```
source kaggle/tmdb-movies-dataset-2023-930k-movies
```

Kaggle hosts thousands of datasets for data science and machine learning competitions. It provides crowdsourced solutions to tackle real-world problems. The goal is to develop predictive models that generalize well to unseen data. Kaggle's platform includes datasets, notebooks, and competitions, fostering collaboration and innovation in data science and machine learning.

Predictive modeling uses statistical techniques to analyze historical data and develop a model that accurately estimates the rating given to a movie by users or critics. Movie Rating Prediction project enables you to explore data preprocessing, feature engineering, and various machine learning algorithms. It provides insights into the factors that influence movie ratings and allows you to build a model that can estimate the rating of unseen movies accurately.

```
!pip install -qq -U gdown
!pip install -qqq -U kaggle
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!ls ~/.kaggle
!chmod 600 /root/.kaggle/kaggle.json
!kaggle datasets download -d tarekmasryo/tmdb-930k-movies-dataset
!unzip -qq -o tmdb-930k-movies-dataset.zip
!rm -f tmdb-930k-movies-dataset.zip
!ls -lh
import gdown, os, glob
!gdown --id 1Ln8ZwCzqCqPCxZ0ZZ_Ed_Ldb_Ldb_Ldb -O kaggle.json
os.makedirs('/root/.kaggle', exist_ok=True)
!cp kaggle.json /root/.kaggle/
!chmod 600 /root/.kaggle/kaggle.json
!ls ~/.kaggle
!kaggle datasets download -d tarekmasryo/tmdb-930k-movies-dataset
!unzip -qq -o tmdb-930k-movies-dataset.zip
!rm -f tmdb-930k-movies-dataset.zip
!ls -lh
import os, glob, pandas as pd, numpy as np, matplotlib.pyplot as plt, seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from scipy.sparse import hstack, csr_matrix
import joblib, warnings, gc
warnings.filterwarnings("ignore")
%matplotlib inline
sns.set_theme(style="whitegrid")
pd.pd.set_option("display.max_columns", 200)
pd.pd.set_option("display.max_rows", 100)
print("✅ Imports loaded successfully")
import os, glob, pandas as pd, numpy as np, matplotlib.pyplot as plt, seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from scipy.sparse import hstack, csr_matrix
import joblib, warnings, gc
warnings.filterwarnings("ignore")
%matplotlib inline
sns.set_theme(style="whitegrid")
pd.pd.set_option("display.max_columns", 200)
pd.pd.set_option("display.max_rows", 100)
print("✅ Imports loaded successfully")
# Find CSV files
csv_files = glob.glob("*.csv")
print("CSV files found:", csv_files)
# Load the first CSV file
df = pd.read_csv(csv_files[0], encoding='latin-1')
print("Shape:", df.shape)
df.head()
print("Total columns:", len(df.columns))
df.columns.tolist()
# Step 2: Clean and prepare data
df['release_date'] = pd.to_datetime(df['release_date'], errors='coerce')
df['release_year'] = df['release_date'].dt.year
df['vote_average'] = df['vote_average'].fillna(df['vote_average'].median())
df['vote_count'] = df['vote_count'].fillna(0)
df['popularity'] = df['popularity'].fillna(df['popularity'].median())
df['runtime'] = df['runtime'].fillna(df['runtime'].median())
df['adult'] = df['adult'].fillna(False)
print("✅ Cleaned data")
# Step 3: Create features
df['genre_count'] = df['genres'].fillna('').apply(lambda x: len(x.split(',')) if x != '' else 0)
df['has_poster'] = df['poster_path'].notna()
df['has_backdrop'] = df['backdrop_path'].notna()
df['is_popular'] = (df['popularity'] > df['popularity'].quantile(0.75))
df['is_recent'] = (df['release_year'] >= 2020)
df['is_classic'] = (df['release_year'] <= 1980)
df['is_long'] = (df['runtime'] > 120)
print("✅ Features created")
# Step 4: Prepare for modeling
target = 'vote_average'
features = ['genre_count', 'has_poster', 'has_backdrop', 'is_popular', 'is_recent', 'is_classic', 'is_long', 'runtime', 'popularity', 'vote_count']
X = df[features]
y = df[target]
print("Target:", target)
print("Features:", features)
print("Shape:", X.shape)
# Step 5: Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print("Train:", X_train.shape)
print("Test:", X_test.shape)
# Step 6: Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
print("✅ Scaled")
# Step 7: Train model
model = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=-1)
model.fit(X_train_scaled, y_train)
print("✅ Model trained")
# Step 8: Evaluate
y_pred = model.predict(X_test_scaled)
mae = mean_absolute_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred)**0.5
r2 = r2_score(y_test, y_pred)
print("📊 Results:")
print("MAE:", mae)
print("RMSE:", rmse)
print("R²:", r2)
# Step 9: Feature importance
importance = pd.DataFrame({'feature': features, 'importance': model.feature_importances_})
importance = importance.sort_values('importance', ascending=False)
print("📊 Feature importance:")
print(importance)
# Step 10: Save model
import joblib
joblib.dump(model, 'model.pkl')
joblib.dump(scaler, 'scaler.pkl')
print("✅ Saved")
# Step 11: Predict function
def predict(genre_count, has_poster, has_backdrop, is_popular, is_recent, is_classic, is_long, runtime, popularity, vote_count):
    data = [[genre_count, has_poster, has_backdrop, is_popular, is_recent, is_classic, is_long, runtime, popularity, vote_count]]
    return model.predict(scaler.transform(data))[0]
print("✅ Function ready")
# Test it
print("Test prediction:", predict(2, True, True, True, True, False, False, 120, 50, 1000))
# Step 12: Visualize
import matplotlib.pyplot as plt, seaborn as sns
plt.figure(figure=(12, 6))
sns.scatterplot(x=y_test, y=y_pred, hue=X_test['genre_count'], palette='viridis', alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual vs Predicted')
plt.legend(title='Genre')
plt.show()
```

```
Saved model and preprocessor successfully!
```

DOCUMENTATION

Check dataset: https://www.kaggle.com/datasets/guru-001/tmdb-930k-movies-dataset

Check columns: Validate the presence of the Rating column.

Drop columns: Remove unnecessary columns (e.g., id, title, and release_date).

Fill missing values: Impute missing Rating and categorical values.

Create dates: Extract release_year from release_date and drop the original.

Separate data: Divide the dataset into features (X) and target variable (y).

Preprocessing: Apply scaling to numeric features and one-hot encoding to categorical features.

Split data: Split the dataset into training and testing sets.