Análise de Sistema de Recomendação - Organização em Blocos

Bloco 1 - Preparação de Dados

```
# Importação de bibliotecas e leitura dos dados MovieLens
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats

from surprise import Dataset, Reader, SVD, accuracy
from surprise.model_selection import train_test_split, cross_validate

# Carrega até 1 milhão de linhas da base de avaliações
baseRatings = pd.read_csv('ratings.csv', nrows=1000000)
baseMovies = pd.read_csv('movies.csv')  # contém os nomes dos filmes
```

Bloco 2 - Preparação para o modelo SVD

```
# Define o intervalo de ratings e transforma o DataFrame para o formato do Surprise
reader = Reader(rating_scale=(baseRatings['rating'].min(), baseRatings['rating'].max()))
data = Dataset.load_from_df(baseRatings[['userId', 'movieId', 'rating']], reader)

# Divide os dados em treino e teste
trainset, testset = train_test_split(data, test_size=0.2, random_state=42)
```

Bloco 3 - Treinamento e Avaliação (Regressão com SVD)

```
# Treina o modelo SVD nos dados de treino
model = SVD(random_state=42)
model.fit(trainset)

# Realiza previsões no conjunto de teste
predictions = model.test(testset)

# Avalia com métricas de regressão
print("\nAvaliação do Modelo SVD (Regressão):")
rmse_val = accuracy.rmse(predictions)
mae_val = accuracy.mae(predictions)
```

Bloco 4 - Validação Cruzada

```
# Validação cruzada com 5 folds para avaliar estabilidade do modelo
print("\nValidação Cruzada (Regressão):")
cv_results = cross_validate(model, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
print(cv_results)
```

Bloco 5 - Visualizações (Regressão)

```
# Prepara arrays com valores reais e previstos
true_ratings = np.array([pred.r_ui for pred in predictions])
pred_ratings = np.array([pred.est for pred in predictions])
```

```
errors = np.abs(true_ratings - pred_ratings)

# Gráfico: Notas Reais vs Notas Previstas
plt.figure(figsize=(8,6))
sns.scatterplot(x=true_ratings, y=pred_ratings, alpha=0.4)
plt.plot([0, 5], [0, 5], '--', color='red')
plt.title('Notas Reais vs. Notas Previstas (SVD)')
plt.xlabel('Nota Real')
plt.ylabel('Nota Prevista')
plt.grid(True)
plt.show()

# Gráfico: Histograma dos Erros Absolutos
plt.figure(figsize=(8,5))
sns.histplot(errors, bins=30, kde=True, color='orange')
plt.title('Distribuição dos Erros Absolutos (|real - previsto|)')
plt.xlabel('Erro Absoluto')
plt.ylabel('Frequência')
plt.grid(True)
plt.show()
```

Bloco 6 - Conversão para Classificação e Avaliação

```
# Define categorias de rating
bins = [0, 3, 6, 10]
labels = ['baixa', 'média', 'alta']
true_categories = pd.cut(true_ratings, bins=bins, labels=labels, include_lowest=True)
pred_categories = pd.cut(pred_ratings, bins=bins, labels=labels, include_lowest=True)

# Calcula métricas de classificação
from sklearn.metrics import accuracy_score, f1_score, recall_score, precision_score,
confusion_matrix

accuracy_cat = accuracy_score(true_categories, pred_categories)
precision_cat = precision_score(true_categories, pred_categories, average='weighted')
f1_cat = f1_score(true_categories, pred_categories, average='weighted')
recall_cat = recall_score(true_categories, pred_categories, average='weighted')

# Matriz de confusão
conf_matrix = confusion_matrix(true_categories, pred_categories, labels=labels)
conf_matrix_df = pd.DataFrame(conf_matrix, index=labels, columns=labels)
plt.figure(figsize=(6,5))
sns.heatmap(conf_matrix_df, annot=True, fmt="d", cmap="Blues")
plt.title('Matriz de Confusão (Categorias de Rating)')
plt.xlabel('Predição')
plt.ylabel('Real')
plt.show()
```

Bloco 7 - Análise dos Resíduos

```
# Resíduos: diferença entre nota real e prevista
```

Análise de Sistema de Recomendação - Organização em Blocos

```
residuals = true_ratings - pred_ratings

# Histograma dos resíduos
plt.figure(figsize=(8,5))
plt.hist(residuals, bins=20, color='skyblue', edgecolor='black')
plt.xlabel('Resíduos (Real - Previsto)')
plt.ylabel('Frequência')
plt.title('Histograma dos Resíduos')
plt.show()

# Q-Q plot dos resíduos para verificar normalidade
plt.figure(figsize=(6,6))
stats.probplot(residuals, dist="norm", plot=plt)
plt.title('Q-Q plot dos Resíduos')
plt.show()
```

Bloco 8 - Recomendação Baseada em Usuário

```
# Função de recomendação colaborativa baseada em similaridade entre usuários
def recommend_movies_user_based(user_id, ratings_matrix, user_similarity_df, baseMovie, top_n=5):
    if user_id not in ratings_matrix.index:
        return f"Usuário {user_id} não encontrado na amostra."

    sim_scores = user_similarity_df[user_id]
    ratings = ratings_matrix.copy()
    user_rated = ratings.loc[user_id][ratings.loc[user_id] > 0].index
    ratings = ratings.drop(columns=user_rated)

    weighted_ratings = ratings.T.dot(sim_scores) / sim_scores.sum()
    recommended_movie_ids = weighted_ratings.sort_values(ascending=False).head(top_n).index
    movie_names = baseMovie[baseMovie['movieId'].isin(recommended_movie_ids)]

    return movie_names[['movieId', 'title']]
```

Bloco 9 - Geração das Recomendações

```
# Gera recomendações para um usuário da amostra
user_id = ratings_matrix.index[13]  # pega o primeiro usuário da amostra
print(f"Recomendações para o usuário {user_id}:")
print(recommend_movies_user_based(user_id, ratings_matrix, user_similarity_df, baseMovie))
```