```
from google.colab import files
uploaded = files.upload()
```

Choose files archive.zip

archive.zip(application/x-zip-compressed) - 9317430 bytes, last modified: 04/11/2025 - 100% done

```
Saving archive.zip to archive.zip
```

```
!unzip archive.zip
```

```
Archive:  archive.zip
  inflating: tmdb_5000_credits.csv
  inflating: tmdb_5000_movies.csv
```

```
import pandas as pd

movies_path = '/content/tmdb_5000_movies.csv'
credits_path = '/content/tmdb_5000_credits.csv'

movies = pd.read_csv(movies_path)
credits = pd.read_csv(credits_path)

# Merge datasets
df = movies.merge(credits, left_on='id', right_on='movie_id', how='left')

print("Movies shape:", movies.shape)
print("Credits shape:", credits.shape)
print("Merged shape:", df.shape)
df.head(2)
```

```
Movies shape: (4803, 20)
Credits shape: (4803, 4)
Merged shape: (4803, 24)
```

	budget	genres	homepage	id	keywords	original_language	original_title	overview	pop
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	en	Avatar	In the 22nd century, a paraplegic Marine is di...	15
1	300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	http://disney.go.com/disneypictures/pirates/	285	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	en	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...	13

2 rows × 24 columns

```
import ast

def parse(x):
    try:
        return [i['name'] for i in ast.literal_eval(x)]
    except:
        return []

df['genres'] = df['genres'].apply(parse)
df['keywords'] = df['keywords'].apply(parse)
df['cast'] = df['cast'].apply(parse)

# Extract only director from crew
def get_director(x):
    try:
        crew_list = ast.literal_eval(x)
        for i in crew_list:
            if i.get('job') == 'Director':
                return [i.get('name')]
        return []
        except:
            return []
    df['crew'] = df['crew'].apply(get_director)

    # Fill missing overviews
    df['overview'] = df['overview'].fillna('')
```

```
import pandas as pd
import re
from nltk.corpus import stopwords

import nltk
nltk.download('stopwords')
STOP_WORDS = set(stopwords.words('english'))

def clean_text(text):
    if pd.isna(text):
        return ""
    text = str(text).lower()
    text = re.sub(r"[^a-z0-9 ]+", " ", text)
    text = re.sub(r"\s+", " ", text).strip()
    tokens = [t for t in text.split() if t not in STOP_WORDS]
    return " ".join(tokens)

def create_soup(df, features):
    soups = []
    for _, row in df.iterrows():
        parts = []
        for feat in features:
            val = row.get(feat, "")
            if isinstance(val, list):
                parts.append(" ".join(val))
            else:
                parts.append(str(val))
        soup = " ".join(parts)
        soups.append(clean_text(soup))
    return soups

FEATURES = ['overview', 'genres', 'keywords', 'cast', 'crew']
```

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

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

```
df['soup'] = create_soup(df, FEATURES)
tfidf = TfidfVectorizer(max_features=5000, ngram_range=(1,2))
X_tfidf = tfidf.fit_transform(df['soup'])
```

```
import pandas as pd
import numpy as np
import re
import ast
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.preprocessing import normalize
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')

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

def clean_text(text):
    if pd.isna(text):
        return ""
    # lowercase, remove non-alphanumeric (keep spaces), collapse spaces
    text = str(text).lower()
    text = re.sub(r"[^a-z0-9 ]+", ' ', text)
    text = re.sub(r"\s+", ' ', text).strip()
    # optionally remove stopwords
```

```
        tokens = [t for t in text.split() if t not in STOP_WORDS]
        return ' '.join(tokens)

    # Parse JSON-like fields found in TMDB datasets (genres, cast, crew, keywords)
    def parse_json_field(x, key='name', top_n=None):
        # x might be a string representation of a list of dicts
        if pd.isna(x) or x == '':
            return []
        try:
            parsed = ast.literal_eval(x)
        except Exception:
            # fallback: try to clean and split
            return []
        out = []
        for i, elem in enumerate(parsed):
            if isinstance(elem, dict) and key in elem:
                out.append(str(elem[key]).replace(' ', ''))  # remove spaces to keep tokens
            elif isinstance(elem, str):
                out.append(elem.replace(' ', ''))
            if top_n and i+1 >= top_n:
                break
        return out

    # Create a 'soup' by combining selected features
    def create_soup(df, features):
        soups = []
        for _, row in df.iterrows():
            parts = []
            for feat in features:
                val = row.get(feat, '')
                if isinstance(val, list):
                    parts.append(' '.join(val))
                else:
                    parts.append(str(val))
            soup = ' '.join(parts)
            soups.append(clean_text(soup))
        return soups


    def get_demo_df():
        data = [
            {'title': 'The Matrix', 'overview': 'A computer hacker learns about the true nature of reality.',
             'genres': "[{'id': 878, 'name': 'Science Fiction'}, {'id': 28, 'name': 'Action'}]",
             'keywords': "[{'id': 1, 'name': 'artificial intelligence'}, {'id': 2, 'name': 'virtual reality'}]",
             'cast': "[{'name':'Keanu Reeves'},{'name':'Laurence Fishburne'},{'name':'Carrie-Anne Moss'}]",
             'crew': "[{'job':'Director','name':'Lana Wachowski'},{'job':'Director','name':'Lilly Wachowski'}]"
            },
            {'title': 'Inception', 'overview': "A thief who steals corporate secrets through dream-sharing technology.",
             'genres': "[{'id': 878, 'name': 'Science Fiction'}, {'id': 53, 'name': 'Thriller'}]",
             'keywords': "[{'id': 3, 'name':'dream'},{'id':4,'name':'subconscious'}]",
             'cast': "[{'name':'Leonardo DiCaprio'},{'name':'Joseph Gordon-Levitt'}]",
             'crew': "[{'job':'Director','name':'Christopher Nolan'}]"
            },
            {'title': 'Johnny Mnemonic', 'overview': 'A courier with a cybernetic brain implant.',
             'genres': "[{'id': 878, 'name': 'Science Fiction'}, {'id': 35, 'name': 'Comedy'}]",
             'keywords': "[{'id':5,'name':'cyberspace'},{'id':6,'name':'hacker'}]",
             'cast': "[{'name':'Keanu Reeves'},{'name':'Dina Meyer'}]",
             'crew': "[{'job':'Director','name':'Robert Longo'}]"
            }
        ]
        return pd.DataFrame(data)

    # Use demo dataset for immediate testing
    df = get_demo_df()

    # Parse JSON fields into lists
    for col in ['genres','keywords','cast','crew']:
        if col in df.columns:
            df[col] = df[col].apply(lambda x: parse_json_field(x, key='name', top_n=5) if col!='crew' else parse_json_field(x, key='
            # For crew you might want to keep directors only
            if col == 'crew':
                # keep only directors
                def keep_directors(x):
                    # x originally list of dicts; our parse above flattened to names, so just return as-is
                    return x
                df['crew'] = df['crew'].apply(keep_directors)
```

```
# Fill missing 'overview' with empty strings
if 'overview' in df.columns:
    df['overview'] = df['overview'].fillna('')

# Feature engineering: choose features to combine
FEATURES = ['overview', 'genres', 'keywords', 'cast', 'crew']

# Create content soup
df['soup'] = create_soup(df, FEATURES)

# TF-IDF pipeline

tfidf = TfidfVectorizer(max_features=5000, ngram_range=(1,2))
X_tfidf = tfidf.fit_transform(df['soup'])

# Computing cosine similarity matrix (dense may be large for big datasets)
cosine_sim = cosine_similarity(X_tfidf, X_tfidf)

# Reverse mapping of indices and movie titles
indices = pd.Series(df.index, index=df['title']).drop_duplicates()

# Recommendation function using TF-IDF cosine similarity

def recommend_tfidf(title, top_n=10):
    if title not in indices:
        raise ValueError(f"Title '{title}' not found in database. Available examples: {list(indices.index)[:10]}")
    idx = indices[title]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:top_n+1]  # skip self
    movie_indices = [i for i, score in sim_scores]
    return df.iloc[movie_indices][['title','overview']].assign(score=[score for i,score in sim_scores])

# Example usage with demo data
print('Recommendations for The Matrix (TF-IDF):')
print(recommend_tfidf('The Matrix', top_n=2))
```

```
Recommendations for The Matrix (TF-IDF):
             title                                          overview  \
2  Johnny Mnemonic         A courier with a cybernetic brain implant.
1        Inception  A thief who steals corporate secrets through d...

      score
2  0.062760
1  0.011657
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Start coding or generate with AI.