Project Title	Movie Recommender Systems
Tools	Jupyter Notebook and VS code
Technologies	Data Science
Project Difficulties level	intermediate

Dataset : Dataset is available in the given link. You can download it at your convenience.

Click here to download data set

Movies Recommender System

About Dataset

Context

These files contain metadata for all 45,000 movies listed in the Full MovieLens Dataset. The dataset consists of movies released on or before July 2017. Data points include cast, crew, plot keywords, budget, revenue, posters, release dates, languages, production companies, countries, TMDB vote counts and vote averages.

This dataset also has files containing 26 million ratings from 270,000 users for all 45,000 movies. Ratings are on a scale of 1-5 and have been obtained from the official GroupLens website.

Content

This dataset consists of the following files:

movies_metadata.csv: The main Movies Metadata file. Contains information on 45,000 movies featured in the Full MovieLens dataset. Features include posters, backdrops, budget, revenue, release dates, languages, production countries and companies.

keywords.csv: Contains the movie plot keywords for our MovieLens movies. Available in the form of a stringified JSON Object.

credits.csv: Consists of Cast and Crew Information for all our movies. Available in the form of a stringified JSON Object.

links.csv: The file that contains the TMDB and IMDB IDs of all the movies featured in the Full MovieLens dataset.

links_small.csv: Contains the TMDB and IMDB IDs of a small subset of 9,000 movies of the Full Dataset.

ratings_small.csv: The subset of 100,000 ratings from 700 users on 9,000 movies.

The Full MovieLens Dataset consisting of 26 million ratings and 750,000 tag applications from 270,000 users on all the 45,000 movies in this dataset can be accessed here

Acknowledgements

This dataset is an ensemble of data collected from TMDB and GroupLens.
The Movie Details, Credits and Keywords have been collected from the TMDB Open API. This product uses the TMDb API but is not endorsed or certified by TMDb. Their API also provides access to data on many additional movies, actors and actresses, crew members, and TV shows. You can try it for yourself here.

The Movie Links and Ratings have been obtained from the Official GroupLens website. The files are a part of the dataset available here

Inspiration

This dataset was assembled as part of my second Capstone Project for Springboard's Data Science Career Track. I wanted to perform an extensive EDA on Movie Data to narrate the history and the story of Cinema and use this metadata in combination with MovieLens ratings to build various types of Recommender Systems.

Both my notebooks are available as kernels with this dataset: The Story of Film and Movie Recommender Systems

Some of the things you can do with this dataset:
Predicting movie revenue and/or movie success based on a certain metric. What movies tend to get higher vote

counts and vote averages on TMDB? Building Content Based and Collaborative Filtering Based Recommendation Engines.

Movie Recommender System Machine Learning Project

This project involves building a movie recommender system using machine learning techniques. Here's a step-by-step guide:

1. Problem Definition

Objective: Develop a movie recommender system that suggests movies to users based on their past behavior and preferences.

2. Data Collection

For this example, we'll use the MovieLens dataset, which is commonly used for movie recommendation systems. You can download it from MovieLens.

3. Data Preprocessing

```
import pandas as pd

# Load the datasets
movies = pd.read_csv('movies.csv')
ratings = pd.read_csv('ratings.csv')

# Display basic info and check for missing values
print(movies.info())
print(ratings.info())

print(movies.head())
print(ratings.head())
```

4. Exploratory Data Analysis (EDA)

```
import seaborn as sns
import matplotlib.pyplot as plt

# Basic statistics
print(ratings.describe())
```

```
# Histogram of ratings
ratings['rating'].hist(bins=30)
plt.title('Distribution of Movie Ratings')
plt.xlabel('Rating')
plt.ylabel('Count')
plt.show()

# Number of ratings per movie
ratings_per_movie = ratings.groupby('movieId').count()['rating']
ratings_per_movie.hist(bins=50)
plt.title('Number of Ratings per Movie')
plt.xlabel('Number of Ratings')
plt.ylabel('Count')
plt.show()
```

5. Building the Recommender System

Collaborative Filtering using Matrix Factorization (SVD)

```
from surprise import Dataset, Reader, SVD
from surprise.model_selection import cross_validate

# Load the data into Surprise format
reader = Reader(rating_scale=(0.5, 5))
data = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], reader)

# Use SVD for matrix factorization
svd = SVD()

# Cross-validation to evaluate the algorithm
cross_validate(svd, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

Training the Model

```
trainset = data.build_full_trainset()
svd.fit(trainset)
```

6. Making Predictions

```
# Predict the rating for a specific user and movie
user_id = 1
movie_id = 10
rating_prediction = svd.predict(user_id, movie_id)
print(f"Predicted rating for user {user_id} and movie {movie_id}: {rating_prediction.est}")
```

7. Recommending Movies

```
# Function to recommend top N movies for a given user
def recommend_movies(user_id, num_recommendations=10):
    # Get a list of all movie ids
    movie_ids = movies['movieId'].unique()

    # Predict ratings for all movies the user hasn't rated yet
    movie_ratings = [svd.predict(user_id, movie_id).est for movie_id in movie_ids]

    # Create a DataFrame of movie ids and predicted ratings
    recommendations = pd.DataFrame({
        'movieId': movie_ids,
        'predicted_rating': movie_ratings
    })

    # Sort the DataFrame by predicted rating in descending order
    recommendations = recommendations.sort_values(by='predicted_rating',
ascending=False)

    # Get the top N recommended movies
    top_recommendations = recommendations.head(num_recommendations)

    # Merge with the movies DataFrame to get movie titles
    top_recommendations = pd.merge(top_recommendations, movies, on='movieId')

    return top_recommendations

# Recommend top 10 movies for user with ID 1
recommendations = recommend_movies(1, 10)
print(recommendations)
```

8. Deployment

To deploy the recommender system, you could create a simple web application using Flask.

```
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route('/recommend', methods=['POST'])
def recommend():
    data = request.get_json(force=True)
    user_id = data['user_id']
    num_recommendations = data.get('num_recommendations', 10)
    recommendations = recommend_movies(user_id, num_recommendations)
    return jsonify(recommendations.to_dict(orient='records'))

if __name__ == '__main__':
    app.run(debug=True)
```

9. Monitoring and Maintenance

Set up logging and monitoring to track the performance of your recommender system, and schedule regular retraining with new data to keep the recommendations relevant.

10. Documentation and Reporting

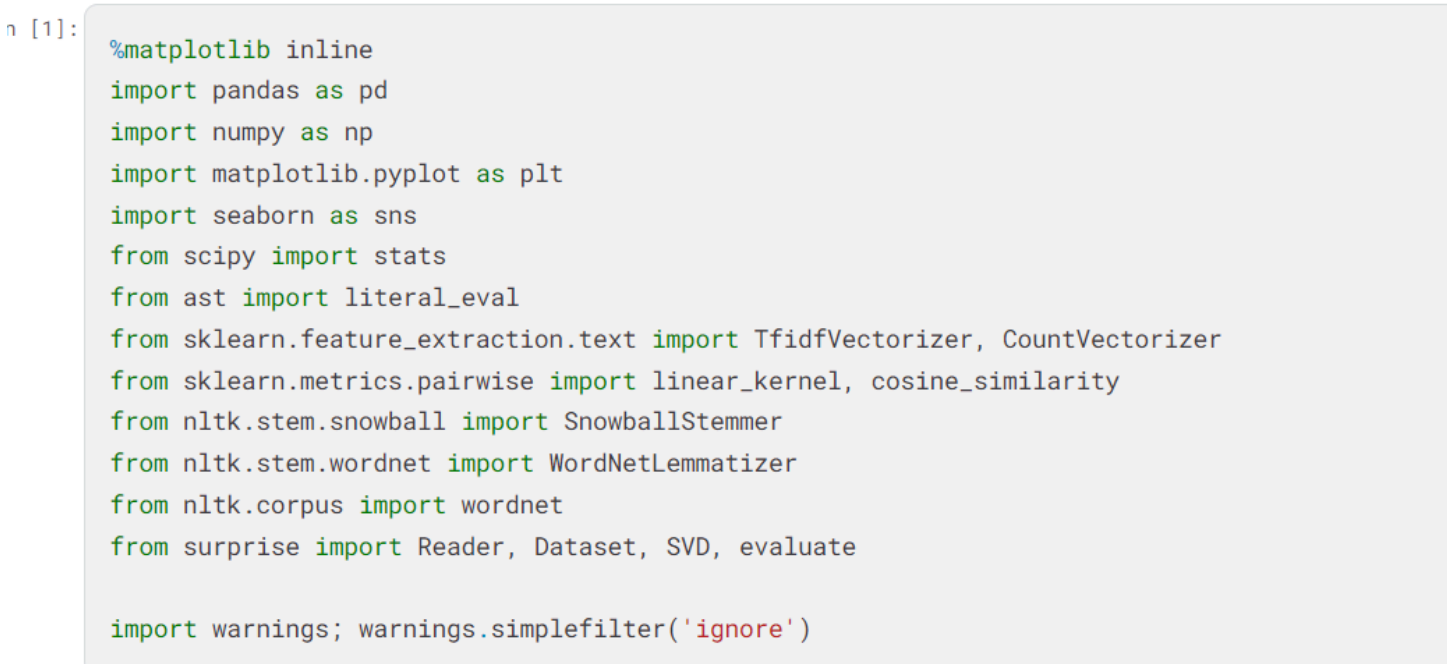
Maintain comprehensive documentation of the project, including data sources, preprocessing steps, model selection, and evaluation results. Create detailed reports and visualizations to communicate findings and insights to stakeholders.

Tools and Technologies

- **Programming Language**: Python
- **Libraries**: pandas, numpy, seaborn, matplotlib, scikit-learn, Surprise, Flask
- **Visualization Tools**: Tableau, Power BI, or any dashboarding tool for advanced visualizations

This is a basic outline of a movie recommender system project. Depending on your specific goals and data, you may need to adjust the steps accordingly.

Sample Project Report



n [1]:

```
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from ast import literal_eval
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.metrics.pairwise import linear_kernel, cosine_similarity
from nltk.stem.snowball import SnowballStemmer
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import wordnet
from surprise import Reader, Dataset, SVD, evaluate

import warnings; warnings.simplefilter('ignore')
```

Simple Recommender¶

The Simple Recommender offers generalized recommnendations to every user based on movie popularity and (sometimes) genre. The basic idea behind this recommender is that movies that are more popular and more critically acclaimed will have a higher probability of being liked by the average audience. This model does not give personalized recommendations based on the user.

The implementation of this model is extremely trivial. All we have to do is sort our movies based on ratings and popularity and display the top movies of our list. As an added step, we can pass in a genre argument to get the top movies of a particular genre.

In [2]:
```
md = pd. read_csv('../input/movies_metadata.csv')
md.head()
```

Out[2]:

	adult	belongs_to_collection	budget	genres	homepage	id	imdb_id	original_language	origin
0	False	{'id': 10194, 'name': 'Toy Story Collection', ...	30000000	[{'id': 16, 'name': 'Animation'}, {'id': 35, '...	http://toystory.disney.com/toy-story	862	tt0114709	en	Toy S
1	False	NaN	65000000	[{'id': 12, 'name': 'Adventure'}, {'id': 14, '...	NaN	8844	tt0113497	en	Juma
2	False	{'id': 119050, 'name': 'Grumpy Old Men Collect...	0	[{'id': 10749, 'name': 'Romance'}, {'id': 35, ...	NaN	15602	tt0113228	en	Grum Old M

In [3]:
```
md['genres'] = md['genres'].fillna('[]').apply(literal_eval).apply(lambda x: [i['name'] for i
in x] if isinstance(x, list) else [])
```

I use the TMDB Ratings to come up with our **Top Movies Chart.** I will use IMDB's *weighted rating* formula to construct my chart. Mathematically, it is represented as follows:

Weighted Rating (WR) = $(\frac{v}{v+m}.R) + (\frac{m}{v+m}.C)$

where,

- v is the number of votes for the movie
- m is the minimum votes required to be listed in the chart
- R is the average rating of the movie
- C is the mean vote across the whole report

The next step is to determine an appropriate value for m, the minimum votes required to be listed in the chart. We will use **95th percentile** as our cutoff. In other words, for a movie to feature in the charts, it must have more votes than at least 95% of the movies in the list.

I will build our overall Top 250 Chart and will define a function to build charts for a particular genre. Let's begin!

In [4]:
```
vote_counts = md[md['vote_count'].notnull()]['vote_count'].astype('int')
vote_averages = md[md['vote_average'].notnull()]['vote_average'].astype('int')
C = vote_averages.mean()
C
```

Out[4]:
```
5.244896612406511
```

In [5]:
```
m = vote_counts.quantile(0.95)
m
```

Out[5]:
```
434.0
```

In [6]:
```
md['year'] = pd.to_datetime(md['release_date'], errors='coerce').apply(lambda x: str(x).split
('-')[0] if x != np.nan else np.nan)
```

In [7]:
```
qualified = md[(md['vote_count'] >= m) & (md['vote_count'].notnull()) & (md['vote_average'].no
tnull())][['title', 'year', 'vote_count', 'vote_average', 'popularity', 'genres']]
qualified['vote_count'] = qualified['vote_count'].astype('int')
qualified['vote_average'] = qualified['vote_average'].astype('int')
qualified.shape
```

Out[7]:
```
(2274, 6)
```

Therefore, to qualify to be considered for the chart, a movie has to have at least **434 votes** on TMDB. We also see that the average rating for a movie on TMDB is **5.244** on a scale of 10. **2274** Movies qualify to be on our chart.

In [8]:
```
def weighted_rating(x):
    v = x['vote_count']
    R = x['vote_average']
    return (v/(v+m) * R) + (m/(m+v) * C)
```

In [9]:
```
qualified['wr'] = qualified.apply(weighted_rating, axis=1)
```

In [10]:
```
qualified = qualified.sort_values('wr', ascending=False).head(250)
```

Top Movies

In [11]:
```
qualified.head(15)
```

Out[11]:

	title	year	vote_count	vote_average	popularity	genres	wr
15480	Inception	2010	14075	8	29.1081	[Action, Thriller, Science Fiction, Mystery, A...	7.917588
12481	The Dark Knight	2008	12269	8	123.167	[Drama, Action, Crime, Thriller]	7.905871
22879	Interstellar	2014	11187	8	32.2135	[Adventure, Drama, Science Fiction]	7.897107
2843	Fight Club	1999	9678	8	63.8696	[Drama]	7.881753
4863	The Lord of the Rings: The Fellowship of the Ring	2001	8892	8	32.0707	[Adventure, Fantasy, Action]	7.871787
292	Pulp Fiction	1994	8670	8	140.95	[Thriller, Crime]	7.868660
314	The Shawshank Redemption	1994	8358	8	51.6454	[Drama, Crime]	7.864000
7000	The Lord of the Rings: The Return of the King	2003	8226	8	29.3244	[Adventure, Fantasy, Action]	7.861927
351	Forrest Gump	1994	8147	8	48.3072	[Comedy, Drama, Romance]	7.860656
5814	The Lord of the Rings: The Two Towers	2002	7641	8	29.4235	[Adventure, Fantasy, Action]	7.851924

Reference link