Runt
0	https://m.media-amazon.com/images/M/MV5BMDFkYT...	The Shawshank Redemption	1994	A	142
1	https://m.media-amazon.com/images/M/MV5BM2MyNj...	The Godfather	1972	A	175
2	https://m.media-amazon.com/images/M/MV5BMTMxNT...	The Dark Knight	2008	UA	152
3	https://m.media-amazon.com/images/M/MV5BMWMwMG...	The Godfather: Part II	1974	A	202
4	https://m.media-amazon.com/images/M/MV5BMWU4N2...	12 Angry Men	1957	U	96

In [3]:
```
# 3. Preprocess Text (Overview Field)

# Fill missing overviews with empty string
series_df['Overview'] = series_df['Overview'].fillna('')

# TF-IDF Vectorization on Overview
tfidf = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf.fit_transform(series_df['Overview'])
```

In [4]:
```
# 4. Compute Cosine Similarity

cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)
```

In [5]:
```
# 5. Recommend Top-10 Similar Series for a Given Index

# Function to recommend top N similar TV Series
def recommend_series(title, top_n=10):
    if title not in series_df['Series_Title'].values:
        return "Series not found."

    idx = series_df[series_df['Series_Title'] == title].index[0]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Exclude the series itself and pick top_n
    top_similar = sim_scores[1:top_n+1]

    result = []
    for i, score in top_similar:
        result.append((series_df['Series_Title'][i], score))

    return pd.DataFrame(result, columns=['Recommended Series', 'Similarity Score
```

In [6]:
```
# 6: Choose a Sentence to Recommend From

# Choose the sentence index to find recommendations for
recommend_from =  series_df.iloc[0 , 7] # 0-based index
recommend_from
```

Out[6]:
```
'Two imprisoned men bond over a number of years, finding solace and eventual re
demption through acts of common decency.'
```

In [7]:
```
# 7. Get Recommendations

recommendations = recommend_series("The Invisible Man")
recommendations
```

Out[7]:

	Recommended Series	Similarity Score
0	Harvey	0.201524
1	Young Frankenstein	0.168055
2	The Butterfly Effect	0.146863
3	The Long Goodbye	0.123039
4	Rogue One	0.116654
5	Trois couleurs: Bleu	0.115169
6	Swades: We, the People	0.106805
7	Shutter Island	0.101418
8	Solaris	0.099157
9	Badhaai ho	0.090087

In [ ]:
```
# 1. Installing the Required Libraries

!pip install numpy==1.24.3 scikit-surprise
!pip install --no-cache-dir --force-reinstall scikit-surprise
```

```
Collecting numpy==1.24.3
  Downloading numpy-1.24.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_6
4.whl.metadata (5.6 kB)
Collecting scikit-surprise
  Downloading scikit_surprise-1.1.4.tar.gz (154 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 154.4/154.4 kB 2.2 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-pa
ckages (from scikit-surprise) (1.4.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-pac
kages (from scikit-surprise) (1.14.1)
Downloading numpy-1.24.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.w
hl (17.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 17.3/17.3 MB 50.0 MB/s eta 0:00:00
Building wheels for collected packages: scikit-surprise
  Building wheel for scikit-surprise (pyproject.toml) ... done
  Created wheel for scikit-surprise: filename=scikit_surprise-1.1.4-cp311-cp311-l
inux_x86_64.whl size=2505217 sha256=8696e75d1548ff64f0e68c033ac1a58ce054a40c28b2f
8659c7fb2dfa88caec5
  Stored in directory: /root/.cache/pip/wheels/2a/8f/6e/7e2899163e2d85d8266daab4a
a1cdabec7a6c56f83c015b5af
Successfully built scikit-surprise
Installing collected packages: numpy, scikit-surprise
  Attempting uninstall: numpy
    Found existing installation: numpy 2.0.2
    Uninstalling numpy-2.0.2:
      Successfully uninstalled numpy-2.0.2
ERROR: pip's dependency resolver does not currently take into account all the pac
kages that are installed. This behaviour is the source of the following dependenc
y conflicts.
tensorflow 2.18.0 requires numpy<2.1.0,>=1.26.0, but you have numpy 1.24.3 which
is incompatible.
pymc 5.21.2 requires numpy>=1.25.0, but you have numpy 1.24.3 which is incompatib
le.
albumentations 2.0.5 requires numpy>=1.24.4, but you have numpy 1.24.3 which is i
ncompatible.
blosc2 3.3.1 requires numpy>=1.26, but you have numpy 1.24.3 which is incompatibl
e.
albucore 0.0.23 requires numpy>=1.24.4, but you have numpy 1.24.3 which is incomp
atible.
jaxlib 0.5.1 requires numpy>=1.25, but you have numpy 1.24.3 which is incompatibl
e.
thinc 8.3.6 requires numpy<3.0.0,>=2.0.0, but you have numpy 1.24.3 which is inco
mpatible.
jax 0.5.2 requires numpy>=1.25, but you have numpy 1.24.3 which is incompatible.
treescope 0.1.9 requires numpy>=1.25.2, but you have numpy 1.24.3 which is incomp
atible.
Successfully installed numpy-1.24.3 scikit-surprise-1.1.4
```

```
Collecting scikit-surprise
  Downloading scikit_surprise-1.1.4.tar.gz (154 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0.0/154.4 kB ? eta -:--:--
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 154.4/154.4 kB 4.6 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Collecting joblib>=1.2.0 (from scikit-surprise)
  Downloading joblib-1.4.2-py3-none-any.whl.metadata (5.4 kB)
Collecting numpy>=1.19.5 (from scikit-surprise)
  Downloading numpy-2.2.5-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.
whl.metadata (62 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.0/62.0 kB 107.0 MB/s eta 0:00:00
Collecting scipy>=1.6.0 (from scikit-surprise)
  Downloading scipy-1.15.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_6
4.whl.metadata (61 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.0/62.0 kB 109.2 MB/s eta 0:00:00
Downloading joblib-1.4.2-py3-none-any.whl (301 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 301.8/301.8 kB 23.9 MB/s eta 0:00:00
Downloading numpy-2.2.5-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.wh
l (16.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 16.4/16.4 MB 193.0 MB/s eta 0:00:00
Downloading scipy-1.15.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.w
hl (37.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 37.6/37.6 MB 157.7 MB/s eta 0:00:00
Building wheels for collected packages: scikit-surprise
  Building wheel for scikit-surprise (pyproject.toml) ... canceledERROR: Operatio
n cancelled by user
^C
```

In [ ]:

```python
# 2. Import Required Libraries

import numpy as np
import pandas as pd
from surprise import SVD, Dataset, Reader
from surprise.model_selection import cross_validate
```

In [ ]:

```python
# 3. Prepare the Sample Dataset

# Define users, items, and ratings
users = [1, 2, 3, 4, 1, 2, 3, 4]
movies = [
    "Star Wars",
    "Hary Porter",
    "Star Wars",
    "Star Wars",
    "Hary Porter",
    "Tom Rider",
    "Hary Porter",
    "Tom Rider",
]
ratings = [1, 3, 4, 2, 3, 4, 1, 1]

# Create a dictionary and convert to DataFrame
rating_dict = {
    "userID": users,
    "ItemID": movies,
    "rating": ratings
}
```

```
df = pd.DataFrame(rating_dict)
df
```

Out[ ]:

	userID	ItemID	rating
0	1	Star Wars	1
1	2	Hary Porter	3
2	3	Star Wars	4
3	4	Star Wars	2
4	1	Hary Porter	3
5	2	Tom Rider	4
6	3	Hary Porter	1
7	4	Tom Rider	1

In [ ]:

```
# 4. Define Reader and Load Dataset

# Define the rating scale (min=1, max=5)
reader = Reader(rating_scale=(1, 5))

# Load dataset in Surprise format
data = Dataset.load_from_df(df[["userID", "ItemID", "rating"]], reader)
```

In [ ]:

```
# 5. Apply SVD Collaborative Filtering Algorithm

# Initialize the SVD algorithm
algo = SVD()

# Fit the algorithm on the data
algo.fit(data.build_full_trainset())
```

Out[ ]: <surprise.prediction_algorithms.matrix_factorization.SVD at 0x784b11b2b050>

In [ ]:

```
# 6. Evaluate the Model using Cross Validation

cross_validate(algo , data , measures = ['rmse' , 'mae'] , cv  =5 , verbose = Tr
```

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    2.0248  0.4334  1.5797  1.9429  0.9482  1.3858  0.6090
MAE (testset)     2.0219  0.3997  1.5750  1.9429  0.9482  1.3775  0.6187
Fit time          0.00    0.00    0.00    0.00    0.00    0.00    0.00
Test time         0.00    0.00    0.00    0.00    0.00    0.00    0.00
```

```
Out[ ]: {'test_rmse': array([2.02475105, 0.43342643, 1.57972267, 1.942881  , 0.9482274
9]),
 'test_mae': array([2.0218816 , 0.39971443, 1.57495684, 1.942881  , 0.9482274
9]),
 'fit_time': (0.0014197826385498047,
  0.00039076805114746094,
  0.0002872943878173828,
  0.0002970695495605469,
  0.00026917457580566406),
 'test_time': (0.00011086463928222656,
  0.00043654441833496094,
  4.124641418457031e-05,
  2.8133392333984375e-05,
  2.7894973754882812e-05)}
```

In [ ]:
```
# 1. Installing the Required Libraries

!pip install numpy==1.24.3 scikit-surprise
!pip install --no-cache-dir --force-reinstall scikit-surprise
```

```
Collecting numpy==1.24.3
  Downloading numpy-1.24.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_6
4.whl.metadata (5.6 kB)
Collecting scikit-surprise
  Downloading scikit_surprise-1.1.4.tar.gz (154 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 154.4/154.4 kB 2.8 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-pa
ckages (from scikit-surprise) (1.4.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-pac
kages (from scikit-surprise) (1.14.1)
Downloading numpy-1.24.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.w
hl (17.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 17.3/17.3 MB 90.7 MB/s eta 0:00:00
Building wheels for collected packages: scikit-surprise
  Building wheel for scikit-surprise (pyproject.toml) ... done
  Created wheel for scikit-surprise: filename=scikit_surprise-1.1.4-cp311-cp311-l
inux_x86_64.whl size=2505203 sha256=958ce6ad760115de3a88c674ed1431eb62d56c3fd3dfb
81940d7f383e6c5f7fe
  Stored in directory: /root/.cache/pip/wheels/2a/8f/6e/7e2899163e2d85d8266daab4a
a1cdabec7a6c56f83c015b5af
Successfully built scikit-surprise
Installing collected packages: numpy, scikit-surprise
  Attempting uninstall: numpy
    Found existing installation: numpy 2.0.2
    Uninstalling numpy-2.0.2:
      Successfully uninstalled numpy-2.0.2
ERROR: pip's dependency resolver does not currently take into account all the pac
kages that are installed. This behaviour is the source of the following dependenc
y conflicts.
tensorflow 2.18.0 requires numpy<2.1.0,>=1.26.0, but you have numpy 1.24.3 which
is incompatible.
pymc 5.21.2 requires numpy>=1.25.0, but you have numpy 1.24.3 which is incompatib
le.
albumentations 2.0.5 requires numpy>=1.24.4, but you have numpy 1.24.3 which is i
ncompatible.
blosc2 3.3.1 requires numpy>=1.26, but you have numpy 1.24.3 which is incompatibl
e.
albucore 0.0.23 requires numpy>=1.24.4, but you have numpy 1.24.3 which is incomp
atible.
jaxlib 0.5.1 requires numpy>=1.25, but you have numpy 1.24.3 which is incompatibl
e.
thinc 8.3.6 requires numpy<3.0.0,>=2.0.0, but you have numpy 1.24.3 which is inco
mpatible.
jax 0.5.2 requires numpy>=1.25, but you have numpy 1.24.3 which is incompatible.
treescope 0.1.9 requires numpy>=1.25.2, but you have numpy 1.24.3 which is incomp
atible.
Successfully installed numpy-1.24.3 scikit-surprise-1.1.4
```

```
Collecting scikit-surprise
  Downloading scikit_surprise-1.1.4.tar.gz (154 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0.0/154.4 kB ? eta -:--:--
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 154.4/154.4 kB 4.4 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Collecting joblib>=1.2.0 (from scikit-surprise)
  Downloading joblib-1.4.2-py3-none-any.whl.metadata (5.4 kB)
Collecting numpy>=1.19.5 (from scikit-surprise)
  Downloading numpy-2.2.5-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.
whl.metadata (62 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.0/62.0 kB 131.9 MB/s eta 0:00:00
Collecting scipy>=1.6.0 (from scikit-surprise)
  Downloading scipy-1.15.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_6
4.whl.metadata (61 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.0/62.0 kB 19.7 MB/s eta 0:00:00
Downloading joblib-1.4.2-py3-none-any.whl (301 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 301.8/301.8 kB 24.9 MB/s eta 0:00:00
Downloading numpy-2.2.5-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.wh
l (16.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 16.4/16.4 MB 184.5 MB/s eta 0:00:00
Downloading scipy-1.15.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.w
hl (37.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 37.6/37.6 MB 228.8 MB/s eta 0:00:00
Building wheels for collected packages: scikit-surprise
```

In [16]:

```python
# 2. Import Required Libraries

import numpy as np
import pandas as pd
from math import sqrt
import time
import os
import psutil
import tracemalloc

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

# Surprise Library for Recommender Systems
from surprise import Dataset, Reader, SVD, KNNBasic, NMF
from surprise.model_selection import cross_validate, train_test_split

# Evaluation Metrics
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Prettify plots
sns.set(style="whitegrid")
```

In [7]:

```python
# 3. Loading the dataset

names = ['user_id', 'item_id', 'rating', 'timestamp']
df = pd.read_csv("./Dataset/ml-100k/u.data", sep='\t', names=names)
```

In [14]:

```python
# 4. Preprocessing the dataset

df.dropna(inplace=True)
```

```
reader = Reader(rating_scale=(1, 5))  # Define the rating scale
data = Dataset.load_from_df(df[['user_id', 'item_id', 'rating']], reader)
```

In [10]:
```
# 5. Defining the models

models = {
    "SVD": SVD(),
    "KNN": KNNBasic(),
    "NMF": NMF()
}
```

In [12]:
```
# 6. List to store the results

results = []
```

In [17]:
```
# 7. Evaluating the models

for name, algo in models.items():

    print(f"Training {name}...")

    tracemalloc.start() # To track thhe used memory and time
    start = time.time()

    # Cross-validate with 5-fold
    cv_results = cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=5, verb

    end = time.time()
    current, peak = tracemalloc.get_traced_memory()
    tracemalloc.stop()

    # Store the formated result

    results.append({
        'Model': name,
        'Avg Fit Time (s)': round(np.mean(cv_results['fit_time']), 4),
        'Avg Memory (KB)': round(peak / 1024, 2),
        'Avg RMSE': round(np.mean(cv_results['test_rmse']), 4),
        'Avg MAE': round(np.mean(cv_results['test_mae']), 4)
    })
```

```
Training SVD...
Training KNN...
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Training NMF...
```

In [18]:
```
# Creating dataframe from results

results_df = pd.DataFrame(results)
```

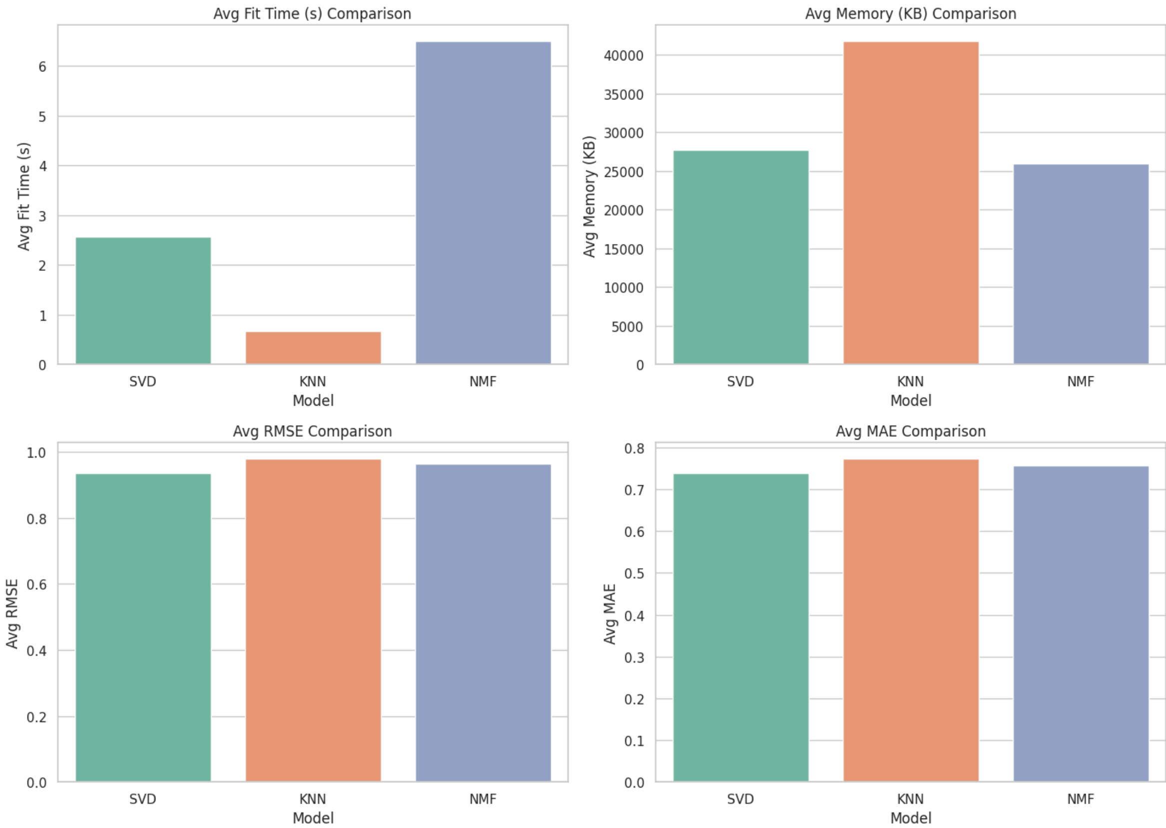
```
print("\nEvaluation Summary:")
print(results_df)
```

```
Evaluation Summary:
  Model  Avg Fit Time (s)  Avg Memory (KB)  Avg RMSE  Avg MAE
0   SVD            2.5738         27708.06    0.9364   0.7382
1   KNN            0.6628         41719.05    0.9783   0.7726
2   NMF            6.4952         25961.37    0.9631   0.7569
```

In [19]:

```
# Plotting the result

metrics = ['Avg Fit Time (s)', 'Avg Memory (KB)', 'Avg RMSE', 'Avg MAE']

plt.figure(figsize=(14, 10))
for i, metric in enumerate(metrics):
    plt.subplot(2, 2, i + 1)
    sns.barplot(x='Model', y=metric, data=results_df, palette='Set2')
    plt.title(f'{metric} Comparison')
    plt.ylabel(metric)
    plt.xlabel("Model")

plt.tight_layout()
plt.show()
```

```
<ipython-input-19-42e1b2ec66b1>:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.barplot(x='Model', y=metric, data=results_df, palette='Set2')
<ipython-input-19-42e1b2ec66b1>:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.barplot(x='Model', y=metric, data=results_df, palette='Set2')
<ipython-input-19-42e1b2ec66b1>:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.barplot(x='Model', y=metric, data=results_df, palette='Set2')
<ipython-input-19-42e1b2ec66b1>:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.barplot(x='Model', y=metric, data=results_df, palette='Set2')
```



In [27]:

```
# Get a list of all movie IDs
movie_ids = df['item_id'].unique()

# Function to predict ratings and get movie titles
def predict_ratings_with_titles(user_id):
    predictions = []
    for movie_id in movie_ids:
        pred = algo.predict(user_id, movie_id)
        predictions.append((movie_id, pred.est))

    # Create a DataFrame from predictions
    preds_df = pd.DataFrame(predictions, columns=['movie_id', 'Predicted Rating'

    # Merge with movie titles
    preds_df = pd.merge(preds_df, movies_df, on='movie_id')

    return preds_df
```

```
Top 10 movie recommendations for user 1:
```

Out[27]:

	movie_id	movie_title	Predicted Rating
1428	1512	World of Apu, The (Apur Sansar) (1959)	5.000000
1239	1367	Faust (1994)	5.000000
1513	851	Two or Three Things I Know About Her (1966)	5.000000
1571	1642	Some Mother's Son (1996)	5.000000
1271	1524	Kaspar Hauser (1993)	5.000000
1608	1643	Angel Baby (1995)	4.942329
1647	1201	Marlene Dietrich: Shadow and Light (1996)	4.924653
277	169	Wrong Trousers, The (1993)	4.888815
541	513	Third Man, The (1949)	4.866707
180	408	Close Shave, A (1995)	4.865740

In [28]:

```
# Example usage: predict ratings for user 1
user_id = 1
predicted_ratings_with_titles = predict_ratings_with_titles(user_id)

# Sort by predicted rating and get top 10
top_10_recommendations = predicted_ratings_with_titles.sort_values(by=['Predicte

# Display the top 10 recommendations with movie ID, title, and rating
print(f"Top 10 movie recommendations for user {user_id}:")
top_10_recommendations[['movie_id', 'movie_title', 'Predicted Rating']]
```

```
Top 10 movie recommendations for user 1:
```

Out[28]:

	movie_id	movie_title	Predicted Rating
1428	1512	World of Apu, The (Apur Sansar) (1959)	5.000000
1239	1367	Faust (1994)	5.000000
1513	851	Two or Three Things I Know About Her (1966)	5.000000
1571	1642	Some Mother's Son (1996)	5.000000
1271	1524	Kaspar Hauser (1993)	5.000000
1608	1643	Angel Baby (1995)	4.942329
1647	1201	Marlene Dietrich: Shadow and Light (1996)	4.924653
277	169	Wrong Trousers, The (1993)	4.888815
541	513	Third Man, The (1949)	4.866707
180	408	Close Shave, A (1995)	4.865740

In [1]:
```python
# 1. Import the Necessary Modules

from typing import List, Tuple, Optional
```

In [2]:
```python
# 2. Define the Constants

# Size constants
N = 9
digits = set(str(i) for i in range(1, 10))
```

In [3]:
```python
# 3. Helper functions

def is_valid(board: List[List[str]], row: int, col: int, num: str) -> bool:
    block_row, block_col = 3 * (row // 3), 3 * (col // 3)
    for i in range(9):
        if board[row][i] == num or board[i][col] == num:
            return False
    for i in range(3):
        for j in range(3):
            if board[block_row + i][block_col + j] == num:
                return False
    return True

def find_empty(board: List[List[str]]) -> Optional[Tuple[int, int]]:
    for i in range(9):
        for j in range(9):
            if board[i][j] == '.':
                return (i, j)
    return None
```

In [4]:
```python
# 4. Backtracking CSP Solver

def solve_sudoku(board: List[List[str]]) -> bool:
    empty = find_empty(board)
    if not empty:
        return True  # Puzzle solved

    row, col = empty
    for num in map(str, range(1, 10)):
        if is_valid(board, row, col, num):
            board[row][col] = num
            if solve_sudoku(board):
                return True
            board[row][col] = '.'  # Backtrack
    return False
```

In [5]:
```python
# 5. Pretty print function

def print_board(board: List[List[str]]):
    for i in range(9):
        print(" ".join(board[i]))
```

In [6]:
```python
# 6. Definig The Sudoku

# Sample Sudoku puzzle ('.' denotes empty cells)
sudoku_board = [
    ['5', '3', '.', '.', '7', '.', '.', '.', '.'],
```

```
    ['6', '.', '.', '1', '9', '5', '.', '.', '.'],
    ['.', '9', '8', '.', '.', '.', '.', '6', '.'],
    ['8', '.', '.', '.', '6', '.', '.', '.', '3'],
    ['4', '.', '.', '8', '.', '3', '.', '.', '1'],
    ['7', '.', '.', '.', '2', '.', '.', '.', '6'],
    ['.', '6', '.', '.', '.', '.', '2', '8', '.'],
    ['.', '.', '.', '4', '1', '9', '.', '.', '5'],
    ['.', '.', '.', '.', '8', '.', '.', '7', '9']
]

print("Initial Sudoku Board:")
print_board(sudoku_board)
```

```
Initial Sudoku Board:
5 3 . . 7 . . . .
6 . . 1 9 5 . . .
. 9 8 . . . . 6 .
8 . . . 6 . . . 3
4 . . 8 . 3 . . 1
7 . . . 2 . . . 6
. 6 . . . . 2 8 .
. . . 4 1 9 . . 5
. . . . 8 . . 7 9
```

In [7]:
```
# 7. Solving the Sudoku

solve_sudoku(sudoku_board)
print("\nSolved Sudoku Board:")
print_board(sudoku_board)
```

```
Solved Sudoku Board:
5 3 4 6 7 8 9 1 2
6 7 2 1 9 5 3 4 8
1 9 8 3 4 2 5 6 7
8 5 9 7 6 1 4 2 3
4 2 6 8 5 3 7 9 1
7 1 3 9 2 4 8 5 6
9 6 1 5 3 7 2 8 4
2 8 7 4 1 9 6 3 5
3 4 5 2 8 6 1 7 9
```

In [ ]:

```
# 1: Importing Required Libraries

import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
import random
import networkx as nx
import pylab as pl
```

In [ ]:

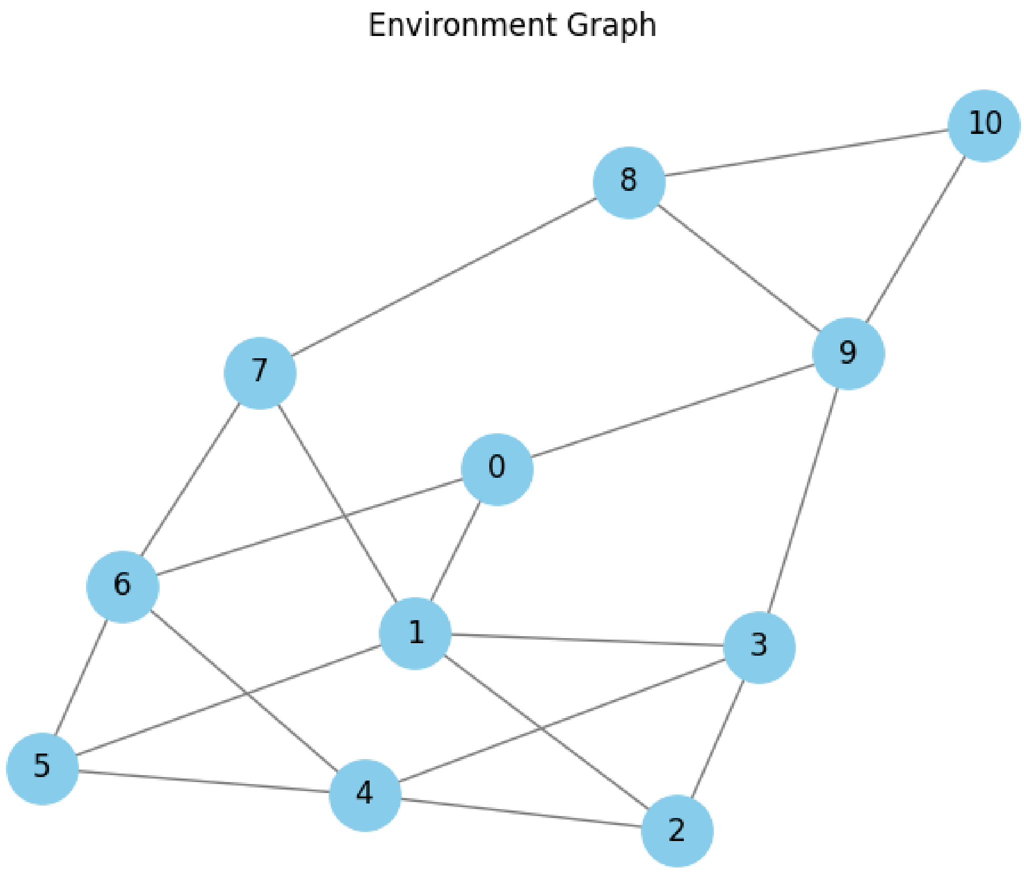
```
# 2: Create the Graph

edges = [(0,1),(1,2),(1,3),(1,5),(5,6),(5,4),(9,10),(2,4),(0,6),(6,7),(8,9),(7,8
         (0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7), (7, 8), (8, 9),

G = nx.Graph()
G.add_edges_from(edges)
```

In [ ]:

```
# 3: Define Goal State

Matrix_size = 11  # Number of nodes
goal = 10         # Goal node
```

In [ ]:

```
# 4: Visualize the Graph

pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True, node_color='skyblue', node_size=800, edge_colc
plt.title("Environment Graph")
plt.show()
```



In [ ]:
```python
# 5: Create Reward Matrix
R = np.matrix(np.ones((Matrix_size, Matrix_size)) * -1)

for edge in edges:
    if edge[1] == goal:
        R[edge] = 100
    else:
        R[edge] = 0

    if edge[0] == goal:
        R[edge[::-1]] = 100
    else:
        R[edge[::-1]] = 0

R[goal, goal] = 100  # Reward for reaching goal
```

In [ ]:
```python
# 6: Initialize Q-Table

Q = np.matrix(np.zeros((Matrix_size, Matrix_size)))
```

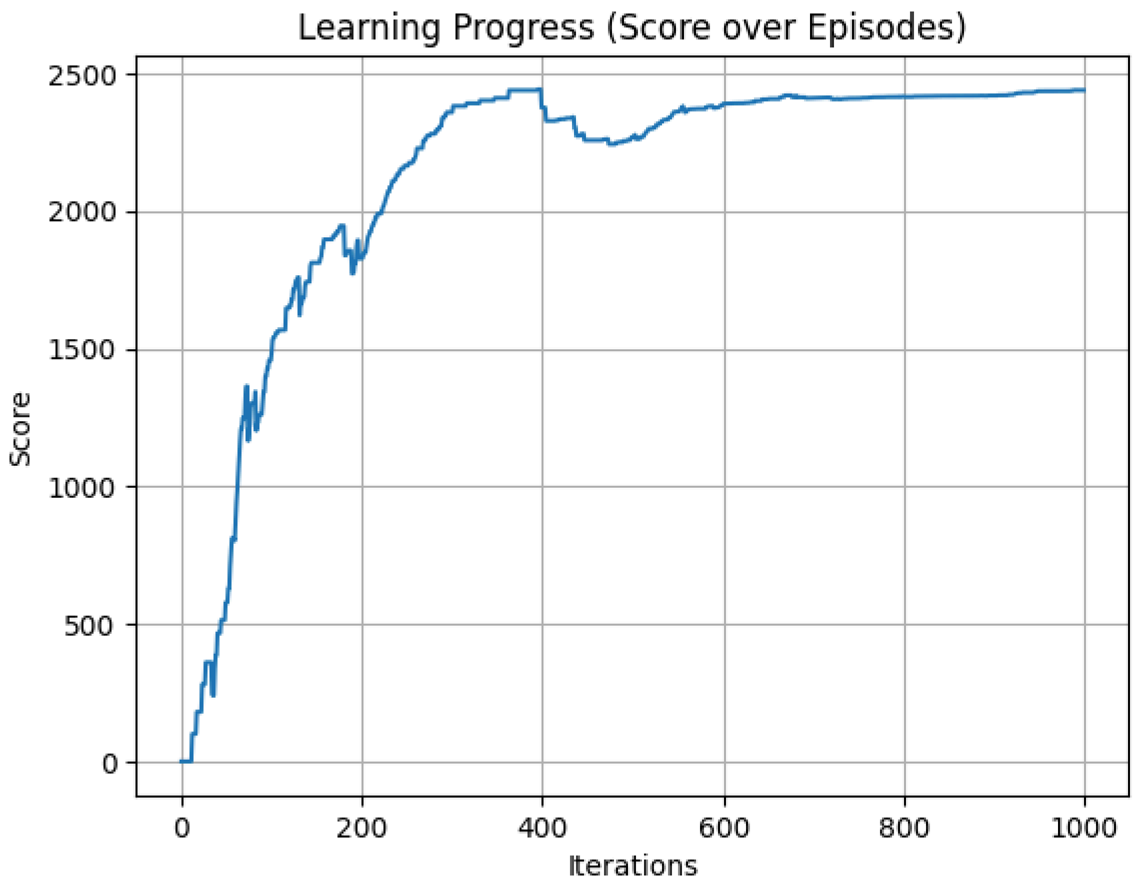
In [ ]:
```python
# 7: Define Helper Functions

def available_actions(state):
    return np.where(R[state, :] >= 0)[1]

def select_next_state(available_actions):
    return int(np.random.choice(available_actions, 1))

def update_Q(current_state, action, gamma=0.8):
    next_state = action
    max_index = np.where(Q[next_state, :] == np.max(Q[next_state, :]))[1]

    if max_index.shape[0] > 1:
        max_index = int(np.random.choice(max_index, 1))
    else:
        max_index = int(max_index)

    max_value = Q[next_state, max_index]
    Q[current_state, next_state] = R[current_state, next_state] + gamma * max_va

    if np.max(Q) > 0:
        return np.sum(Q / np.max(Q) * 100)
    else:
        return 0
```

In [ ]:
```python
# 8: Train the Agent using Q-Learning
score = []
epochs = 1000
for i in range(epochs):
    current_state = np.random.randint(0, Matrix_size)
    available_act = available_actions(current_state)
    action = select_next_state(available_act)
    score.append(update_Q(current_state, action))
```

```
<ipython-input-12-2a5041c4bf58>:6: DeprecationWarning: Conversion of an array wit
h ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extrac
t a single element from your array before performing this operation. (Deprecated
NumPy 1.25.)
  return int(np.random.choice(available_actions, 1))
<ipython-input-12-2a5041c4bf58>:13: DeprecationWarning: Conversion of an array wi
th ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extra
ct a single element from your array before performing this operation. (Deprecated
NumPy 1.25.)
  max_index = int(np.random.choice(max_index, 1))
<ipython-input-12-2a5041c4bf58>:15: DeprecationWarning: Conversion of an array wi
th ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extra
ct a single element from your array before performing this operation. (Deprecated
NumPy 1.25.)
  max_index = int(max_index)
```

In [ ]:

```
# 9: Plot the Learning Progress

plt.plot(score)
plt.title("Learning Progress (Score over Episodes)")
plt.xlabel("Iterations")
plt.ylabel("Score")
plt.grid(True)
plt.show()
```



In [ ]:

```
# 10: Extract the Optimal Path
def get_optimal_path(start_state, goal_state):
    current_state = start_state
    path = [current_state]

    while current_state != goal_state:
        next_step = np.where(Q[current_state, :] == np.max(Q[current_state, :]))
```

```python
        if next_step.shape[0] > 1:
            next_step = int(np.random.choice(next_step, 1))
        else:
            next_step = int(next_step)

        path.append(next_step)
        current_state = next_step

    return path
```

In [ ]:

```python
# 11: Test the Agent

initial_state = random.randint(0, 10)  # Change this to test from other nodes
optimal_path = get_optimal_path(initial_state, goal)
print(f"Shortest path from {initial_state} to {goal} using Q-learning: {optimal_
```

```
Shortest path from 1 to 10 using Q-learning: [1, 3, 9, 10]
```

```
<ipython-input-15-c21c65bea689>:12: DeprecationWarning: Conversion of an array wi
th ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extra
ct a single element from your array before performing this operation. (Deprecated
NumPy 1.25.)
  next_step = int(next_step)
```

In [ ]:
```python
# 1: Importing Required Libraries

import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
import random
import networkx as nx
import pylab as pl
```

In [ ]:
```python
# 2: Create the maze

maze = [
    [0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
    [0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0],
    [0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0],
    [1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1],
    [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
    [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
]

ROWS = len(maze)
COLS = len(maze[0])
```

In [ ]:
```python
# 3: Convert 2D position to state number

def pos_to_state(row, col):
    return row * COLS + col
```

In [ ]:
```python
# 4: Convert state number to 2D position

def state_to_pos(state):
    return divmod(state, COLS)
```

In [ ]:
```python
# 5: Get valid neighbors

def get_neighbors(r, c):
    moves = [(-1,0), (1,0), (0,-1), (0,1)]
    neighbors = []
    for dr, dc in moves:
        nr, nc = r + dr, c + dc
        if 0 <= nr < ROWS and 0 <= nc < COLS and maze[nr][nc] == 1:
            neighbors.append((nr, nc))
    return neighbors
```

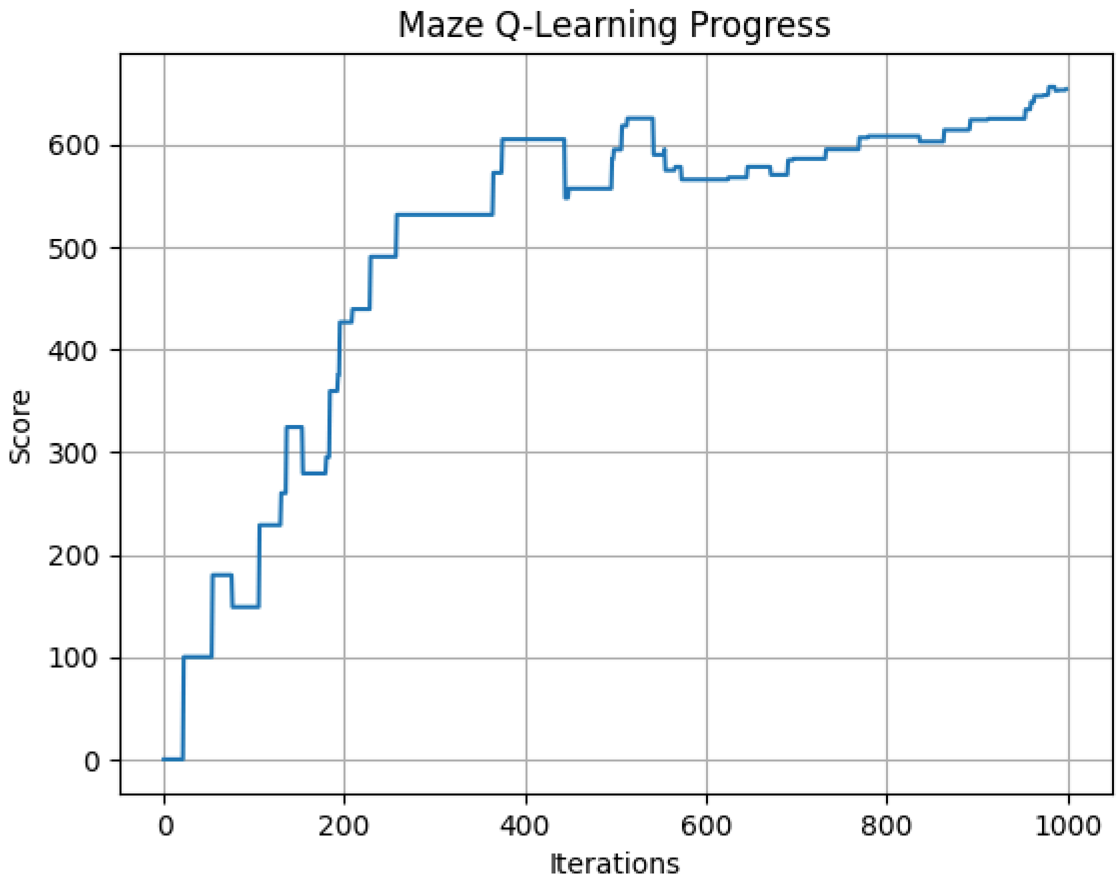
In [ ]:
```python
# 6: Create reward matrix

N = ROWS * COLS
R = np.full((N, N), -1)

goal = (6, 11)
goal_state = pos_to_state(*goal)

for r in range(ROWS):
    for c in range(COLS):
        if maze[r][c] == 1:
            s = pos_to_state(r, c)
```

```
            for nr, nc in get_neighbors(r, c):
                ns = pos_to_state(nr, nc)
                if (nr, nc) == goal:
                    R[s, ns] = 100
                else:
                    R[s, ns] = 0
```

In [ ]:
```
# 7: Q Matrix

Q = np.zeros((N, N))
```

In [ ]:
```
# 8: Q-Learning

gamma = 0.8
epochs = 1000
scores = []

for _ in range(epochs):
    current_pos = (random.randint(0, ROWS-1), random.randint(0, COLS-1))
    while maze[current_pos[0]][current_pos[1]] != 1:
        current_pos = (random.randint(0, ROWS-1), random.randint(0, COLS-1))

    state = pos_to_state(*current_pos)

    valid_actions = np.where(R[state] >= 0)[0]
    action = np.random.choice(valid_actions)

    next_state = action
    max_q = np.max(Q[next_state])
    Q[state, next_state] = R[state, next_state] + gamma * max_q

    scores.append(np.sum(Q / (np.max(Q) if np.max(Q) > 0 else 1) * 100))
```

In [ ]:
```
# 9: Plot the Learning Progress

plt.plot(scores)
plt.title("Maze Q-Learning Progress")
plt.xlabel("Iterations")
plt.ylabel("Score")
plt.grid(True)
plt.show()
```



In [ ]:
```
# 10: Extract the Optimal Path

def get_optimal_path(start):
    path = []
    current_state = pos_to_state(*start)
    path.append(start)

    while current_state != goal_state:
        next_states = np.where(Q[current_state] == np.max(Q[current_state]))[0]
        if len(next_states) > 1:
            next_state = np.random.choice(next_states)
        else:
            next_state = next_states[0]

        next_pos = state_to_pos(next_state)
        path.append(next_pos)
        current_state = next_state

        if len(path) > 100:
            break  # Prevent infinite loop

    return path
```

In [ ]:
```
# 11: Test the Agent

start = (0, 1)
optimal_path = get_optimal_path(start)
print("Optimal path from start to goal:")
print(optimal_path)
```

```
Optimal path from start to goal:
[(0, 1), (np.int64(0), np.int64(2)), (np.int64(1), np.int64(7)), (np.int64(4), n
p.int64(0)), (np.int64(6), np.int64(10)), (np.int64(3), np.int64(1)), (np.int64
(2), np.int64(1)), (np.int64(1), np.int64(7)), (np.int64(3), np.int64(2)), (np.in
t64(5), np.int64(6)), (np.int64(5), np.int64(7)), (np.int64(5), np.int64(8)), (n
p.int64(5), np.int64(9)), (np.int64(5), np.int64(10)), (np.int64(5), np.int64(1
1)), (np.int64(6), np.int64(11))]
```