Documentation: Personalized Movie Recommendation System

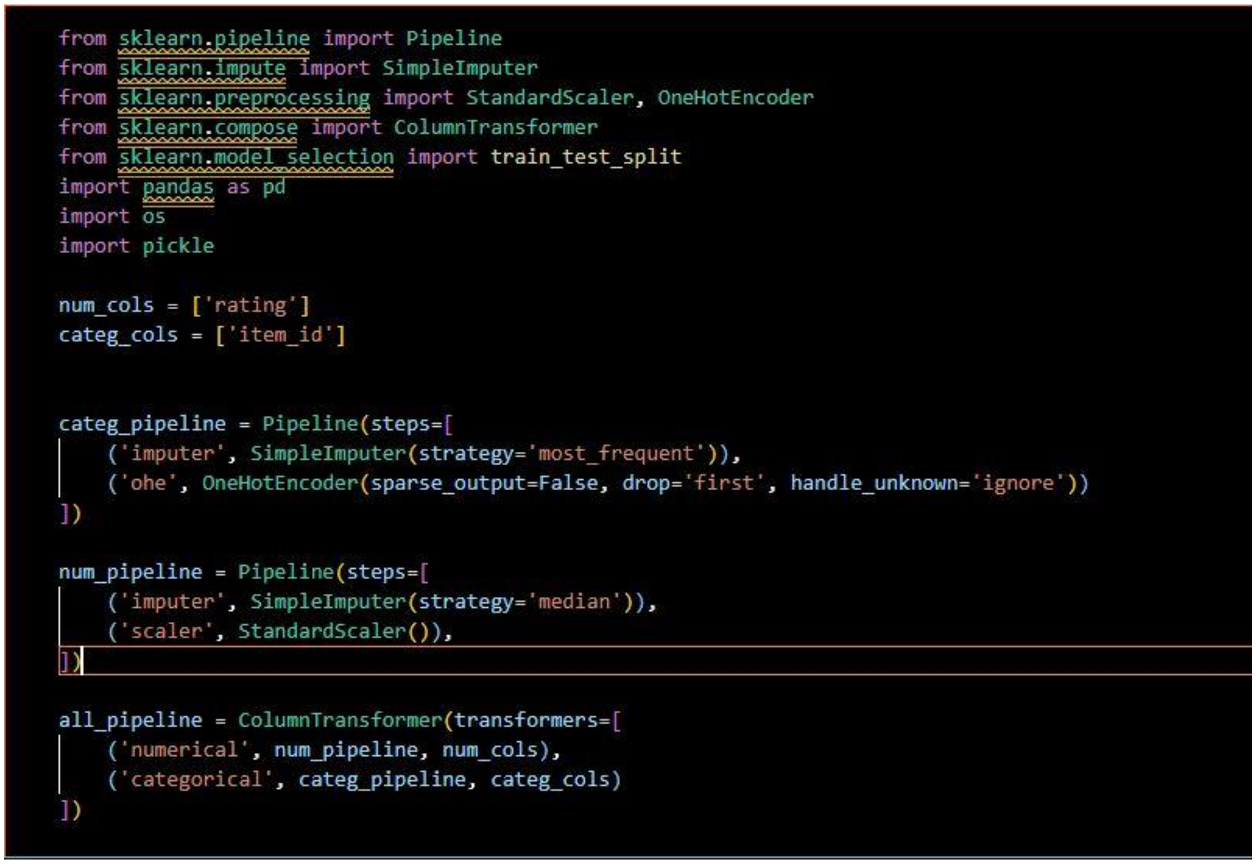
---

1. Introduction:
This documentation outlines the development of the *Personalized Movie Recommendation System*, which uses **Generative Adversarial Networks (GANs)** and **autoencoders** to predict and recommend movies based on user preferences. The system is integrated with **FastAPI** for real-time user requests and responses.

---

2. Data Collection and Preprocessing:
Data preprocessing involves handling numerical and categorical data, such as user ratings and movie IDs. We use **scikit-learn** pipelines to clean and transform the data, preparing it for modeling.

Code Sample for Preprocessing Pipeline:

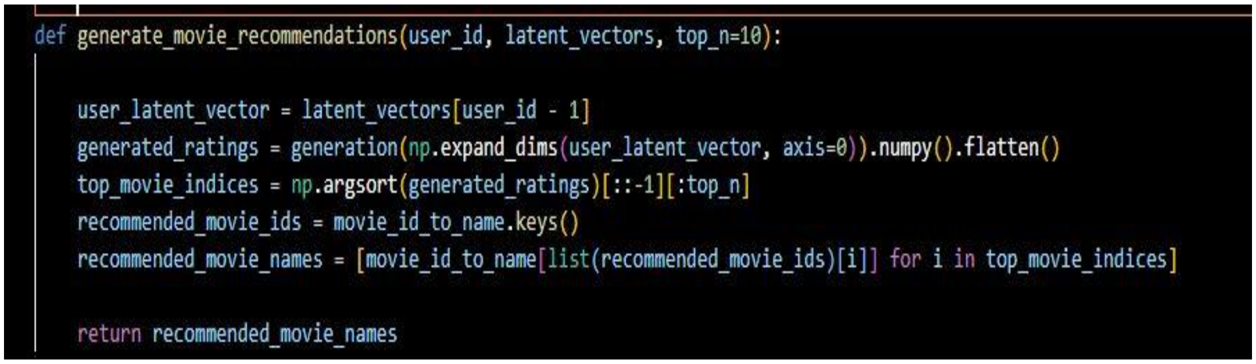


```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
import pandas as pd
import os
import pickle

num_cols = ['rating']
categ_cols = ['item_id']


categ_pipeline = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('ohe', OneHotEncoder(sparse_output=False, drop='first', handle_unknown='ignore'))
])

num_pipeline = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler()),
])

all_pipeline = ColumnTransformer(transformers=[
    ('numerical', num_pipeline, num_cols),
    ('categorical', categ_pipeline, categ_cols)
])
```

---

3. Recommendation Models:
The recommendation models leverage **GANs** and **autoencoders**. The **GAN** generates personalized recommendations by learning user-item matrices, while the **autoencoder** compresses user preferences into latent vectors for more precise predictions.

Code Sample for GAN-based Recommendations:

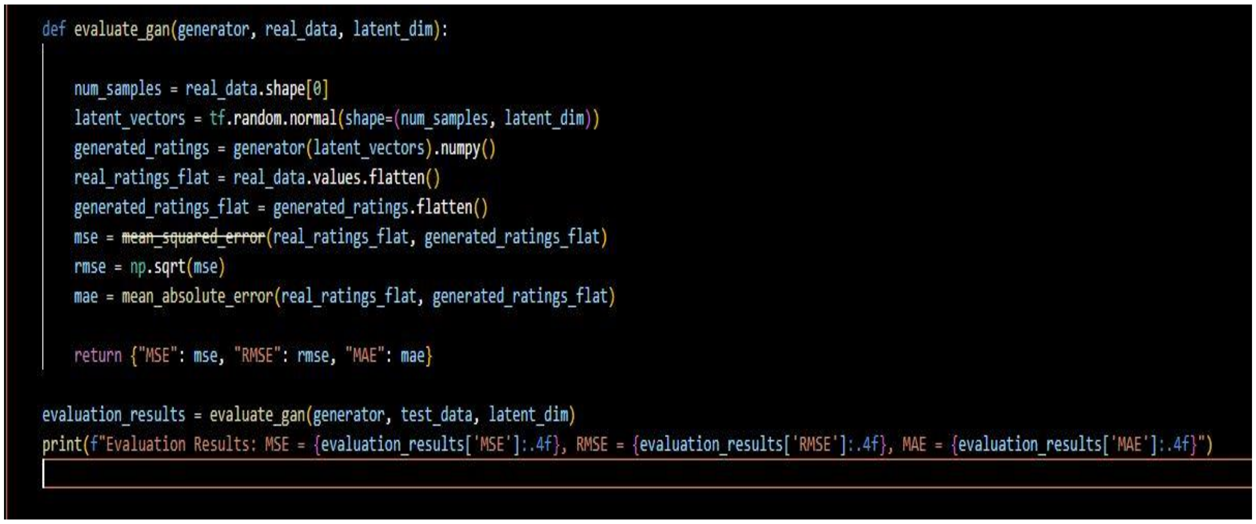


```python
def generate_movie_recommendations(user_id, latent_vectors, top_n=10):

    user_latent_vector = latent_vectors[user_id - 1]
    generated_ratings = generation(np.expand_dims(user_latent_vector, axis=0)).numpy().flatten()
    top_movie_indices = np.argsort(generated_ratings)[::-1][:top_n]
    recommended_movie_ids = movie_id_to_name.keys()
    recommended_movie_names = [movie_id_to_name[list(recommended_movie_ids)[i]] for i in top_movie_indices]

    return recommended_movie_names
```

---

4. Model Evaluation and Optimization:
The models are evaluated using standard metrics such as **MSE**, **RMSE**, and **MAE**. We optimized the GAN and autoencoder models to enhance recommendation accuracy.

Code Sample for Evaluation:



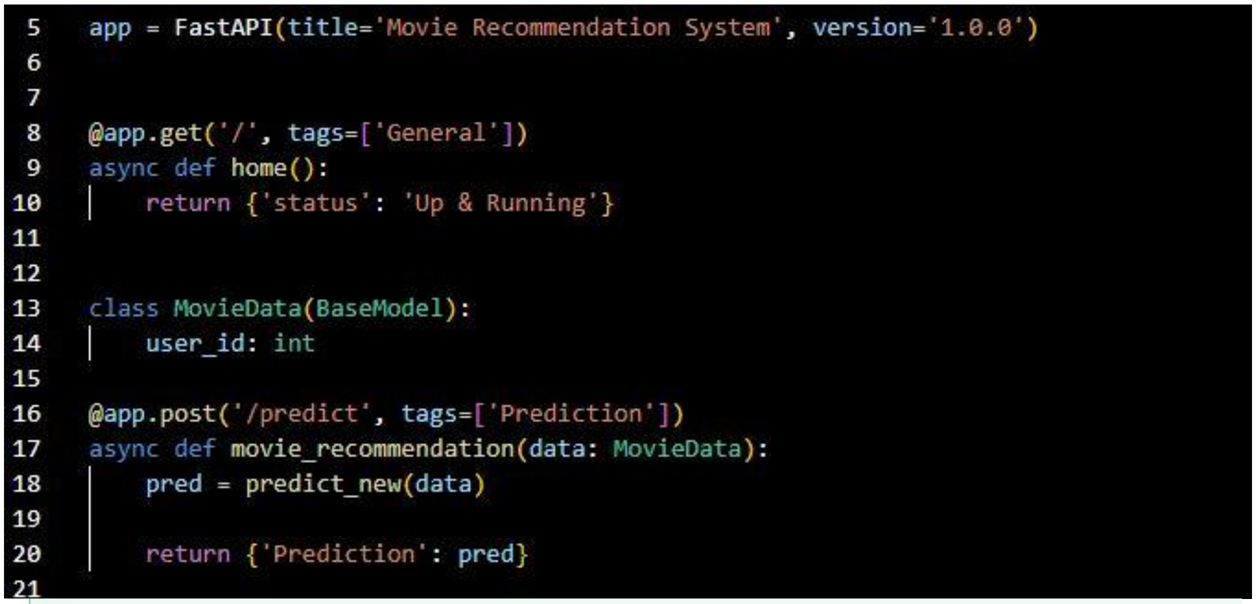
```python
def evaluate_gan(generator, real_data, latent_dim):

    num_samples = real_data.shape[0]
    latent_vectors = tf.random.normal(shape=(num_samples, latent_dim))
    generated_ratings = generator(latent_vectors).numpy()
    real_ratings_flat = real_data.values.flatten()
    generated_ratings_flat = generated_ratings.flatten()
    mse = ~~mean_squared_error~~(real_ratings_flat, generated_ratings_flat)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(real_ratings_flat, generated_ratings_flat)

    return {"MSE": mse, "RMSE": rmse, "MAE": mae}

evaluation_results = evaluate_gan(generator, test_data, latent_dim)
print(f"Evaluation Results: MSE = {evaluation_results['MSE']:.4f}, RMSE = {evaluation_results['RMSE']:.4f}, MAE = {evaluation_results['MAE']:.4f}")
```

---

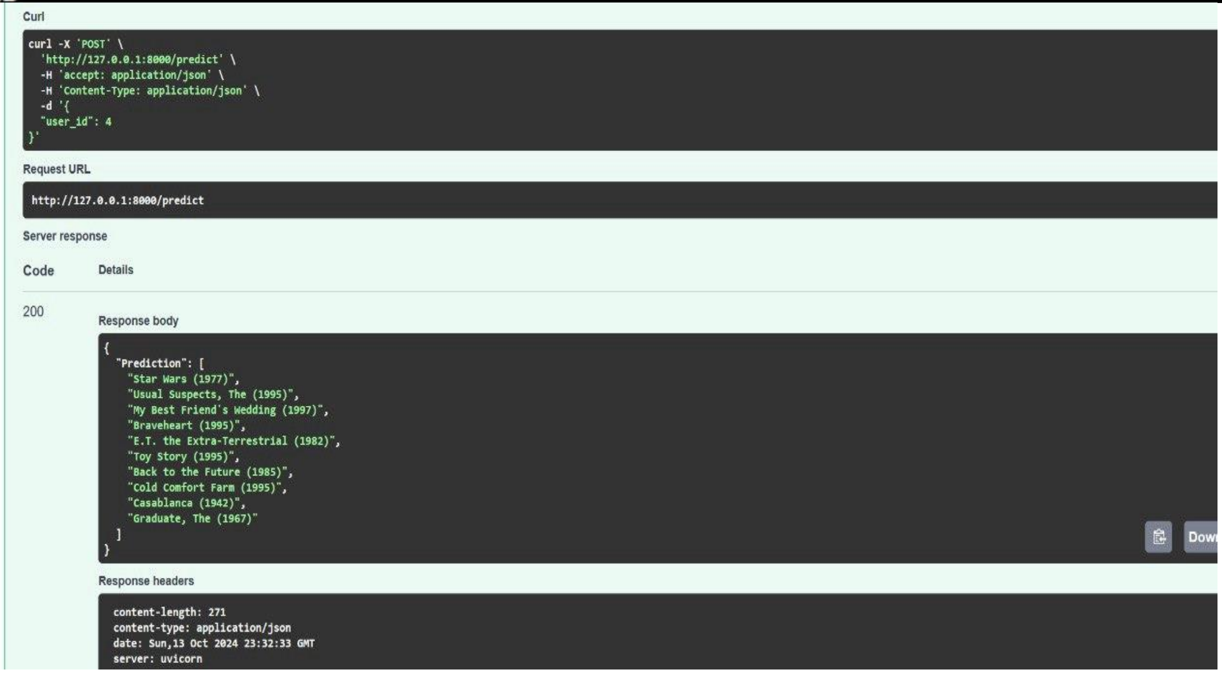
5. API Integration:
The system is deployed using **FastAPI**. The API handles two main endpoints:

- A health check (`/`) to ensure the service is operational.
- A prediction endpoint (`/predict`) to return movie recommendations based on user input (user ID).

Code Sample for FastAPI:



```python
app = FastAPI(title='Movie Recommendation System', version='1.0.0')


@app.get('/', tags=['General'])
async def home():
    return {'status': 'Up & Running'}


class MovieData(BaseModel):
    user_id: int

@app.post('/predict', tags=['Prediction'])
async def movie_recommendation(data: MovieData):
    pred = predict_new(data)

    return {'Prediction': pred}

```



Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/predict' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "user_id": 4
}'
```

Request URL

```
http://127.0.0.1:8000/predict
```

Server response

Code Details

200

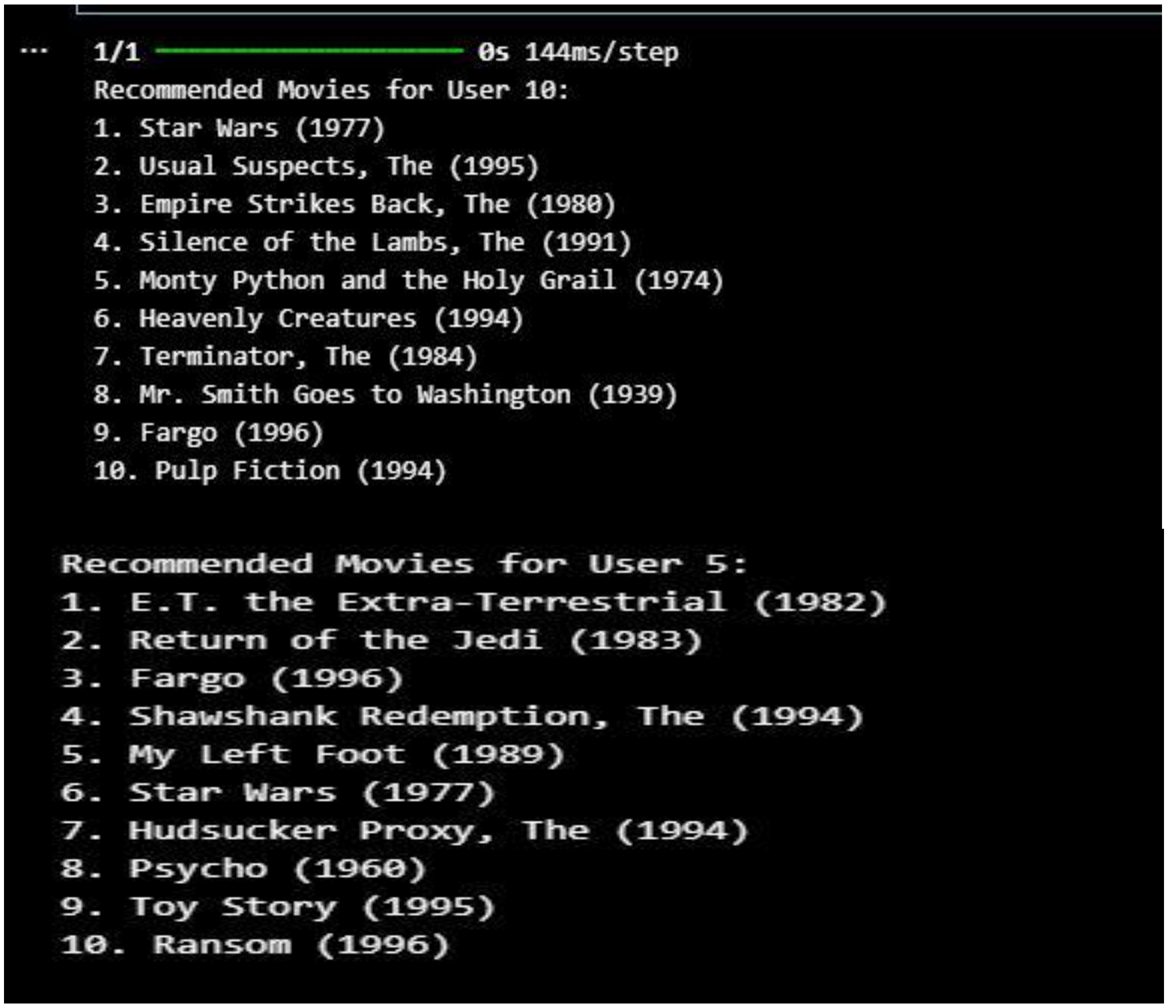
Response body

```
{
  "Prediction": [
    "Star Wars (1977)",
    "Usual Suspects, The (1995)",
    "My Best Friend's Wedding (1997)",
    "Braveheart (1995)",
    "E.T. the Extra-Terrestrial (1982)",
    "Toy Story (1995)",
    "Back to the Future (1985)",
    "Cold Comfort Farm (1995)",
    "Casablanca (1942)",
    "Graduate, The (1967)"
  ]
}
```

Response headers

```
content-length: 271
content-type: application/json
date: Sun,13 Oct 2024 23:32:33 GMT
server: uvicorn
```

6.Test samples



```
...  1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 144ms/step
     Recommended Movies for User 10:
     1. Star Wars (1977)
     2. Usual Suspects, The (1995)
     3. Empire Strikes Back, The (1980)
     4. Silence of the Lambs, The (1991)
     5. Monty Python and the Holy Grail (1974)
     6. Heavenly Creatures (1994)
     7. Terminator, The (1984)
     8. Mr. Smith Goes to Washington (1939)
     9. Fargo (1996)
     10. Pulp Fiction (1994)

   Recommended Movies for User 5:
   1. E.T. the Extra-Terrestrial (1982)
   2. Return of the Jedi (1983)
   3. Fargo (1996)
   4. Shawshank Redemption, The (1994)
   5. My Left Foot (1989)
   6. Star Wars (1977)
   7. Hudsucker Proxy, The (1994)
   8. Psycho (1960)
   9. Toy Story (1995)
   10. Ransom (1996)
```

---

7. Conclusion:
The *Personalized Movie Recommendation System* uses advanced deep learning techniques such as **GANs** and **autoencoders** to provide personalized movie recommendations. By integrating with **FastAPI**, the system can deliver real-time recommendations to users.