Book Recommendation System Using Goodreads Interactions Dataset

Berkehan Göktürk
Azra İrem Dalhasanoğlu
Beyzanur Elma

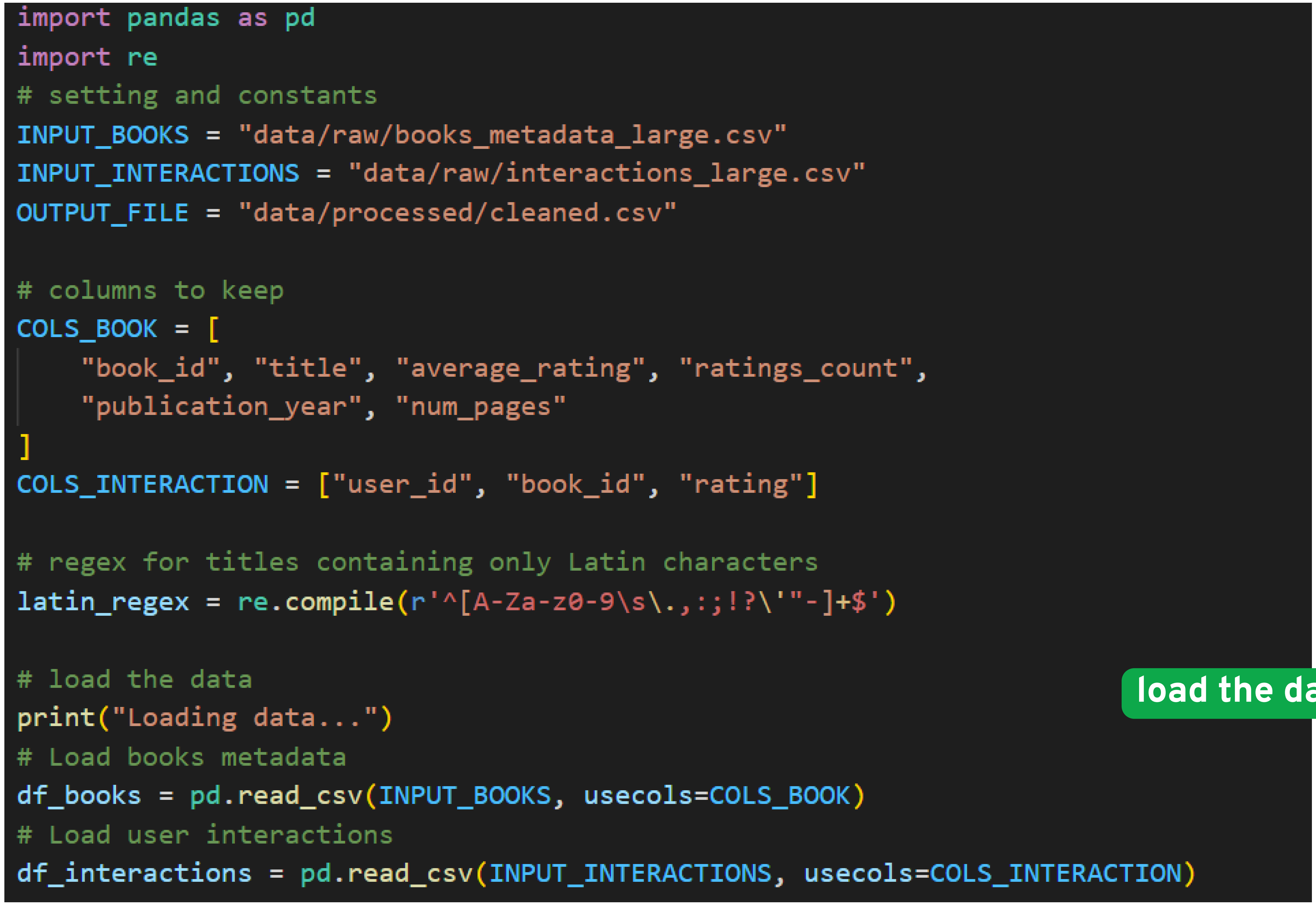
Dataset

In this study, we utilized the **Goodreads Interactions Dataset to develop a book recommendation system**. This dataset captures the interactions of registered Goodreads users with books — such as reading, rating, and shelving activities — and contains both user-item historical interaction data as well as metadata related to the books.

Dataset Analysis, Cleaning, and Merging

The dataset consists of two main files: interactions and books metadata. It contains numerous features such as user_id, book_id, publication_year, num_pages, and many others. In line with our project objectives, we **preprocessed the data** by removing irrelevant features, handling missing values either by imputation or deletion, and correcting characters to comply with UTF-8 encoding standards. Finally, we merged the two files to create a unified dataset, which served as the input for model training.

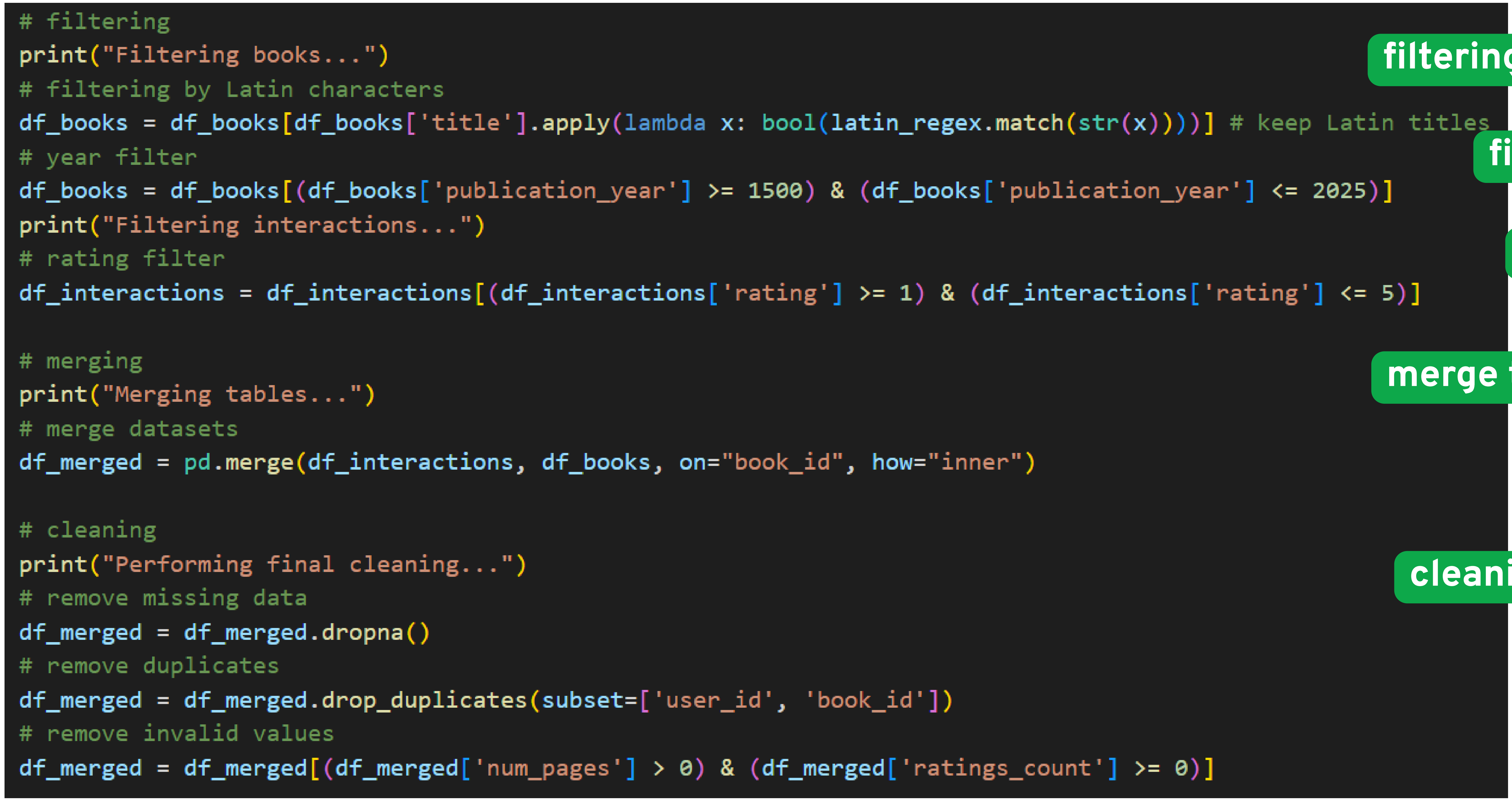
Dataset Analysis, Cleaning, and Merging



```python
import pandas as pd
import re
# setting and constants
INPUT_BOOKS = "data/raw/books_metadata_large.csv"
INPUT_INTERACTIONS = "data/raw/interactions_large.csv"
OUTPUT_FILE = "data/processed/cleaned.csv"

# columns to keep
COLS_BOOK = [
    "book_id", "title", "average_rating", "ratings_count",
    "publication_year", "num_pages"
]
COLS_INTERACTION = ["user_id", "book_id", "rating"]

# regex for titles containing only Latin characters
latin_regex = re.compile(r'^[A-Za-z0-9\s\.,:;!?\'"-]+$')

# load the data
print("Loading data...")
# Load books metadata
df_books = pd.read_csv(INPUT_BOOKS, usecols=COLS_BOOK)
# Load user interactions
df_interactions = pd.read_csv(INPUT_INTERACTIONS, usecols=COLS_INTERACTION)
```

load the data

Dataset Analysis, Cleaning, and Merging



```python
# filtering
print("Filtering books...")
# filtering by Latin characters
df_books = df_books[df_books['title'].apply(lambda x: bool(latin_regex.match(str(x))))] # keep Latin titles
# year filter
df_books = df_books[(df_books['publication_year'] >= 1500) & (df_books['publication_year'] <= 2025)]
print("Filtering interactions...")
# rating filter
df_interactions = df_interactions[(df_interactions['rating'] >= 1) & (df_interactions['rating'] <= 5)]

# merging
print("Merging tables...")
# merge datasets
df_merged = pd.merge(df_interactions, df_books, on="book_id", how="inner")

# cleaning
print("Performing final cleaning...")
# remove missing data
df_merged = df_merged.dropna()
# remove duplicates
df_merged = df_merged.drop_duplicates(subset=['user_id', 'book_id'])
# remove invalid values
df_merged = df_merged[(df_merged['num_pages'] > 0) & (df_merged['ratings_count'] >= 0)]
```

filtering by Latin characters

filtering by year

filtering by rating

merge the tables

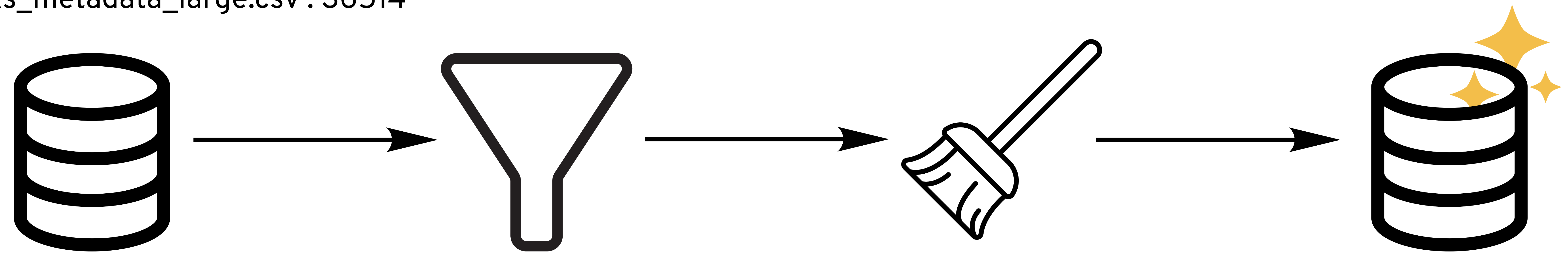
cleaning

Merge* interactions with books on 'book_id'.
Cleaning* remove missing data, duplications and outliers

the amount of raw data

interactions_large.csv : 2734350

books_metadata_large.csv : 36514



the amount of cleaned data

901670

Model Development: Approaches, Experiments, and Results

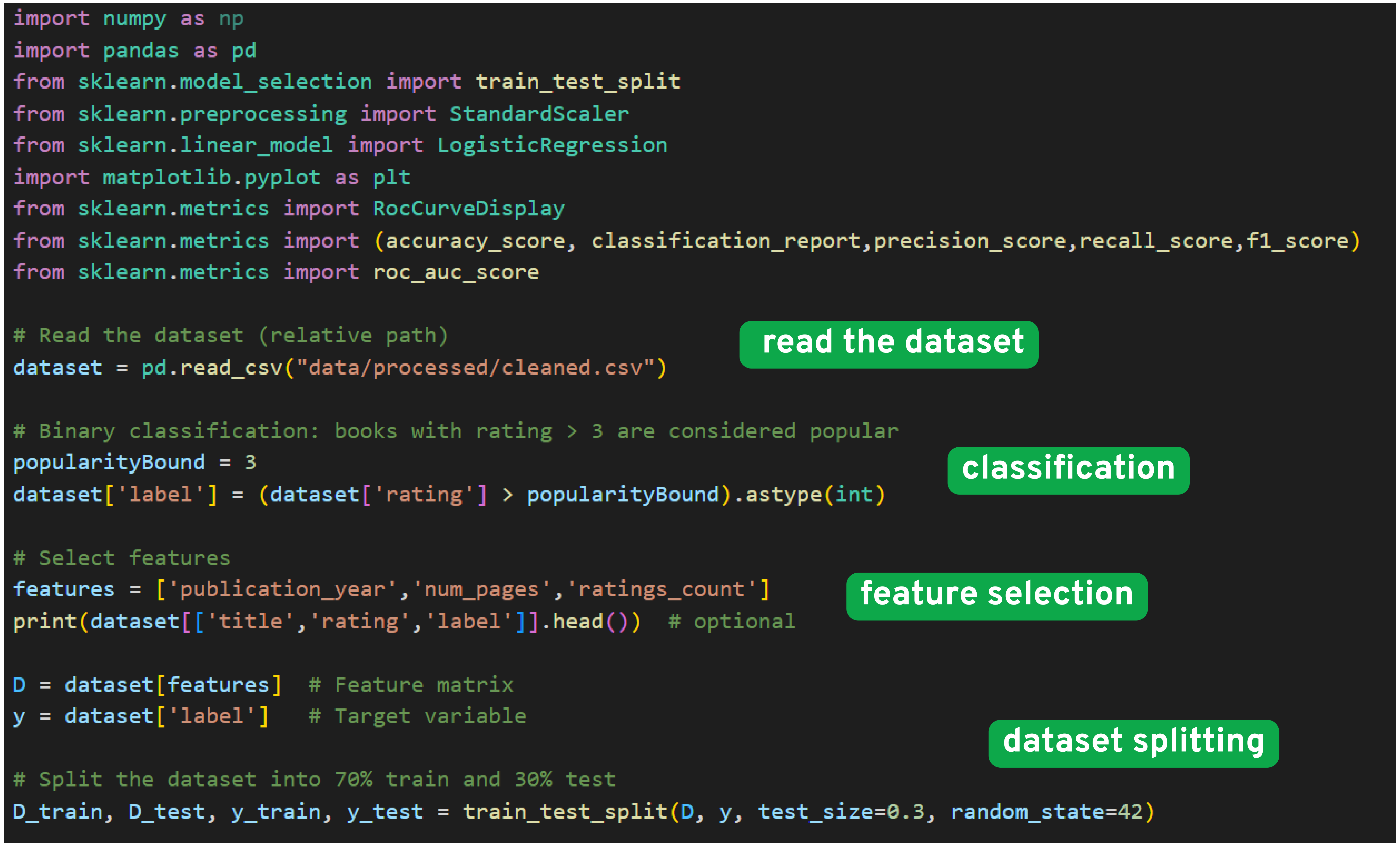
Model Development

- Read the dataset
- Classificaiton
- Feature selection
- Split into training and test sets
- **Model creating and training**
- Predictions and evaluation

Model creating and training

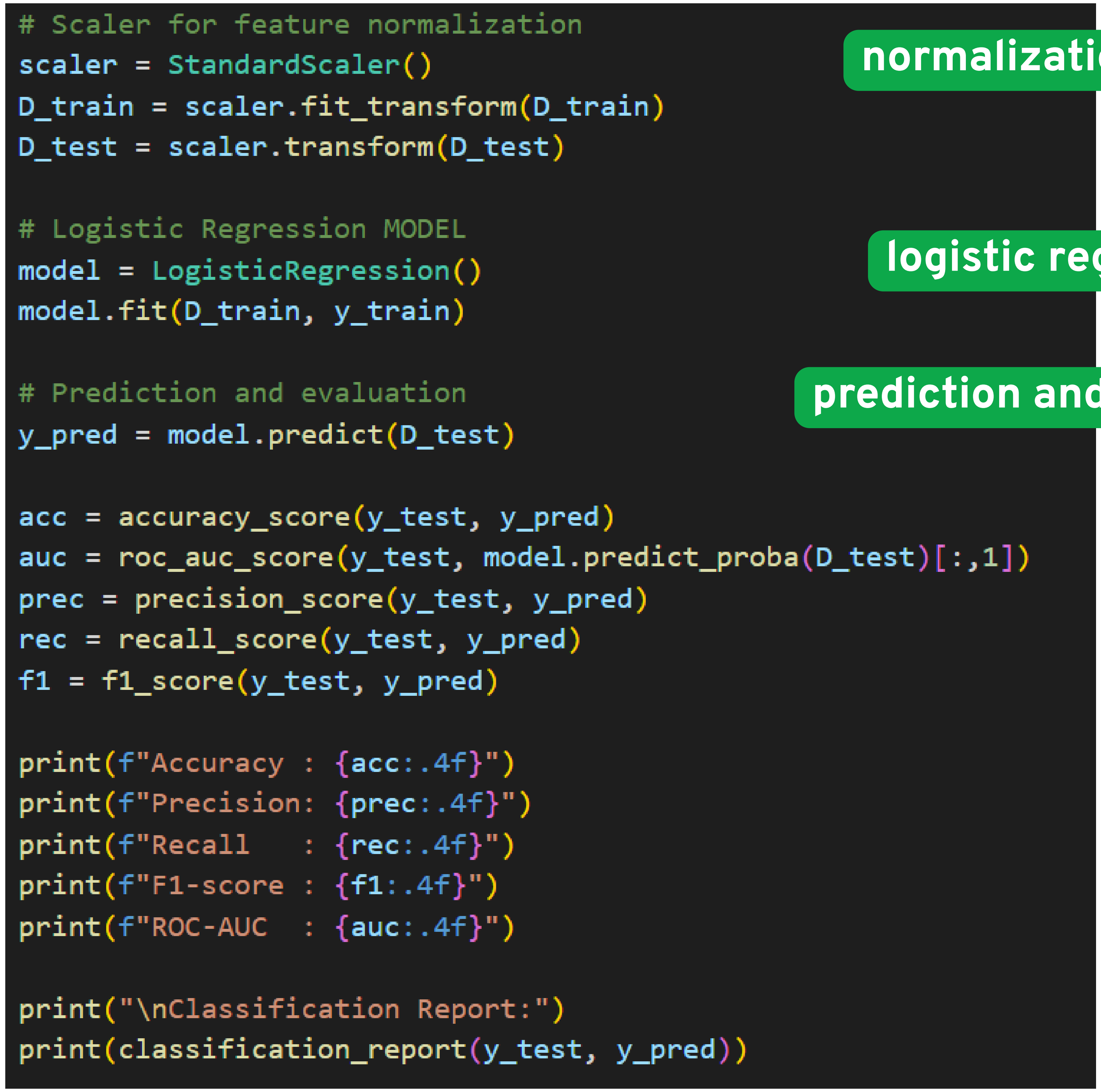
- Logistic Regression
- Random Forest
- Multilayer Perceptron
- Boosting
- Stacking Ensemble
- SVD

try 1 : Logistic Regression



```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import (accuracy_score, classification_report,precision_score,recall_score,f1_score)
from sklearn.metrics import roc_auc_score

# Read the dataset (relative path)
dataset = pd.read_csv("data/processed/cleaned.csv")

# Binary classification: books with rating > 3 are considered popular
popularityBound = 3
dataset['label'] = (dataset['rating'] > popularityBound).astype(int)

# Select features
features = ['publication_year','num_pages','ratings_count']
print(dataset[['title','rating','label']].head())  # optional

D = dataset[features]  # Feature matrix
y = dataset['label']   # Target variable

# Split the dataset into 70% train and 30% test
D_train, D_test, y_train, y_test = train_test_split(D, y, test_size=0.3, random_state=42)
```

try 1 : Logistic Regression



```
# Scaler for feature normalization
scaler = StandardScaler()
D_train = scaler.fit_transform(D_train)
D_test = scaler.transform(D_test)

# Logistic Regression MODEL
model = LogisticRegression()
model.fit(D_train, y_train)

# Prediction and evaluation
y_pred = model.predict(D_test)

acc = accuracy_score(y_test, y_pred)
auc = roc_auc_score(y_test, model.predict_proba(D_test)[:,1])
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Accuracy : {acc:.4f}")
print(f"Precision: {prec:.4f}")
print(f"Recall   : {rec:.4f}")
print(f"F1-score : {f1:.4f}")
print(f"ROC-AUC  : {auc:.4f}")

print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

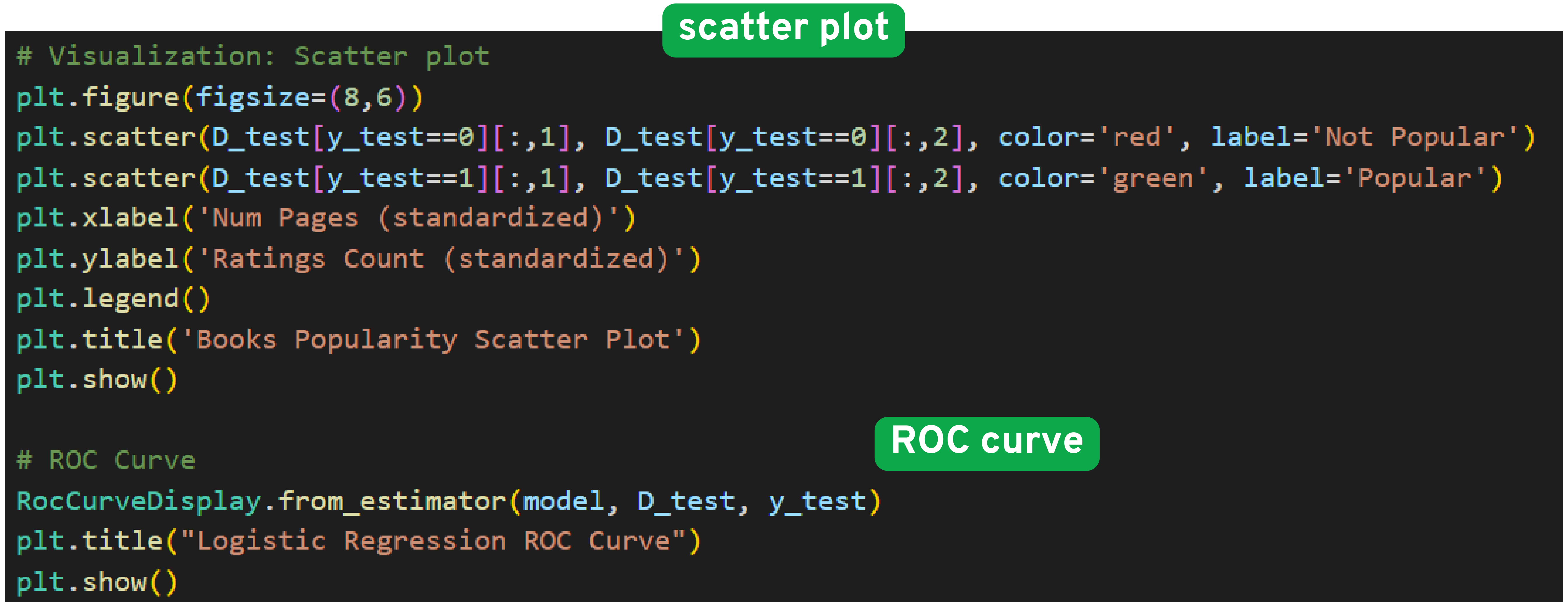
normalization

logistic regression model

prediction and accuracy

Normalization* to ensure all features contribute equally to the model.

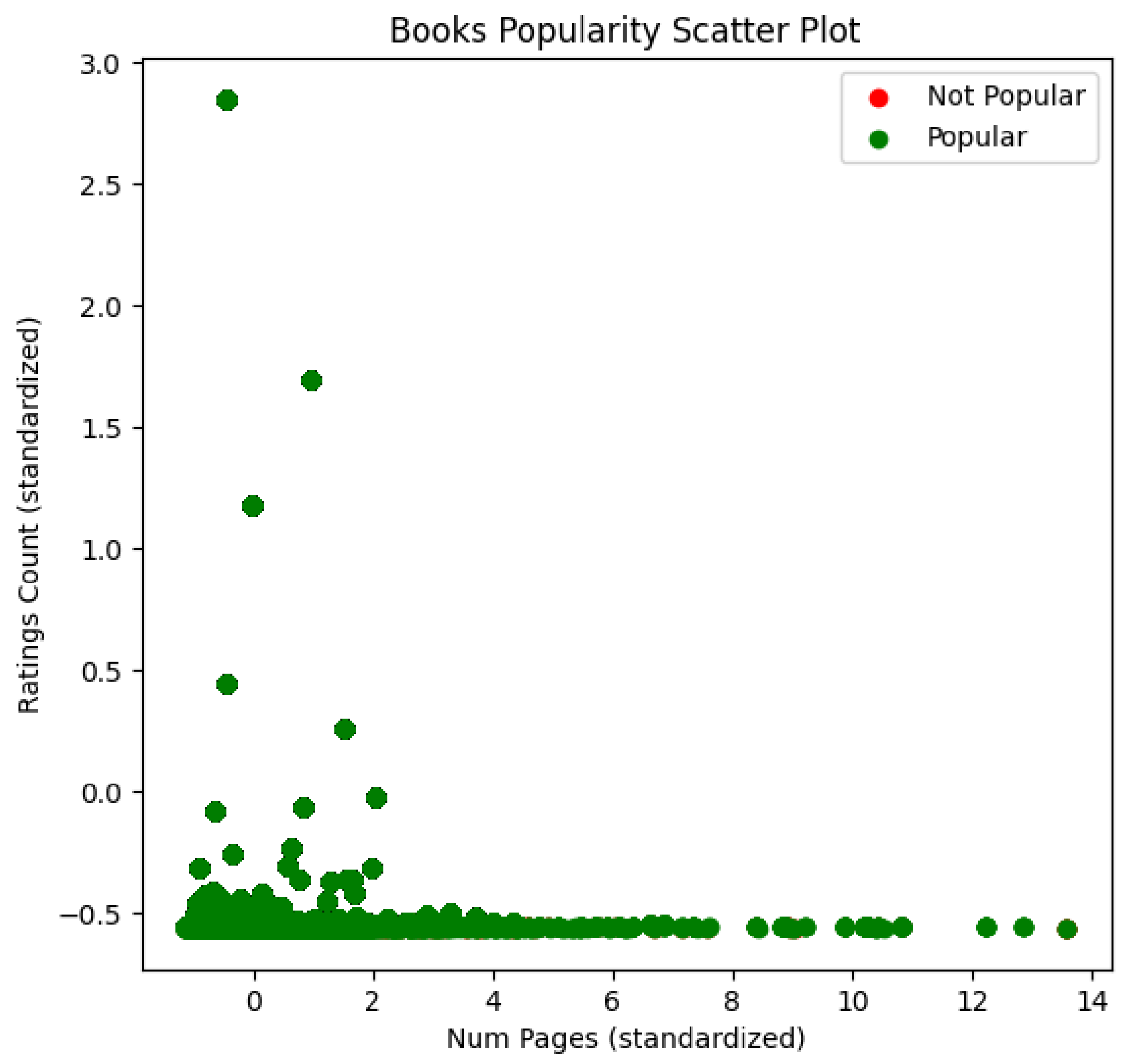
try 1 : Logistic Regression



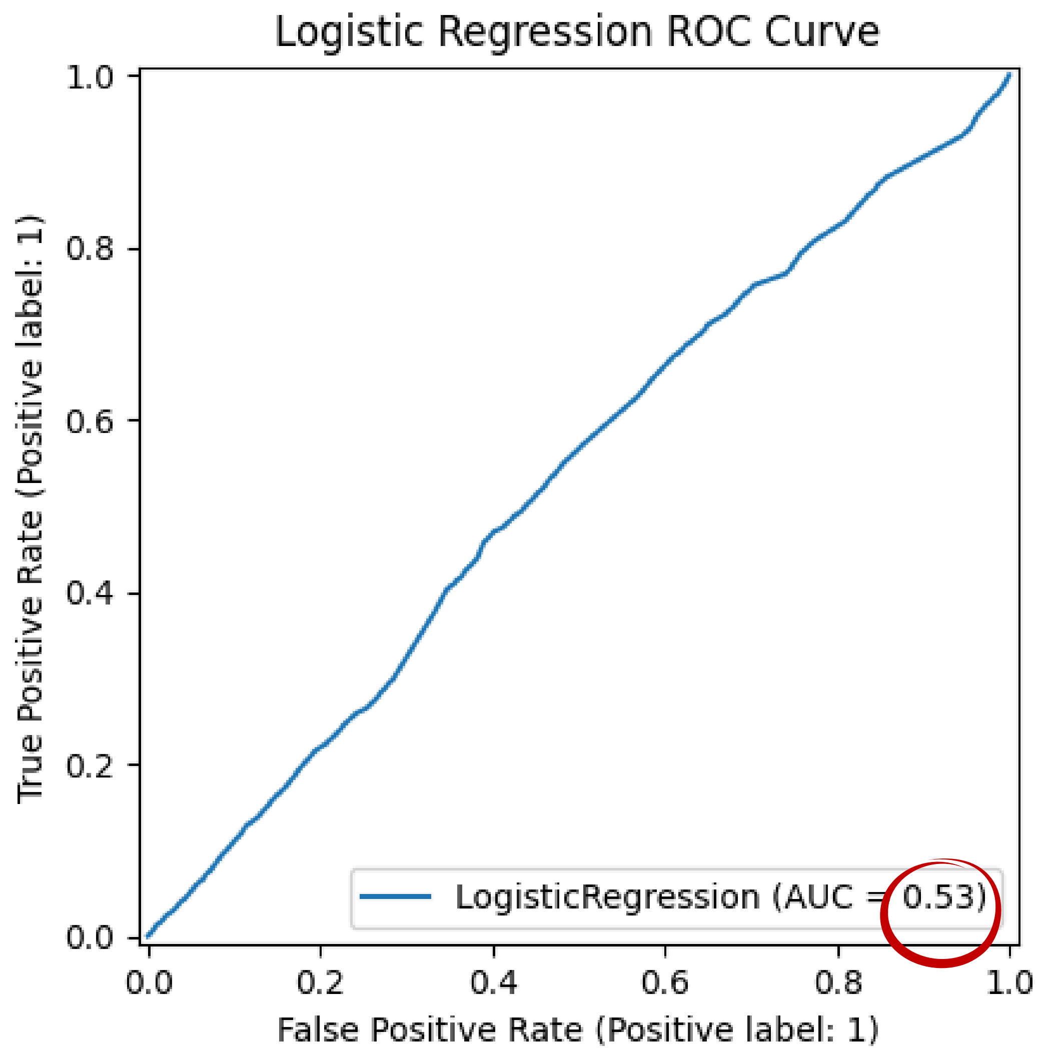
```
# Visualization: Scatter plot
plt.figure(figsize=(8,6))
plt.scatter(D_test[y_test==0][:,1], D_test[y_test==0][:,2], color='red', label='Not Popular')
plt.scatter(D_test[y_test==1][:,1], D_test[y_test==1][:,2], color='green', label='Popular')
plt.xlabel('Num Pages (standardized)')
plt.ylabel('Ratings Count (standardized)')
plt.legend()
plt.title('Books Popularity Scatter Plot')
plt.show()

# ROC Curve
RocCurveDisplay.from_estimator(model, D_test, y_test)
plt.title("Logistic Regression ROC Curve")
plt.show()
```

ROC curve* to evaluate classification performance.

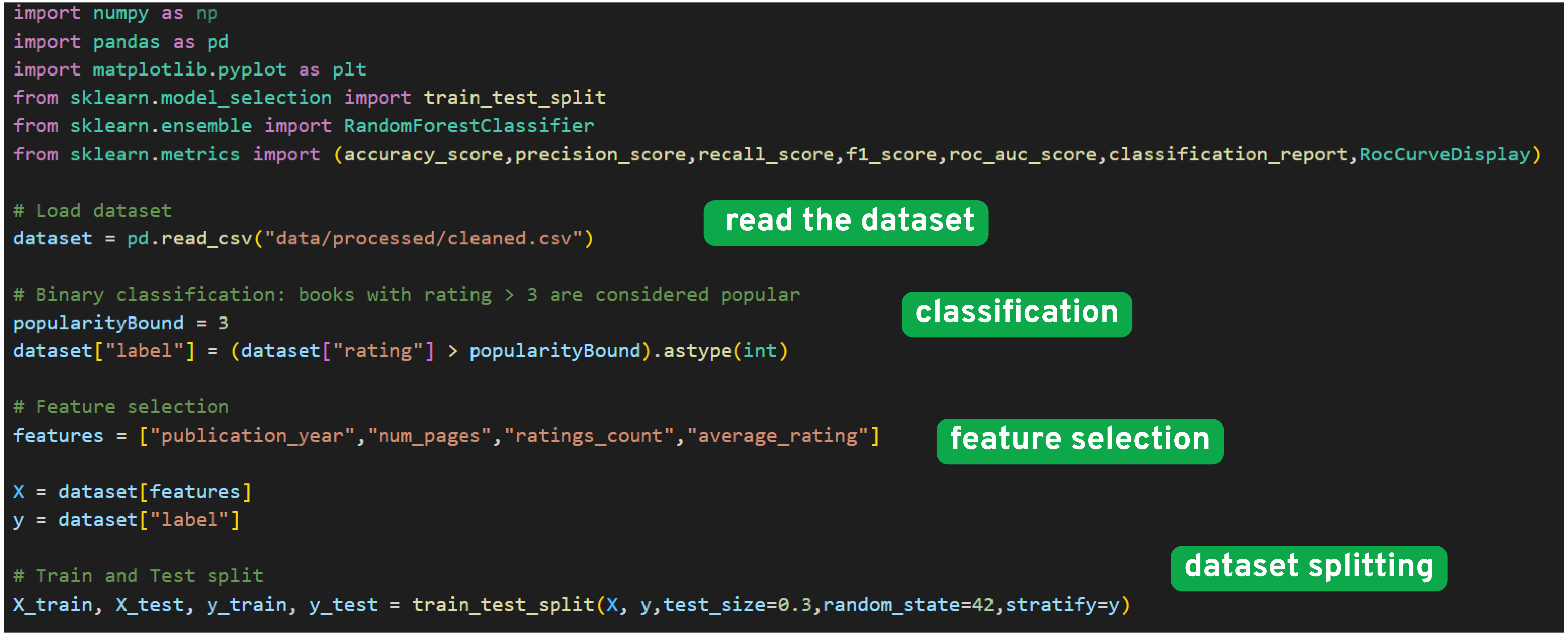
try 1 : Logistic Regression



```
Accuracy : 0.7548
Precision: 0.7548
Recall   : 1.0000
F1-score : 0.8603
ROC-AUC  : 0.5310
```



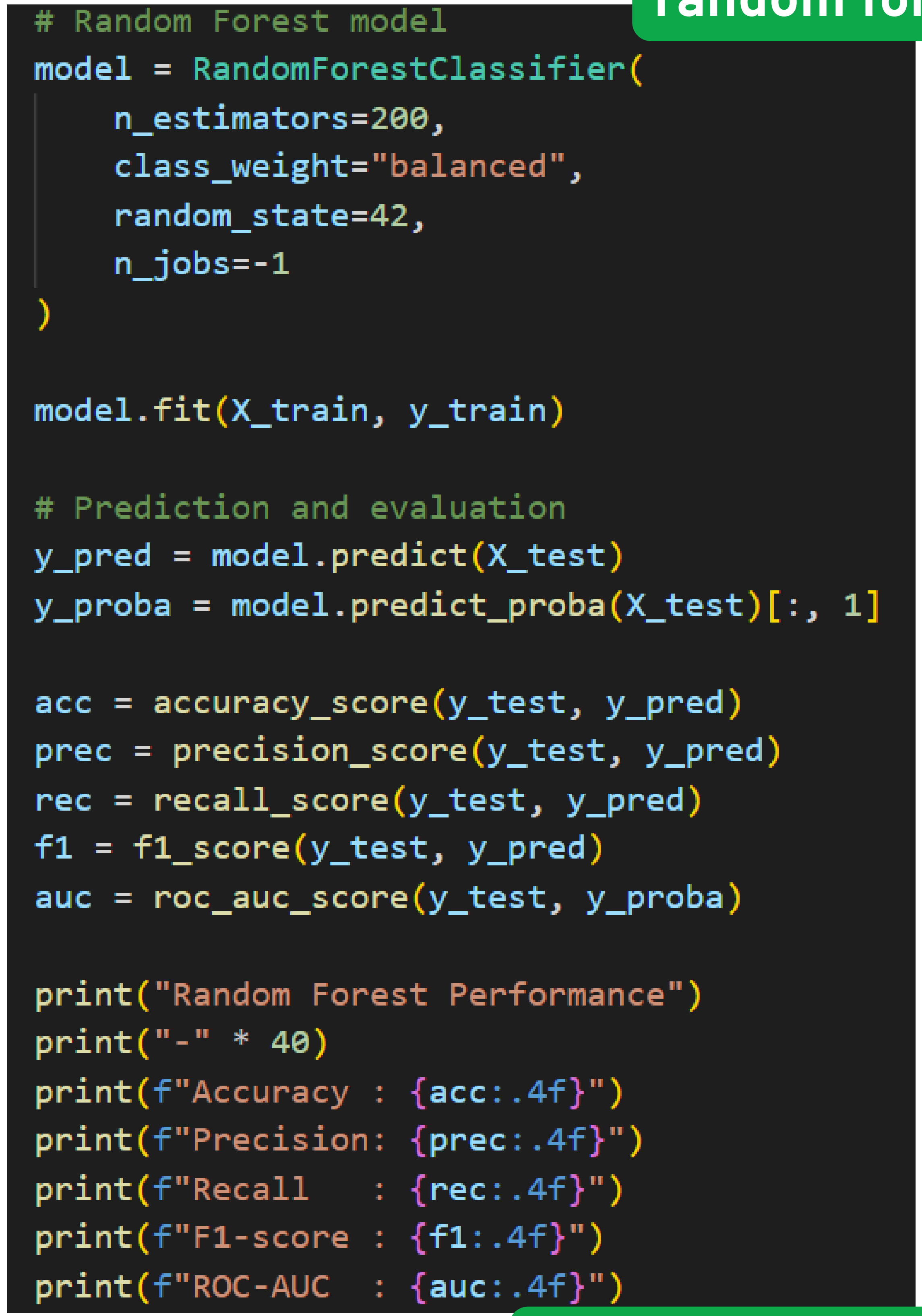
try 2 : Random Forest



```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (accuracy_score,precision_score,recall_score,f1_score,roc_auc_score,classification_report,RocCurveDisplay)

# Load dataset
dataset = pd.read_csv("data/processed/cleaned.csv")

# Binary classification: books with rating > 3 are considered popular
popularityBound = 3
dataset["label"] = (dataset["rating"] > popularityBound).astype(int)

# Feature selection
features = ["publication_year","num_pages","ratings_count","average_rating"]

X = dataset[features]
y = dataset["label"]

# Train and Test split
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.3,random_state=42,stratify=y)
```

try 2 : Random Forest

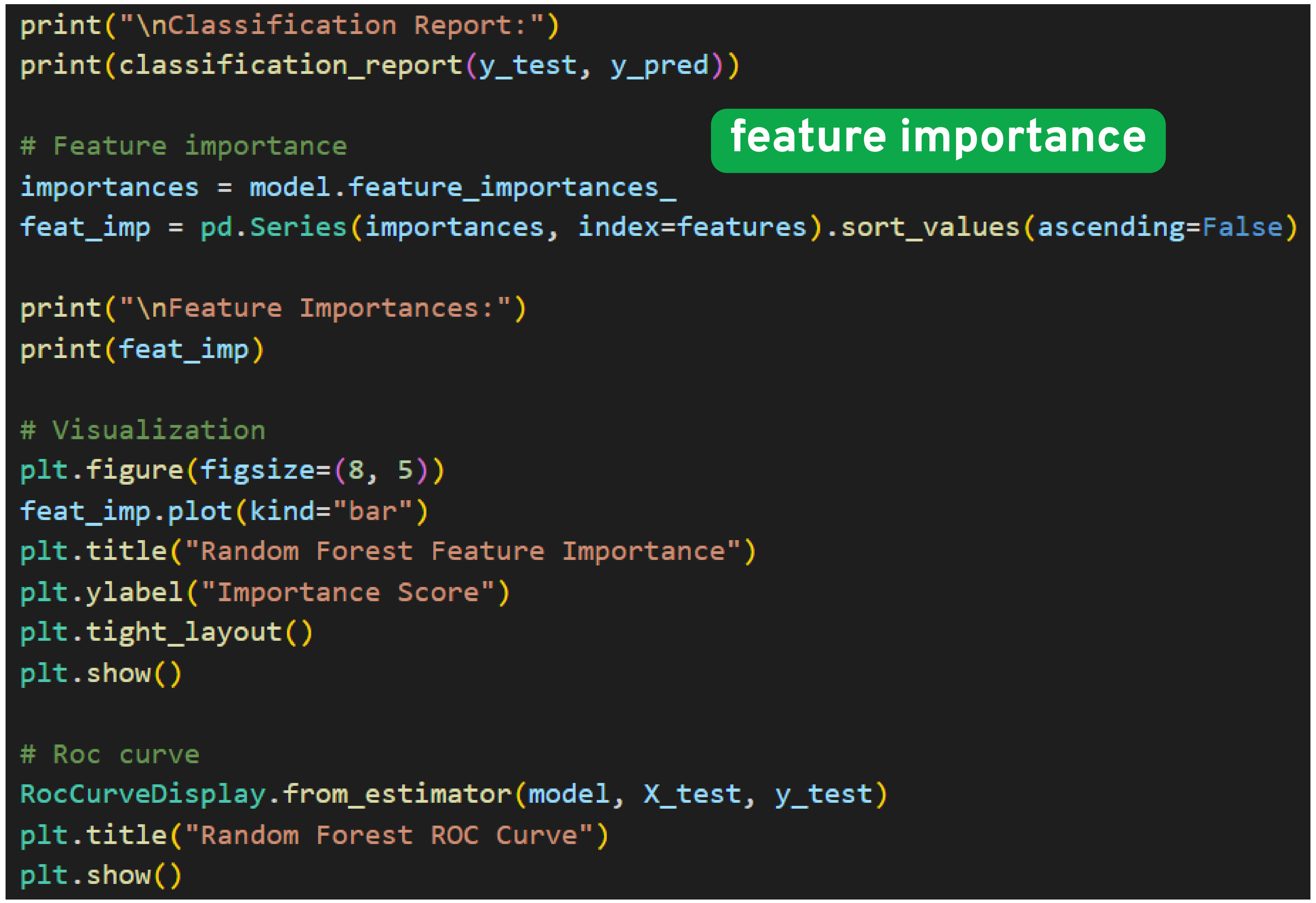
random forest model



```python
# Random Forest model
model = RandomForestClassifier(
    n_estimators=200,
    class_weight="balanced",
    random_state=42,
    n_jobs=-1
)

model.fit(X_train, y_train)

# Prediction and evaluation
y_pred = model.predict(X_test)
y_proba = model.predict_proba(X_test)[:, 1]

acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
auc = roc_auc_score(y_test, y_proba)

print("Random Forest Performance")
print("-" * 40)
print(f"Accuracy : {acc:.4f}")
print(f"Precision: {prec:.4f}")
print(f"Recall   : {rec:.4f}")
print(f"F1-score : {f1:.4f}")
print(f"ROC-AUC  : {auc:.4f}")
```

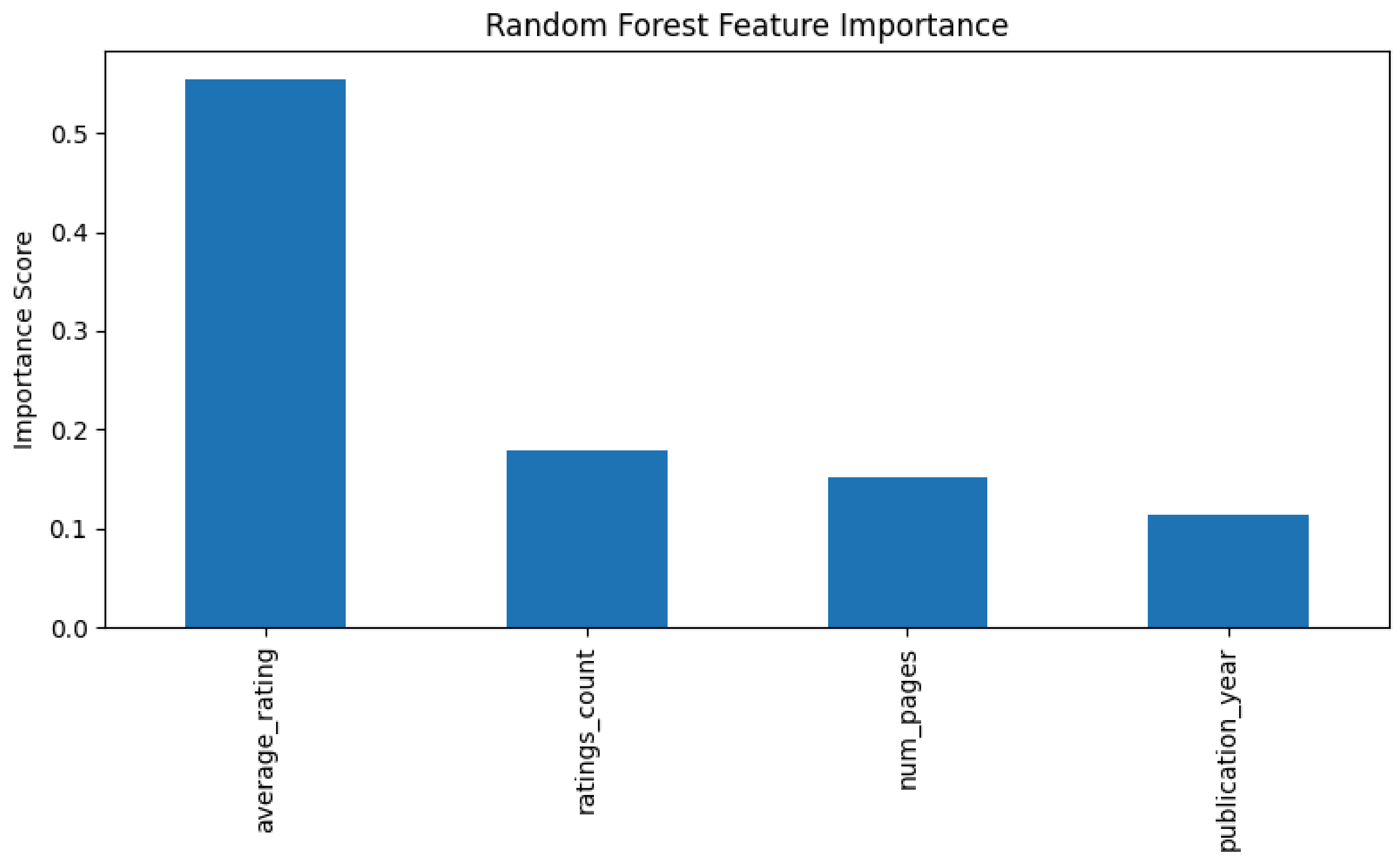
prediction and accuracy

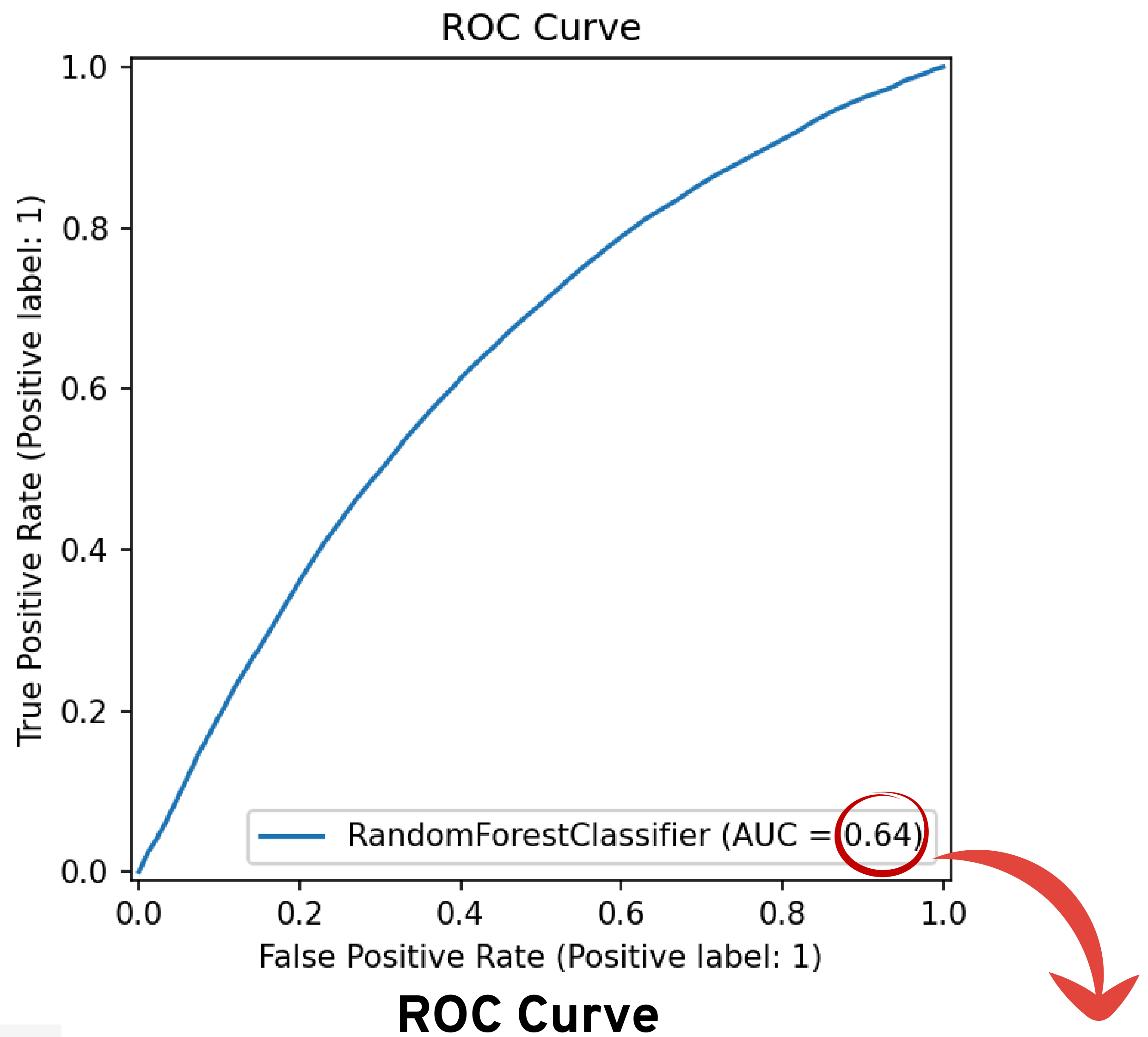
feature importance



```python
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Feature importance
importances = model.feature_importances_
feat_imp = pd.Series(importances, index=features).sort_values(ascending=False)

print("\nFeature Importances:")
print(feat_imp)

# Visualization
plt.figure(figsize=(8, 5))
feat_imp.plot(kind="bar")
plt.title("Random Forest Feature Importance")
plt.ylabel("Importance Score")
plt.tight_layout()
plt.show()

# Roc curve
RocCurveDisplay.from_estimator(model, X_test, y_test)
plt.title("Random Forest ROC Curve")
plt.show()
```

try 2 : Random Forest



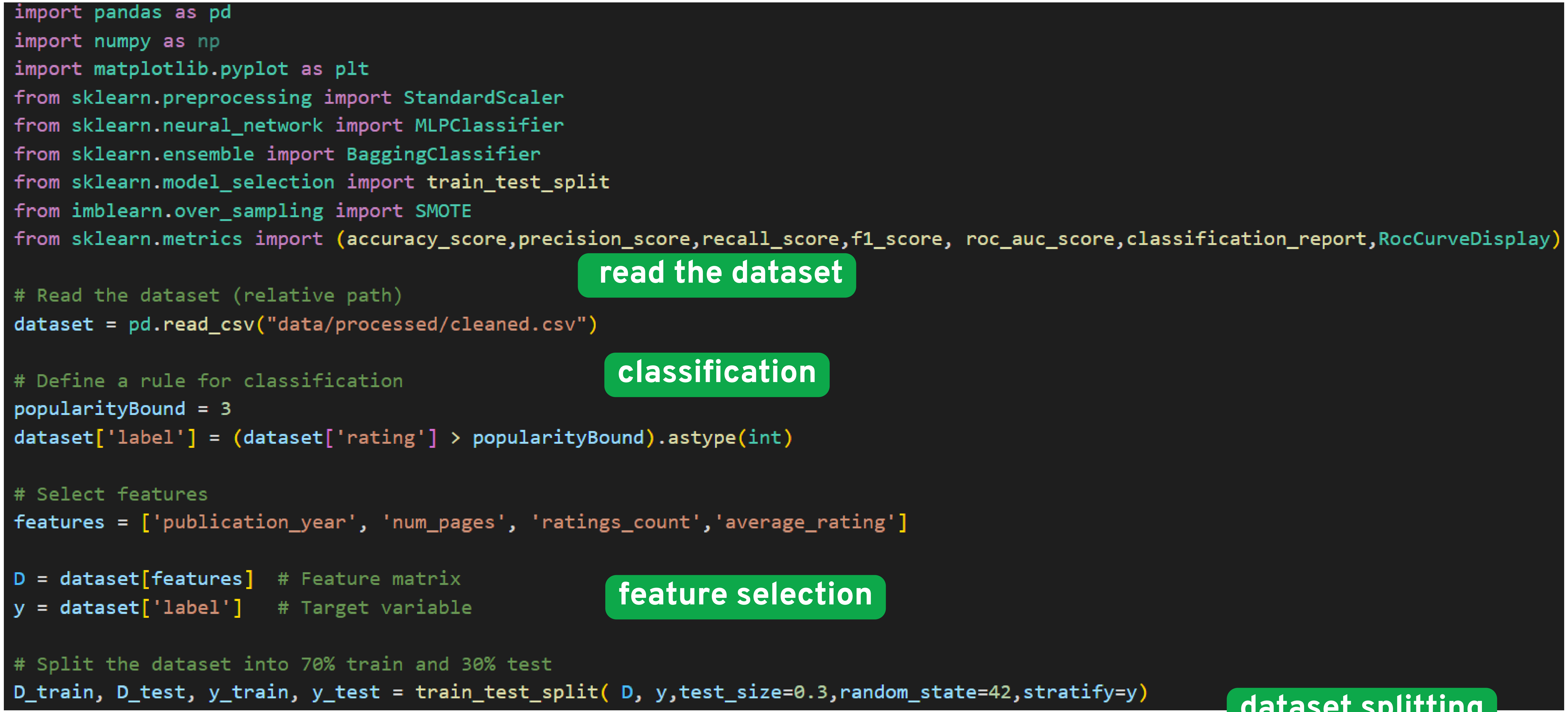


ROC Curve

```
Accuracy : 0.6172
Precision: 0.8211
Recall   : 0.6298
F1-score : 0.7128
ROC-AUC  : 0.6376
```

Random Forest achieved a higher AUC value compared to Logistic Regression.

try 3 : Multilayer Perceptron



```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.metrics import (accuracy_score,precision_score,recall_score,f1_score, roc_auc_score,classification_report,RocCurveDisplay)

# Read the dataset (relative path)
dataset = pd.read_csv("data/processed/cleaned.csv")

# Define a rule for classification
popularityBound = 3
dataset['label'] = (dataset['rating'] > popularityBound).astype(int)

# Select features
features = ['publication_year', 'num_pages', 'ratings_count','average_rating']

D = dataset[features]  # Feature matrix
y = dataset['label']   # Target variable

# Split the dataset into 70% train and 30% test
D_train, D_test, y_train, y_test = train_test_split( D, y,test_size=0.3,random_state=42,stratify=y)
```

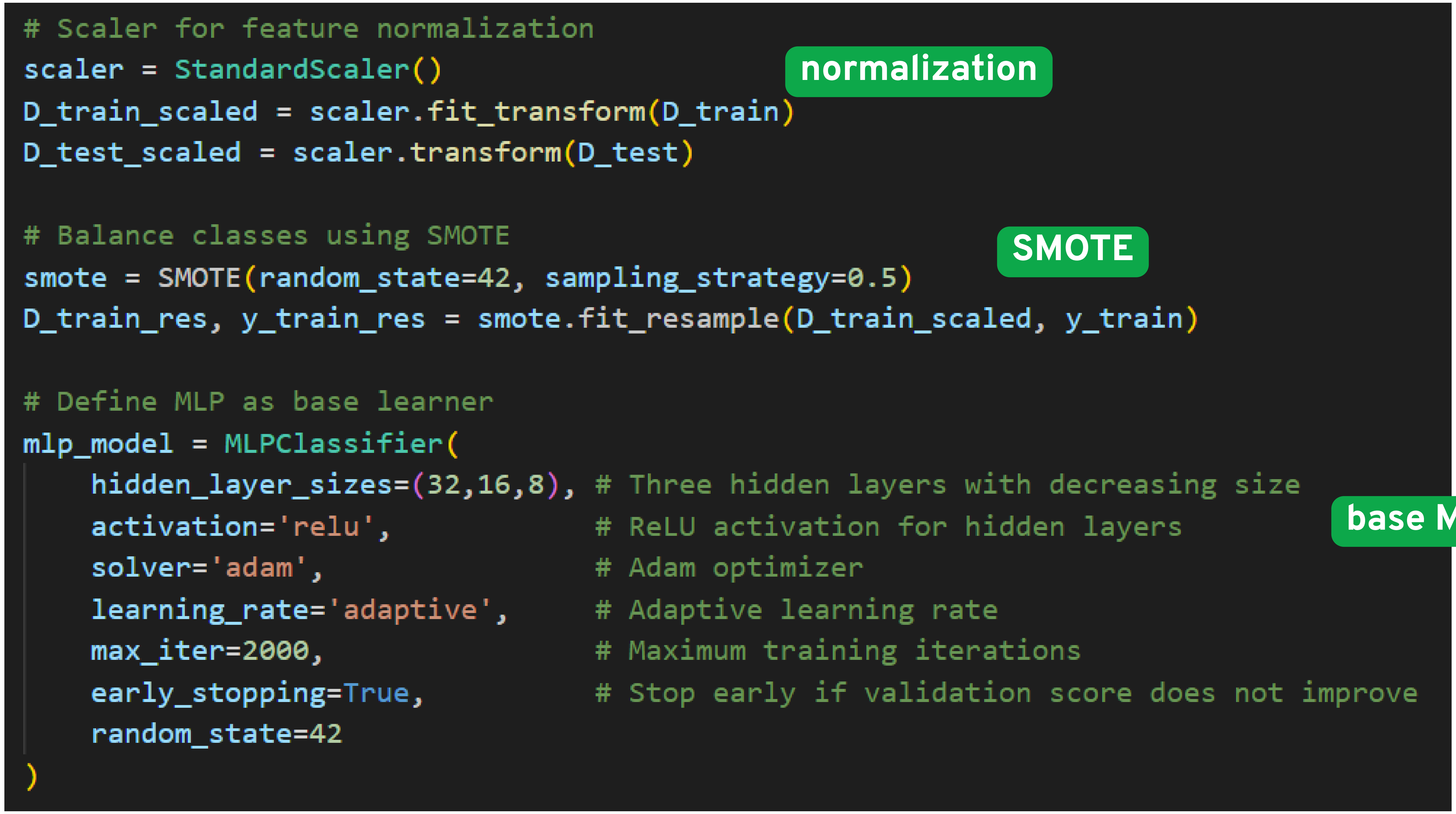
read the dataset

classification

feature selection

dataset splitting

try 3 : Multilayer Perceptron



```
# Scaler for feature normalization
scaler = StandardScaler()
D_train_scaled = scaler.fit_transform(D_train)
D_test_scaled = scaler.transform(D_test)

# Balance classes using SMOTE
smote = SMOTE(random_state=42, sampling_strategy=0.5)
D_train_res, y_train_res = smote.fit_resample(D_train_scaled, y_train)

# Define MLP as base learner
mlp_model = MLPClassifier(
    hidden_layer_sizes=(32,16,8), # Three hidden layers with decreasing size
    activation='relu',            # ReLU activation for hidden layers
    solver='adam',                # Adam optimizer
    learning_rate='adaptive',     # Adaptive learning rate
    max_iter=2000,                # Maximum training iterations
    early_stopping=True,          # Stop early if validation score does not improve
    random_state=42
)
```

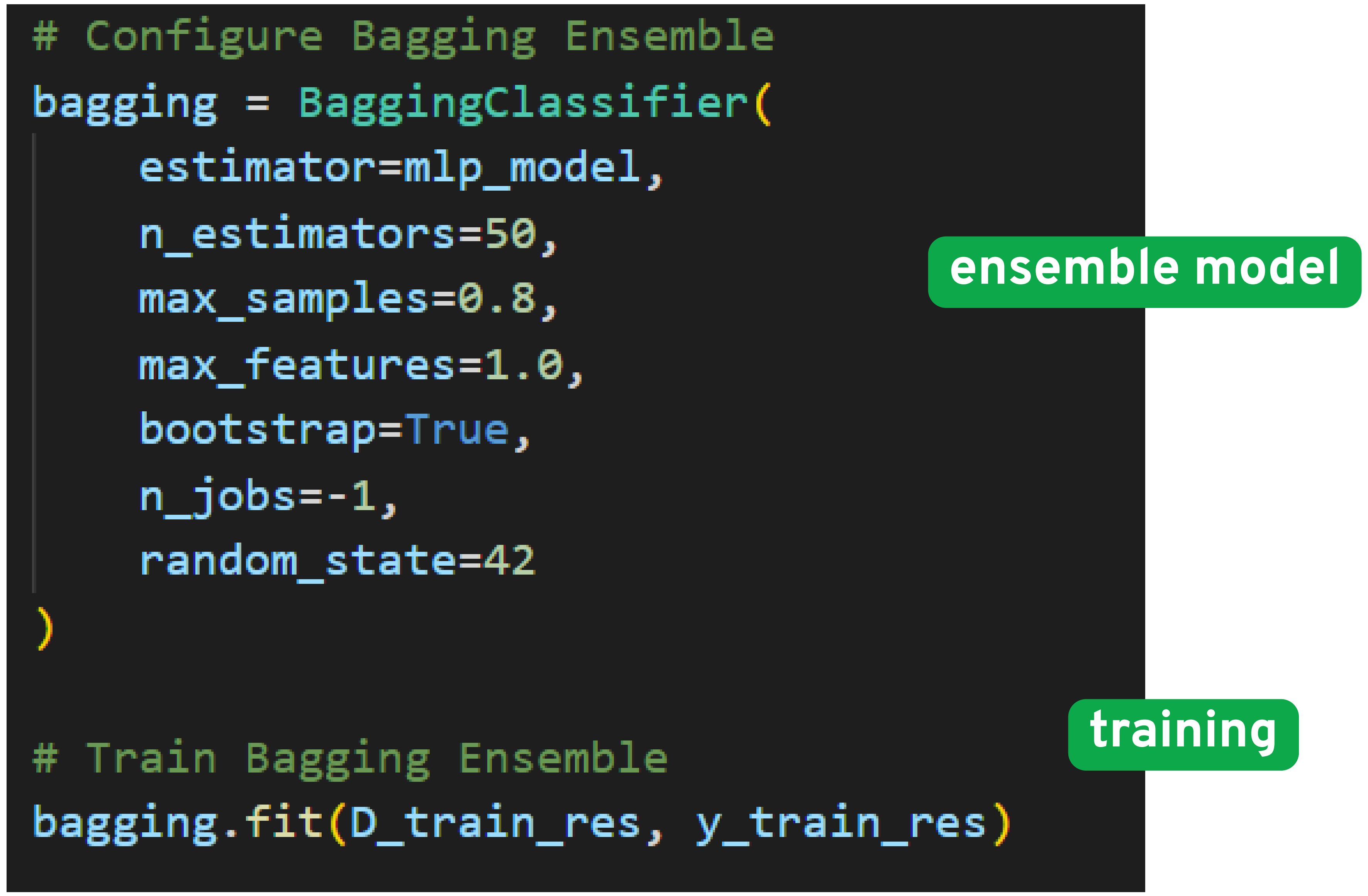
normalization

SMOTE

base MLP model

SMOTE*: To improve the model, it generates synthetic samples for the minority class.
MLPClassifier* is a single neural network model that learns patterns from the data.

try 3 : Multilayer Perceptron

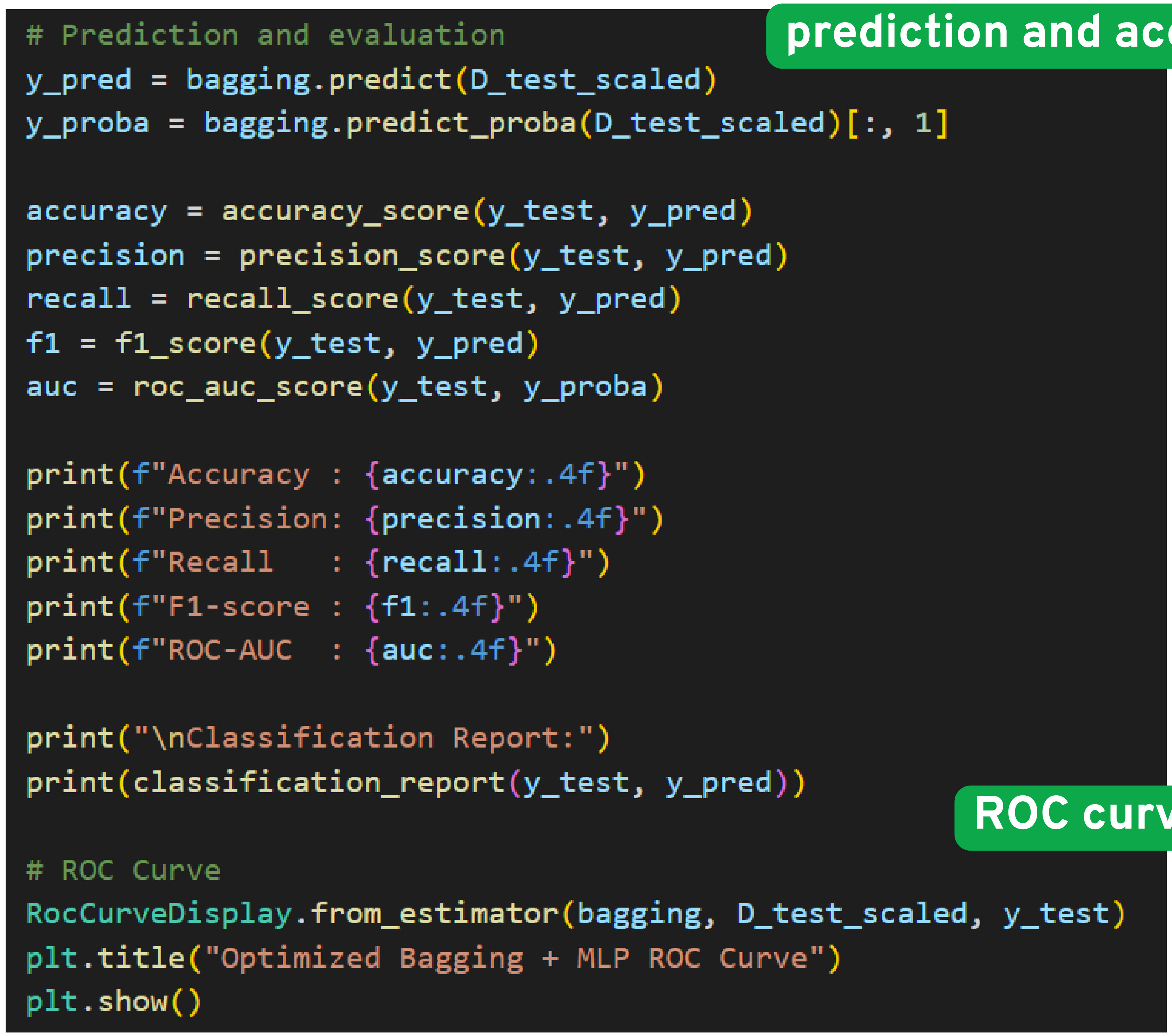


```
# Configure Bagging Ensemble
bagging = BaggingClassifier(
    estimator=mlp_model,
    n_estimators=50,
    max_samples=0.8,
    max_features=1.0,
    bootstrap=True,
    n_jobs=-1,
    random_state=42
)

# Train Bagging Ensemble
bagging.fit(D_train_res, y_train_res)
```

We define **mlp_model** first as the **base learner** for the **bagging ensemble**. **BaggingClassifier** creates multiple copies of mlp_model and trains each on a different subset of the data.

BaggingClassifier is an ensemble method that combines multiple base learners to reduce errors.

try 3 : Multilayer Perceptron

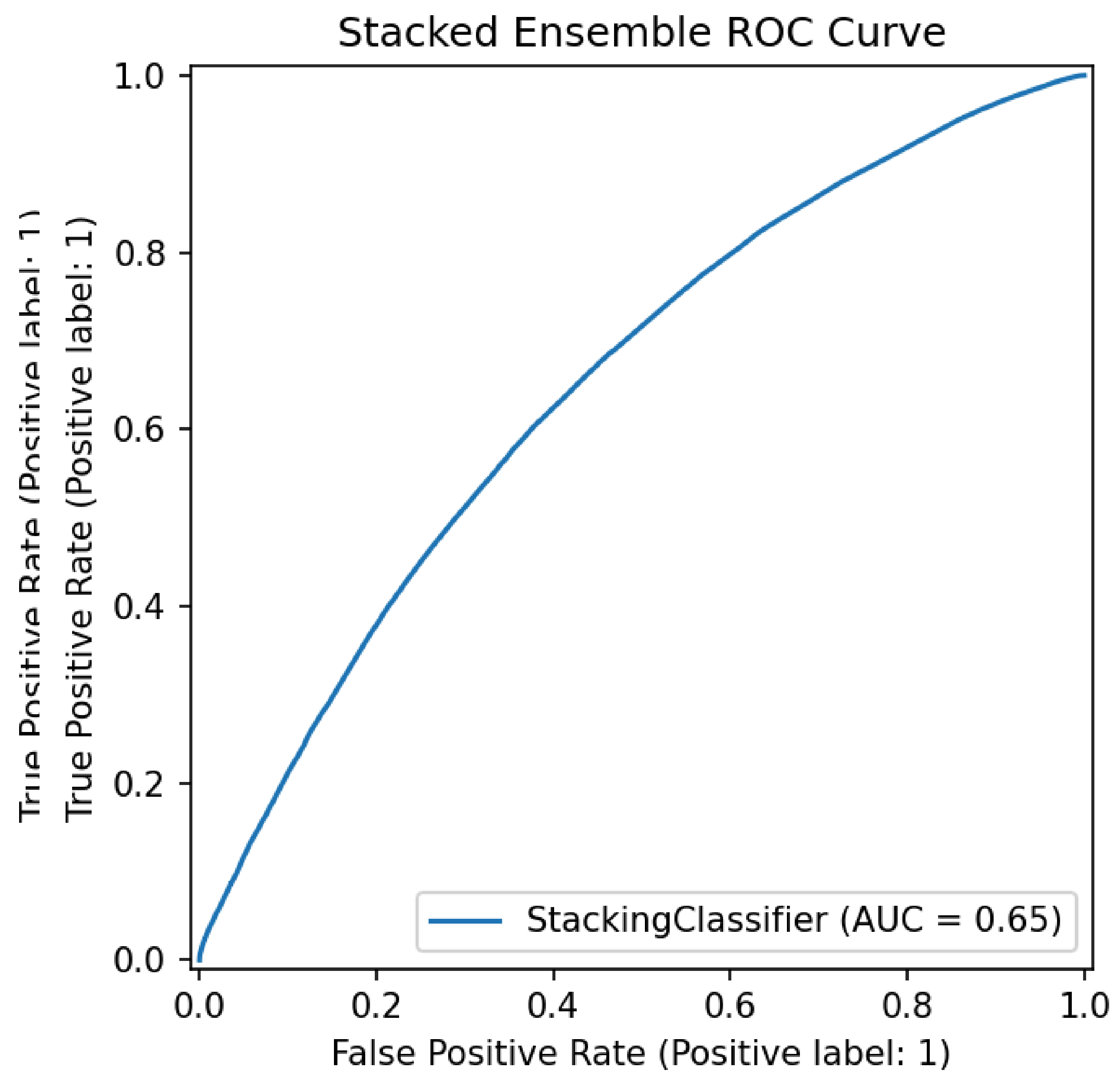


```
# Prediction and evaluation
y_pred = bagging.predict(D_test_scaled)
y_proba = bagging.predict_proba(D_test_scaled)[:, 1]

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
auc = roc_auc_score(y_test, y_proba)

print(f"Accuracy : {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall   : {recall:.4f}")
print(f"F1-score : {f1:.4f}")
print(f"ROC-AUC  : {auc:.4f}")

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# ROC Curve
RocCurveDisplay.from_estimator(bagging, D_test_scaled, y_test)
plt.title("Optimized Bagging + MLP ROC Curve")
plt.show()
```

prediction and accuracy

ROC curve

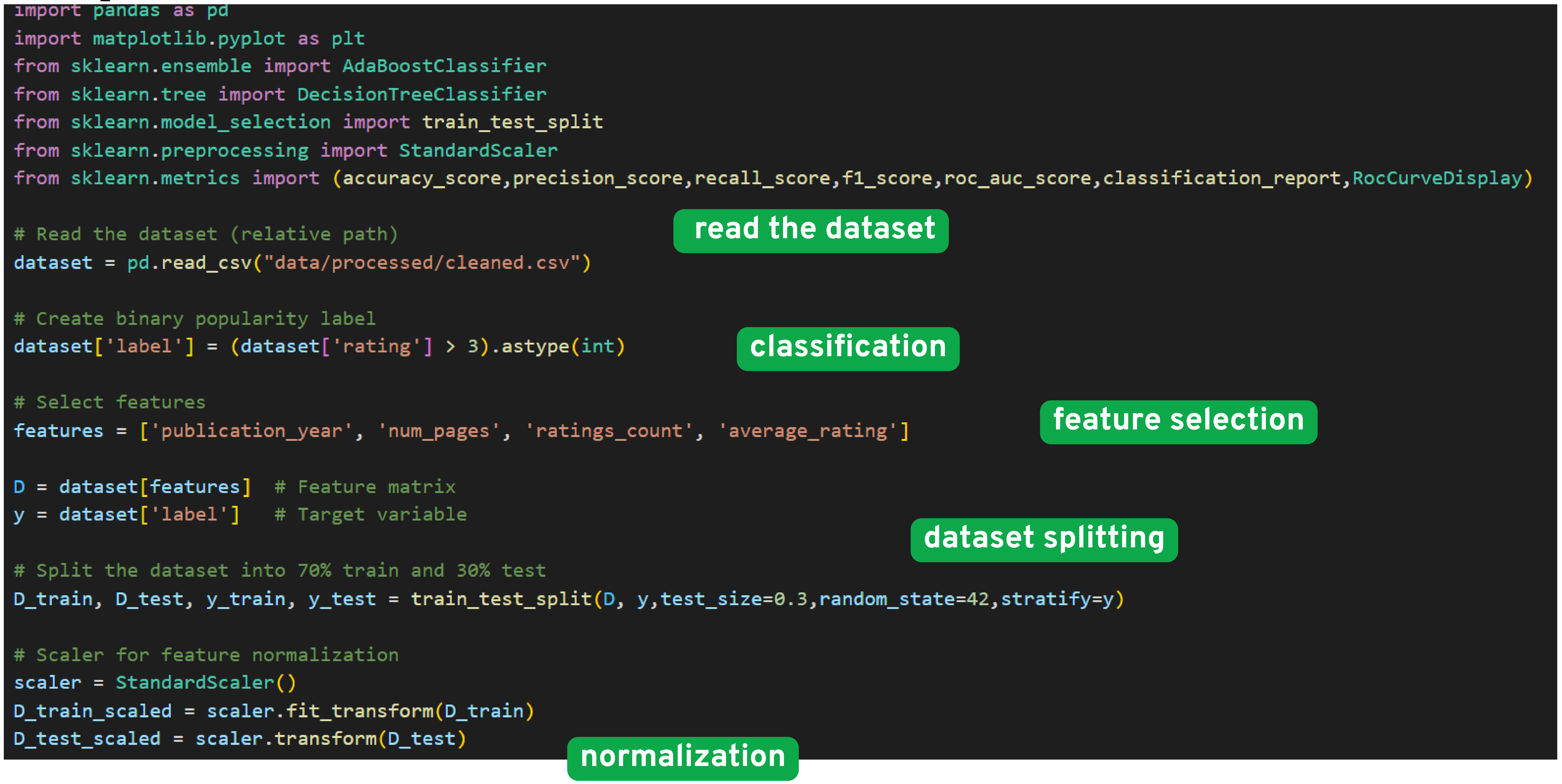
try 3 : Multilayer Perceptron



```
Accuracy : 0.7506
Precision: 0.7741
Recall   : 0.9453
F1-score : 0.8512
ROC-AUC  : 0.6537
```

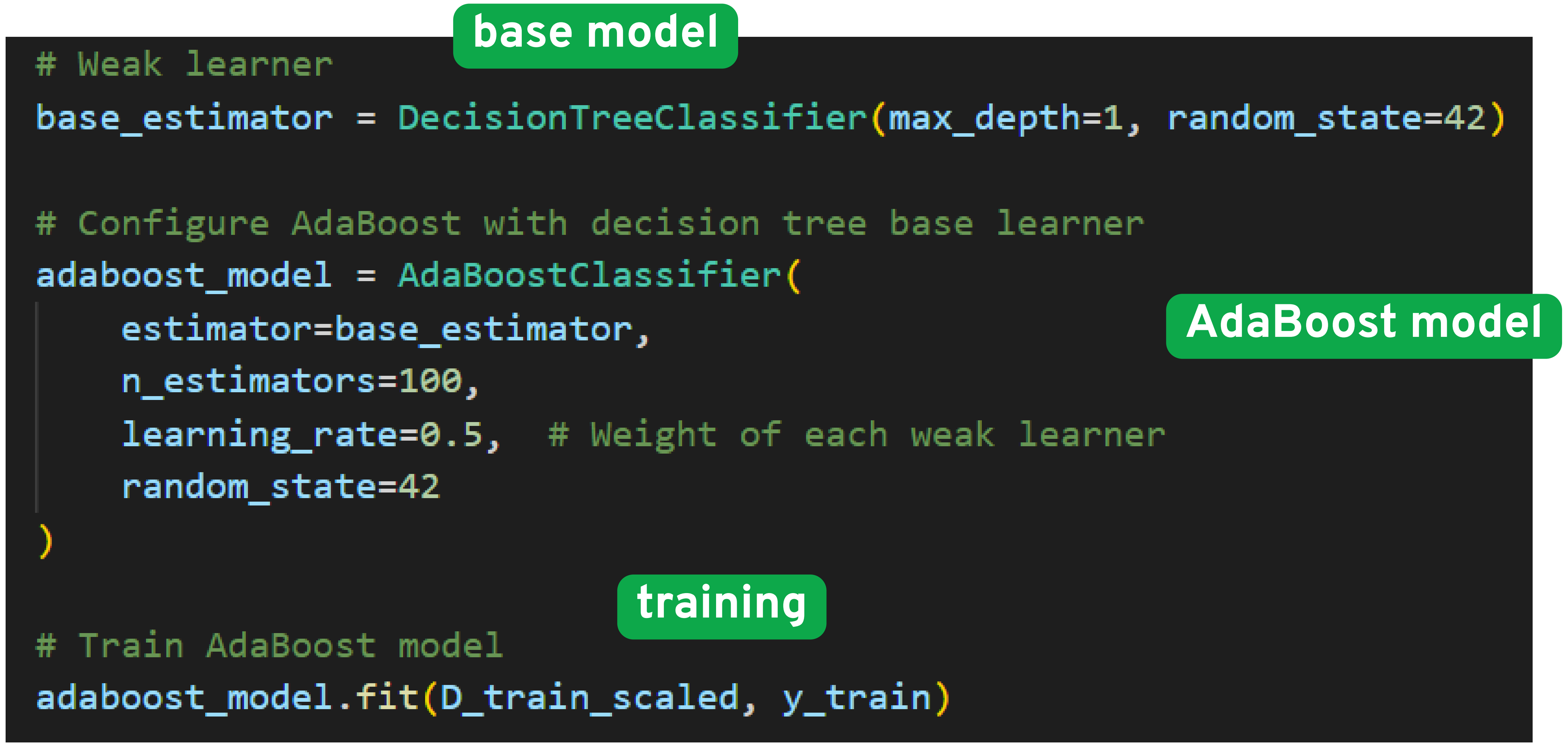
Our dataset has too few numeric features, so the MLP does not have enough information, making the model weak.

try 4 : Adaboost



```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (accuracy_score,precision_score,recall_score,f1_score,roc_auc_score,classification_report,RocCurveDisplay)

# Read the dataset (relative path)
dataset = pd.read_csv("data/processed/cleaned.csv")

# Create binary popularity label
dataset['label'] = (dataset['rating'] > 3).astype(int)

# Select features
features = ['publication_year', 'num_pages', 'ratings_count', 'average_rating']

D = dataset[features]  # Feature matrix
y = dataset['label']   # Target variable

# Split the dataset into 70% train and 30% test
D_train, D_test, y_train, y_test = train_test_split(D, y,test_size=0.3,random_state=42,stratify=y)

# Scaler for feature normalization
scaler = StandardScaler()
D_train_scaled = scaler.fit_transform(D_train)
D_test_scaled = scaler.transform(D_test)
```

try 4 : Adaboost

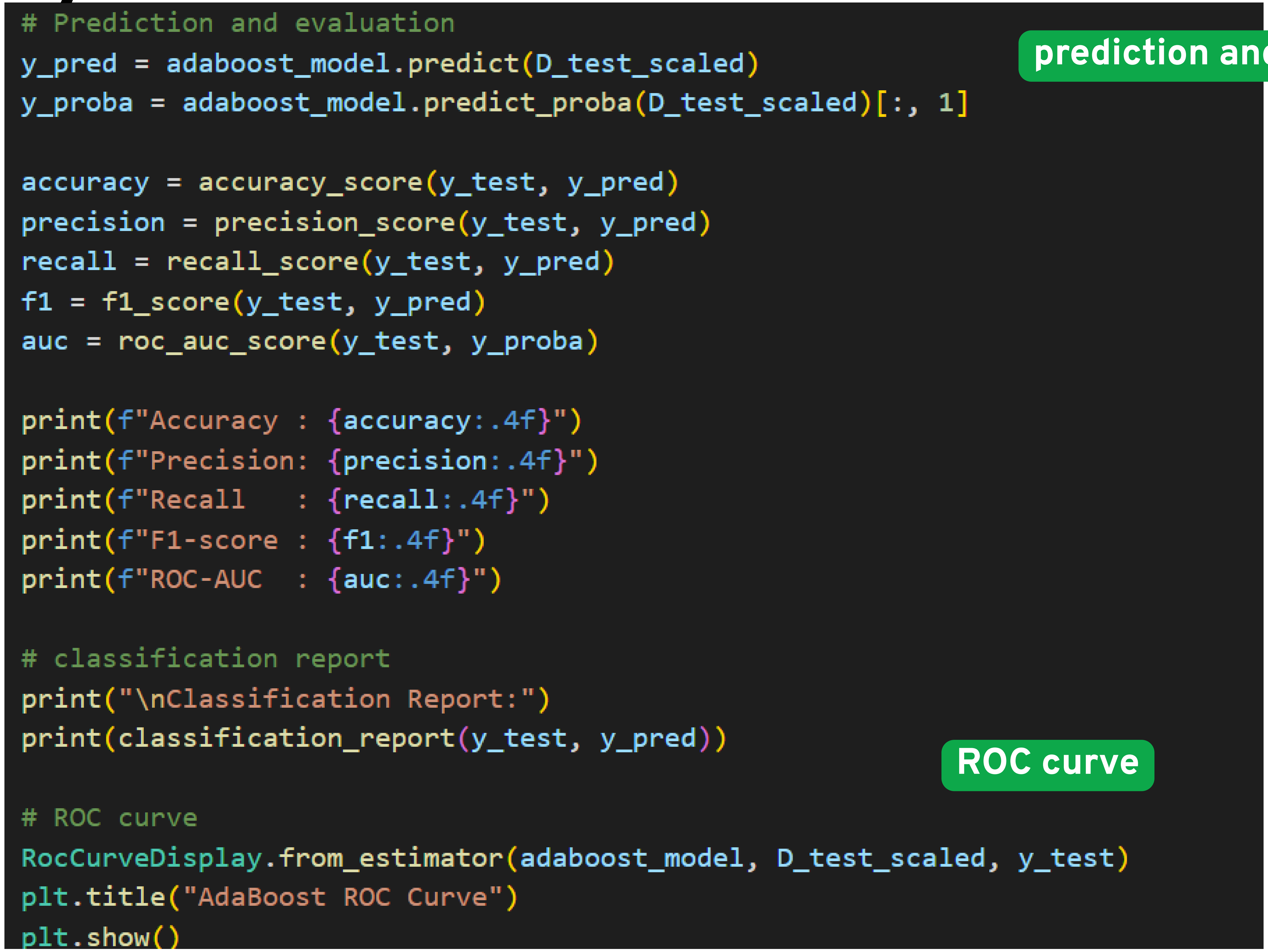


```
# Weak learner
base_estimator = DecisionTreeClassifier(max_depth=1, random_state=42)

# Configure AdaBoost with decision tree base learner
adaboost_model = AdaBoostClassifier(
    estimator=base_estimator,
    n_estimators=100,
    learning_rate=0.5,  # Weight of each weak learner
    random_state=42
)

# Train AdaBoost model
adaboost_model.fit(D_train_scaled, y_train)
```

Bagging trains **many independent models in parallel** using bootstrap samples, while **AdaBoost** trains models **sequentially** and increases the weight of **misclassified** samples to focus on hard cases.

base_estimator* : A shallow decision stump (single-split tree) used as a **fast weak learner for AdaBoost.**
AdaBoost* **trains n weak learners and combines them**, giving more weight to hard examples.

try 4 : Adaboost

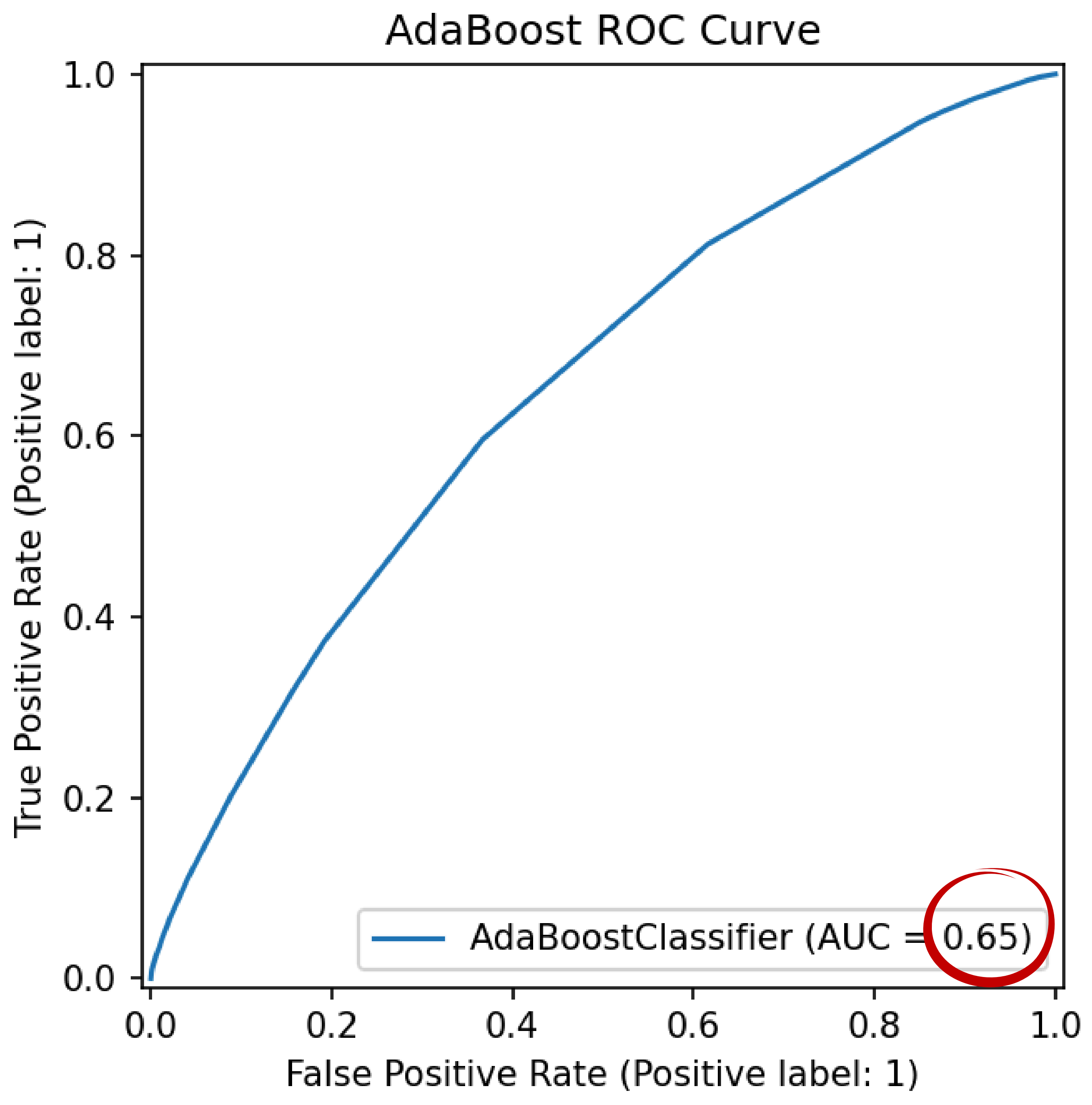


```
# Prediction and evaluation
y_pred = adaboost_model.predict(D_test_scaled)
y_proba = adaboost_model.predict_proba(D_test_scaled)[:, 1]

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
auc = roc_auc_score(y_test, y_proba)

print(f"Accuracy : {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall   : {recall:.4f}")
print(f"F1-score : {f1:.4f}")
print(f"ROC-AUC  : {auc:.4f}")

# classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# ROC curve
RocCurveDisplay.from_estimator(adaboost_model, D_test_scaled, y_test)
plt.title("AdaBoost ROC Curve")
plt.show()
```

prediction and accuracy

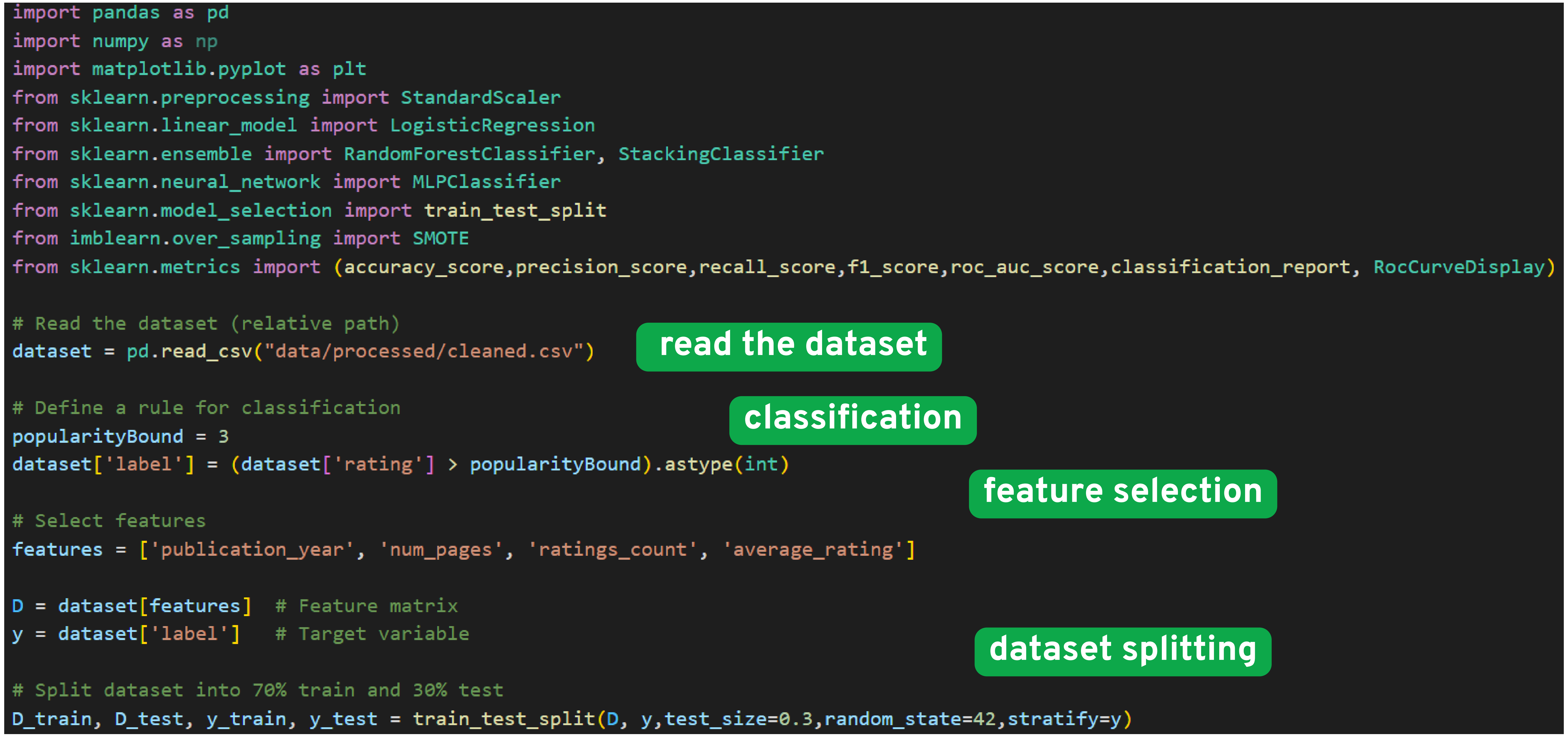
ROC curve

try 4 : Adaboost



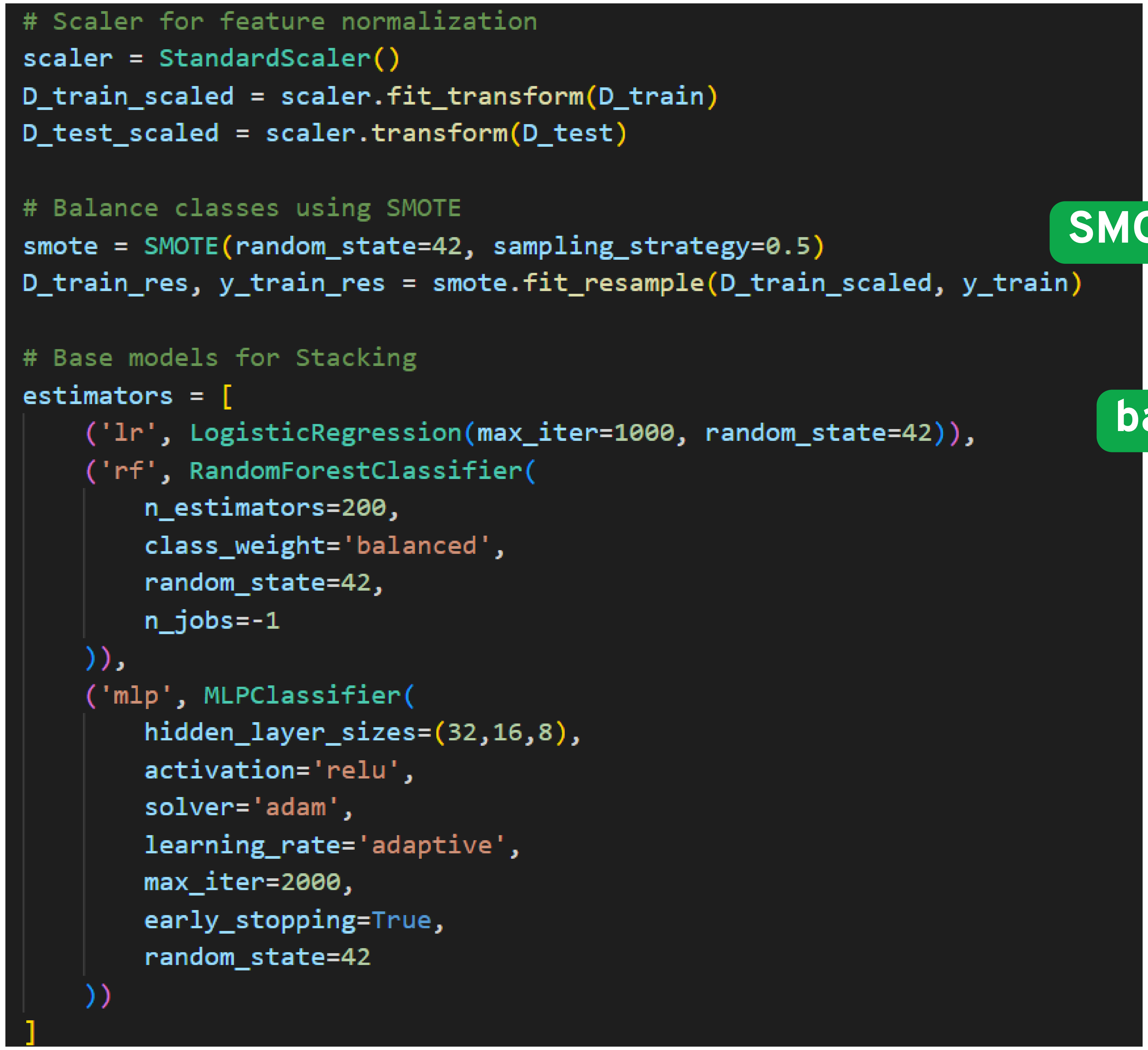
```
Accuracy : 0.7560
Precision: 0.7564
Recall   : 0.9979
F1-score : 0.8605
ROC-AUC  : 0.6496
```

try 5 : Stacking Ensemble Using Logistic Regression, Random Forest, and MLP



```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, StackingClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.metrics import (accuracy_score,precision_score,recall_score,f1_score,roc_auc_score,classification_report, RocCurveDisplay)

# Read the dataset (relative path)
dataset = pd.read_csv("data/processed/cleaned.csv")

# Define a rule for classification
popularityBound = 3
dataset['label'] = (dataset['rating'] > popularityBound).astype(int)

# Select features
features = ['publication_year', 'num_pages', 'ratings_count', 'average_rating']

D = dataset[features]  # Feature matrix
y = dataset['label']   # Target variable

# Split dataset into 70% train and 30% test
D_train, D_test, y_train, y_test = train_test_split(D, y,test_size=0.3,random_state=42,stratify=y)
```

try 5 : Stacking Ensemble Using Logistic Regression, Random Forest, and MLP

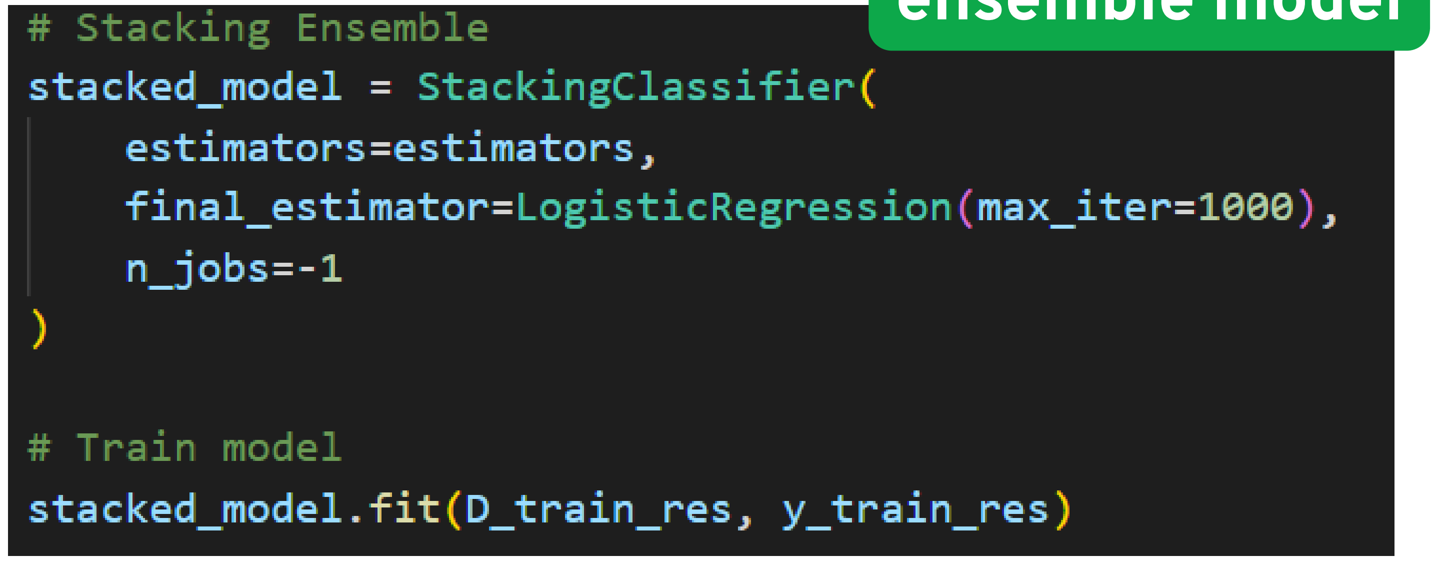


```python
# Scaler for feature normalization
scaler = StandardScaler()
D_train_scaled = scaler.fit_transform(D_train)
D_test_scaled = scaler.transform(D_test)

# Balance classes using SMOTE
smote = SMOTE(random_state=42, sampling_strategy=0.5)
D_train_res, y_train_res = smote.fit_resample(D_train_scaled, y_train)

# Base models for Stacking
estimators = [
    ('lr', LogisticRegression(max_iter=1000, random_state=42)),
    ('rf', RandomForestClassifier(
        n_estimators=200,
        class_weight='balanced',
        random_state=42,
        n_jobs=-1
    )),
    ('mlp', MLPClassifier(
        hidden_layer_sizes=(32,16,8),
        activation='relu',
        solver='adam',
        learning_rate='adaptive',
        max_iter=2000,
        early_stopping=True,
        random_state=42
    ))
]
```

SMOTE

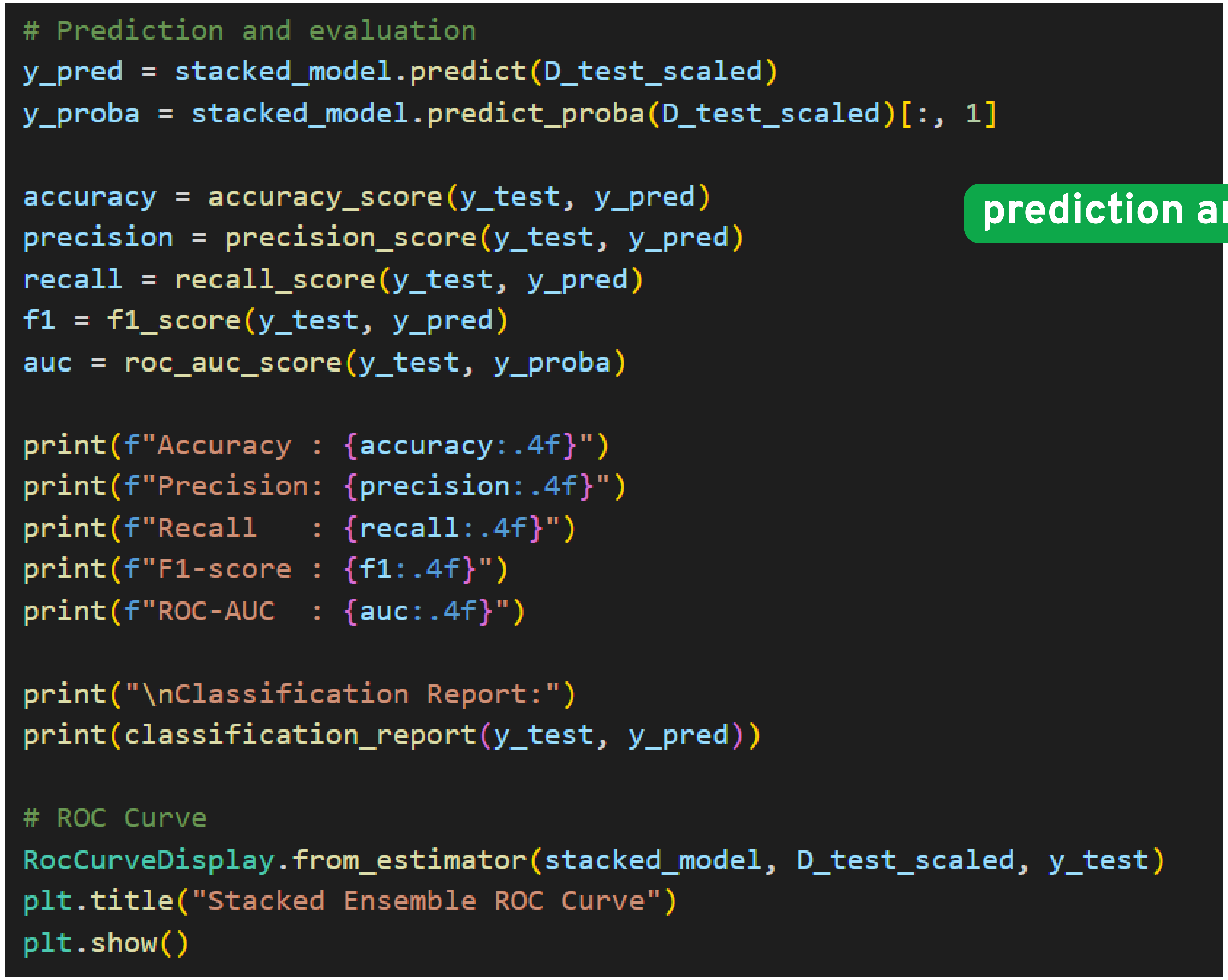
base models

ensemble model



```python
# Stacking Ensemble
stacked_model = StackingClassifier(
    estimators=estimators,
    final_estimator=LogisticRegression(max_iter=1000),
    n_jobs=-1
)

# Train model
stacked_model.fit(D_train_res, y_train_res)
```

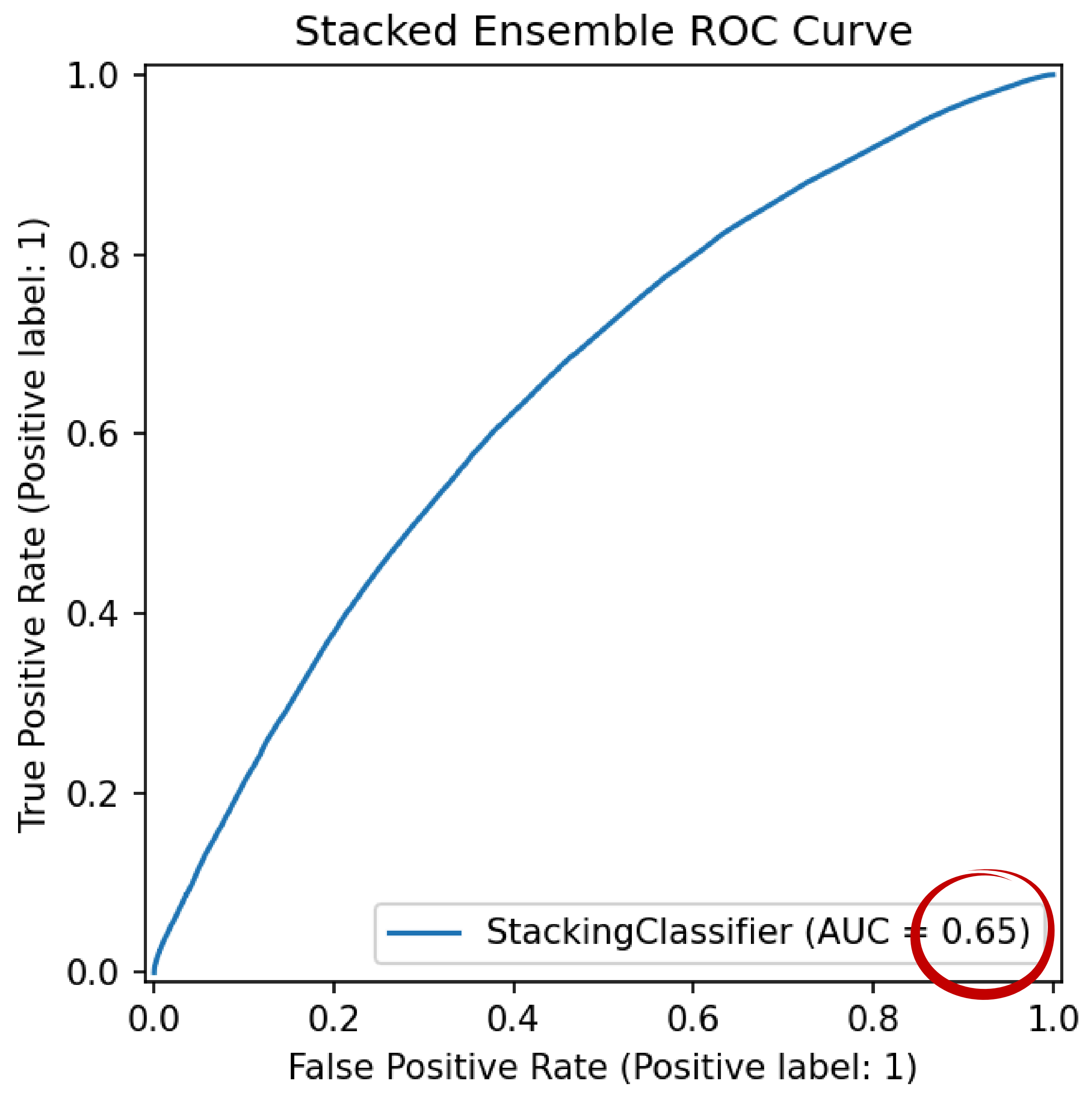
try 5 : Stacking Ensemble Using Logistic Regression, Random Forest, and MLP



```
# Prediction and evaluation
y_pred = stacked_model.predict(D_test_scaled)
y_proba = stacked_model.predict_proba(D_test_scaled)[:, 1]

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
auc = roc_auc_score(y_test, y_proba)

print(f"Accuracy : {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall   : {recall:.4f}")
print(f"F1-score : {f1:.4f}")
print(f"ROC-AUC  : {auc:.4f}")

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# ROC Curve
RocCurveDisplay.from_estimator(stacked_model, D_test_scaled, y_test)
plt.title("Stacked Ensemble ROC Curve")
plt.show()
```

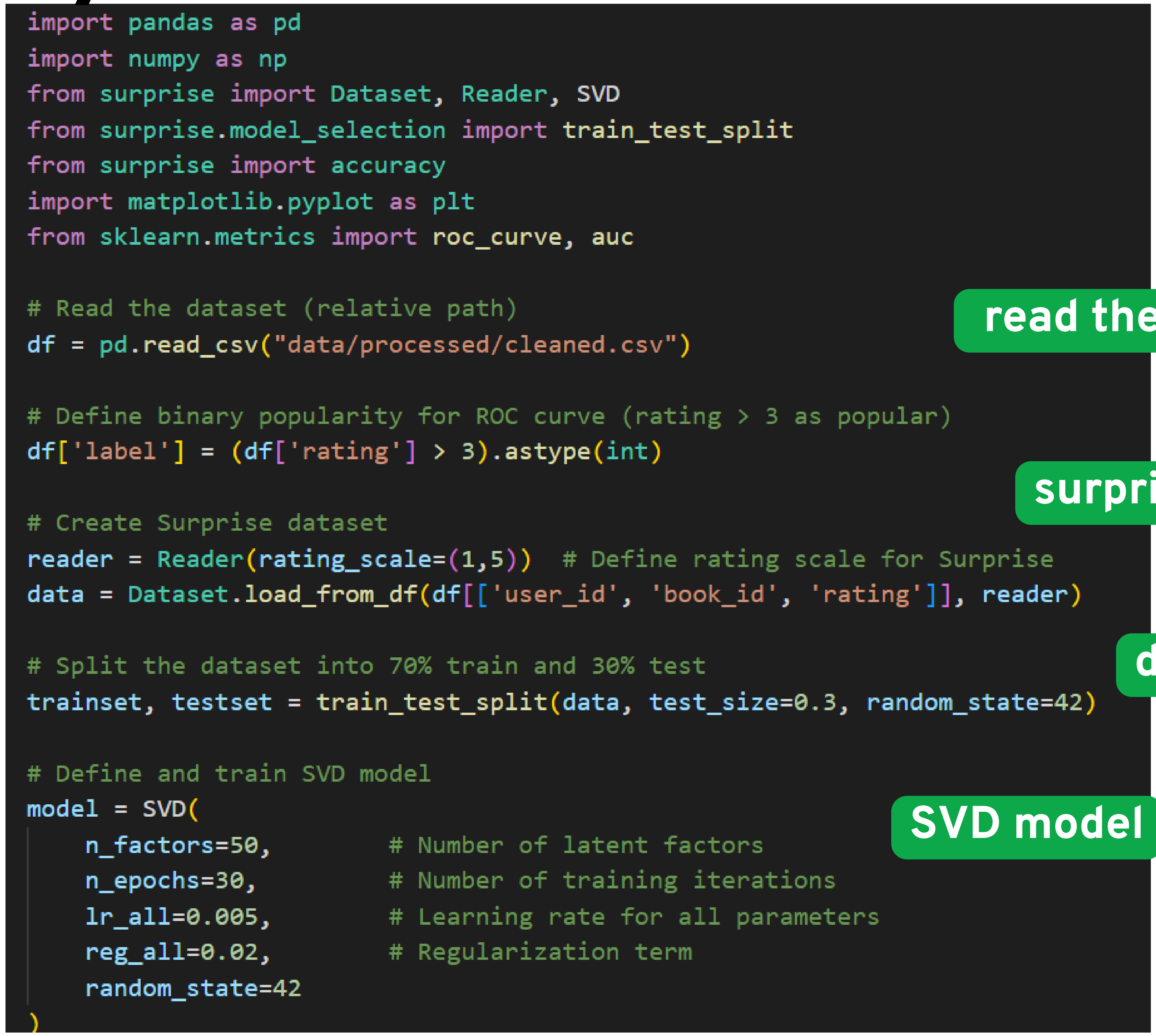
prediction and accuracy

try 5 : Stacking Ensemble Using Logistic Regression, Random Forest, and MLP



```
Accuracy : 0.7341
Precision: 0.7840
Recall   : 0.8937
F1-score : 0.8353
ROC-AUC  : 0.6493
```

try 6 : SVD



```python
import pandas as pd
import numpy as np
from surprise import Dataset, Reader, SVD
from surprise.model_selection import train_test_split
from surprise import accuracy
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Read the dataset (relative path)
df = pd.read_csv("data/processed/cleaned.csv")

# Define binary popularity for ROC curve (rating > 3 as popular)
df['label'] = (df['rating'] > 3).astype(int)

# Create Surprise dataset
reader = Reader(rating_scale=(1,5))  # Define rating scale for Surprise
data = Dataset.load_from_df(df[['user_id', 'book_id', 'rating']], reader)

# Split the dataset into 70% train and 30% test
trainset, testset = train_test_split(data, test_size=0.3, random_state=42)

# Define and train SVD model
model = SVD(
    n_factors=50,         # Number of latent factors
    n_epochs=30,          # Number of training iterations
    lr_all=0.005,         # Learning rate for all parameters
    reg_all=0.02,         # Regularization term
    random_state=42
)
```

read the dataset

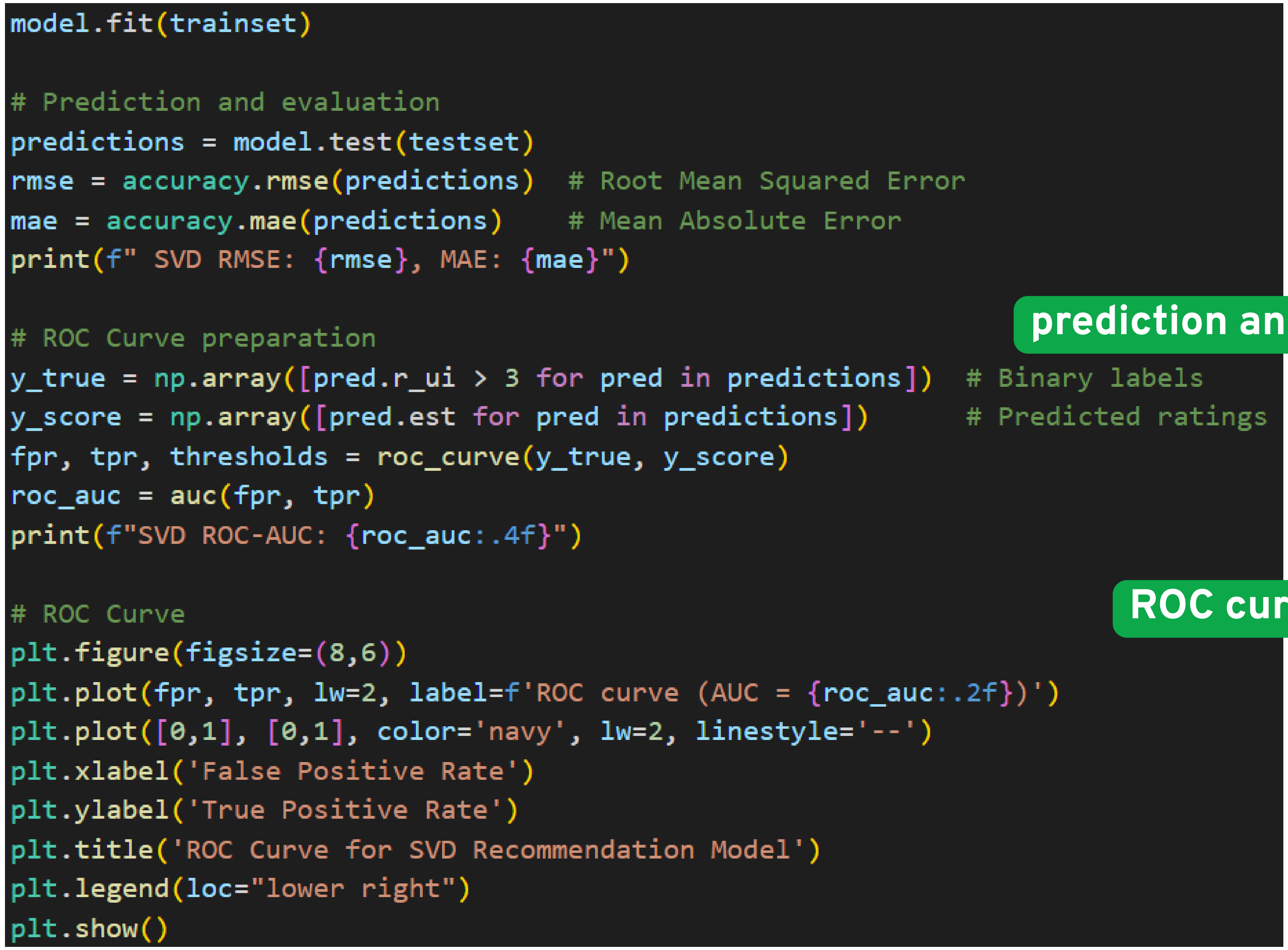
surprise dataset

dataset splitting

SVD model

The SVD model* is a **recommendation system algorithm** that learns the hidden (latent) features of users and items to predict missing ratings.

try 6 : SVD

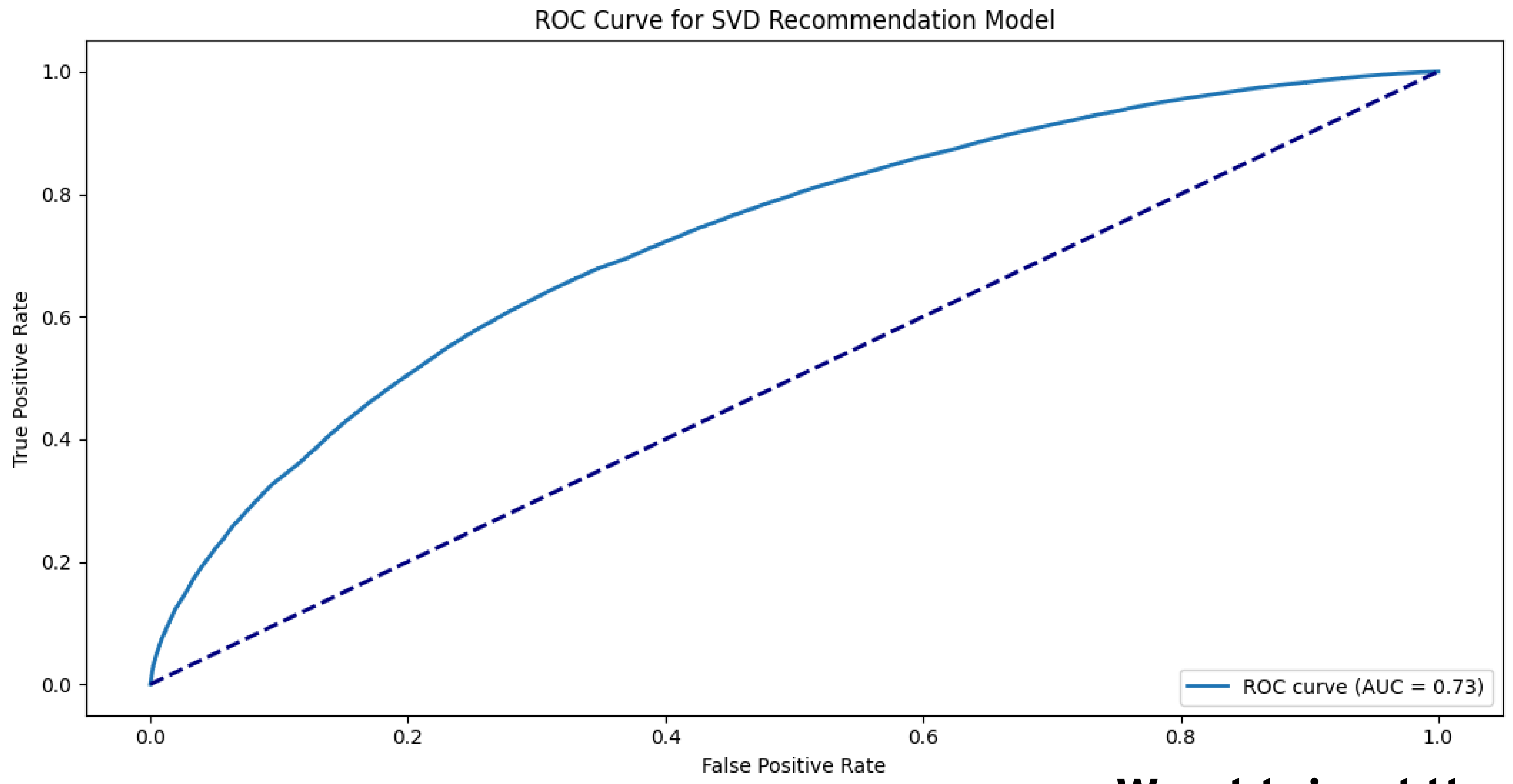


```
model.fit(trainset)

# Prediction and evaluation
predictions = model.test(testset)
rmse = accuracy.rmse(predictions)  # Root Mean Squared Error
mae = accuracy.mae(predictions)    # Mean Absolute Error
print(f" SVD RMSE: {rmse}, MAE: {mae}")

# ROC Curve preparation
y_true = np.array([pred.r_ui > 3 for pred in predictions])  # Binary labels
y_score = np.array([pred.est for pred in predictions])      # Predicted ratings
fpr, tpr, thresholds = roc_curve(y_true, y_score)
roc_auc = auc(fpr, tpr)
print(f"SVD ROC-AUC: {roc_auc:.4f}")

# ROC Curve
plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0,1], [0,1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for SVD Recommendation Model')
plt.legend(loc="lower right")
plt.show()
```

prediction and accuracy

ROC curve

try 6 : SVD

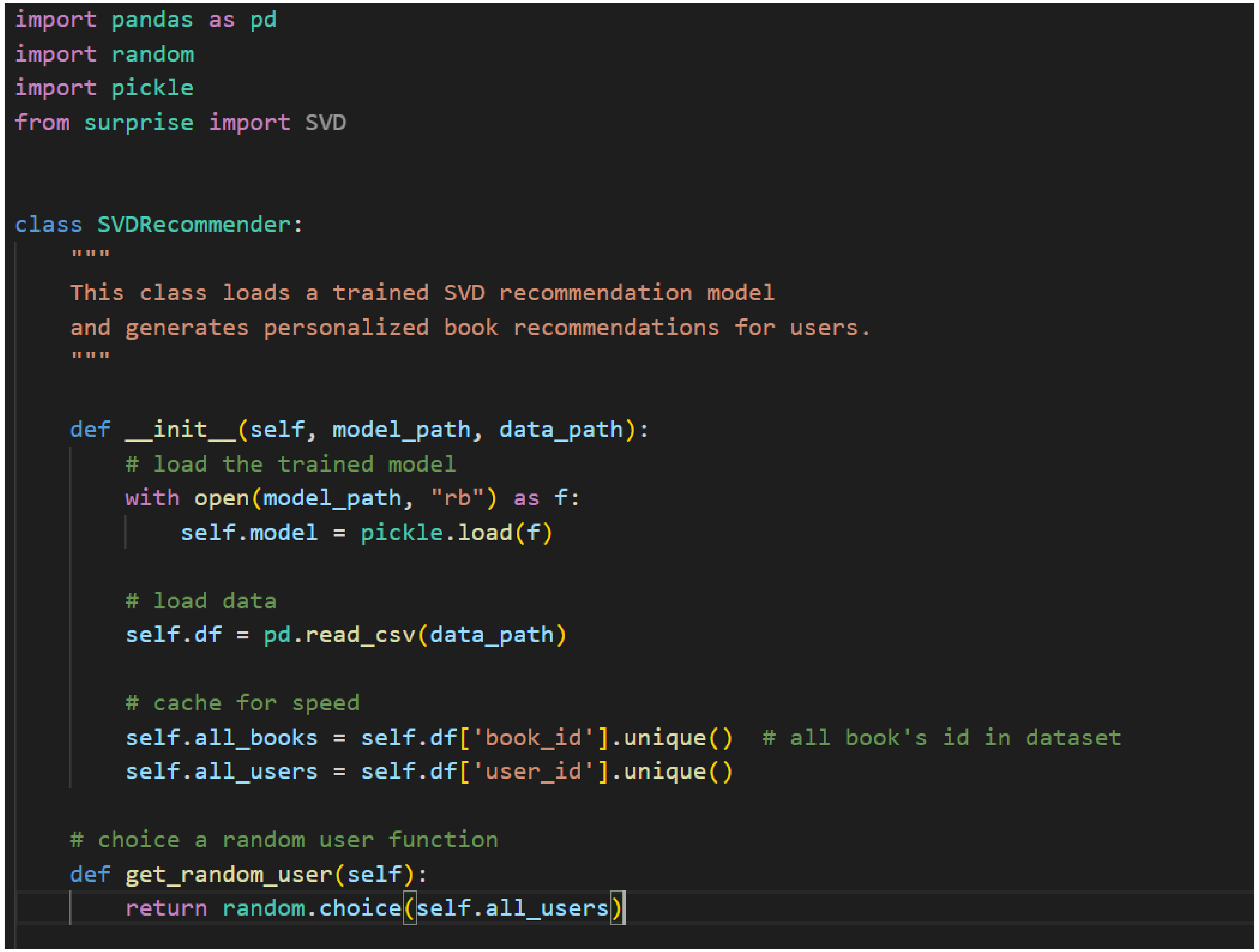


```
RMSE: 0.8657
MAE:  0.6842
 SVD RMSE: 0.8656942182728519, MAE: 0.6842251631518097
SVD ROC-AUC: 0.7262
```

We obtained the best AUC result with the SVD model, so we will proceed using SVD for the recommendation system.

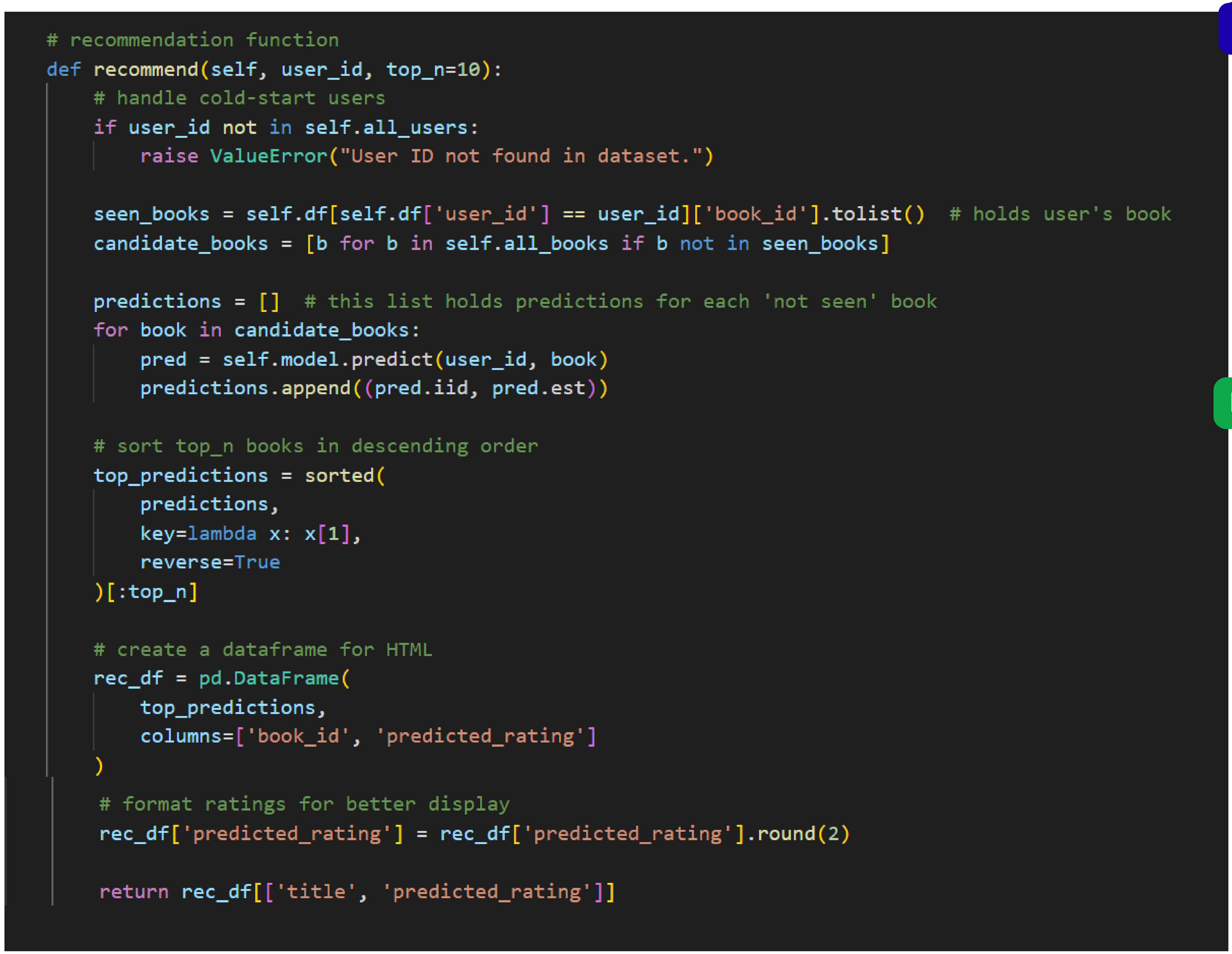
Book Recommendation System with SVD

Book Recommendation System with SVD



```python
import pandas as pd
import random
import pickle
from surprise import SVD


class SVDRecommender:
    """
    This class loads a trained SVD recommendation model
    and generates personalized book recommendations for users.
    """

    def __init__(self, model_path, data_path):
        # load the trained model
        with open(model_path, "rb") as f:
            self.model = pickle.load(f)

        # load data
        self.df = pd.read_csv(data_path)

        # cache for speed
        self.all_books = self.df['book_id'].unique()  # all book's id in dataset
        self.all_users = self.df['user_id'].unique()

    # choice a random user function
    def get_random_user(self):
        return random.choice(self.all_users)
```

Book Recommendation System with SVD



```python
# recommendation function
def recommend(self, user_id, top_n=10):
    # handle cold-start users
    if user_id not in self.all_users:
        raise ValueError("User ID not found in dataset.")

    seen_books = self.df[self.df['user_id'] == user_id]['book_id'].tolist()  # holds user's book
    candidate_books = [b for b in self.all_books if b not in seen_books]

    predictions = []  # this list holds predictions for each 'not seen' book
    for book in candidate_books:
        pred = self.model.predict(user_id, book)
        predictions.append((pred.iid, pred.est))

    # sort top_n books in descending order
    top_predictions = sorted(
        predictions,
        key=lambda x: x[1],
        reverse=True
    )[:top_n]

    # create a dataframe for HTML
    rec_df = pd.DataFrame(
        top_predictions,
        columns=['book_id', 'predicted_rating']
    )
    # format ratings for better display
    rec_df['predicted_rating'] = rec_df['predicted_rating'].round(2)

    return rec_df[['title', 'predicted_rating']]
```

seen_books holds the books that have already been rated by the user (user_id).

All books in the DataFrame are stored in all_books.

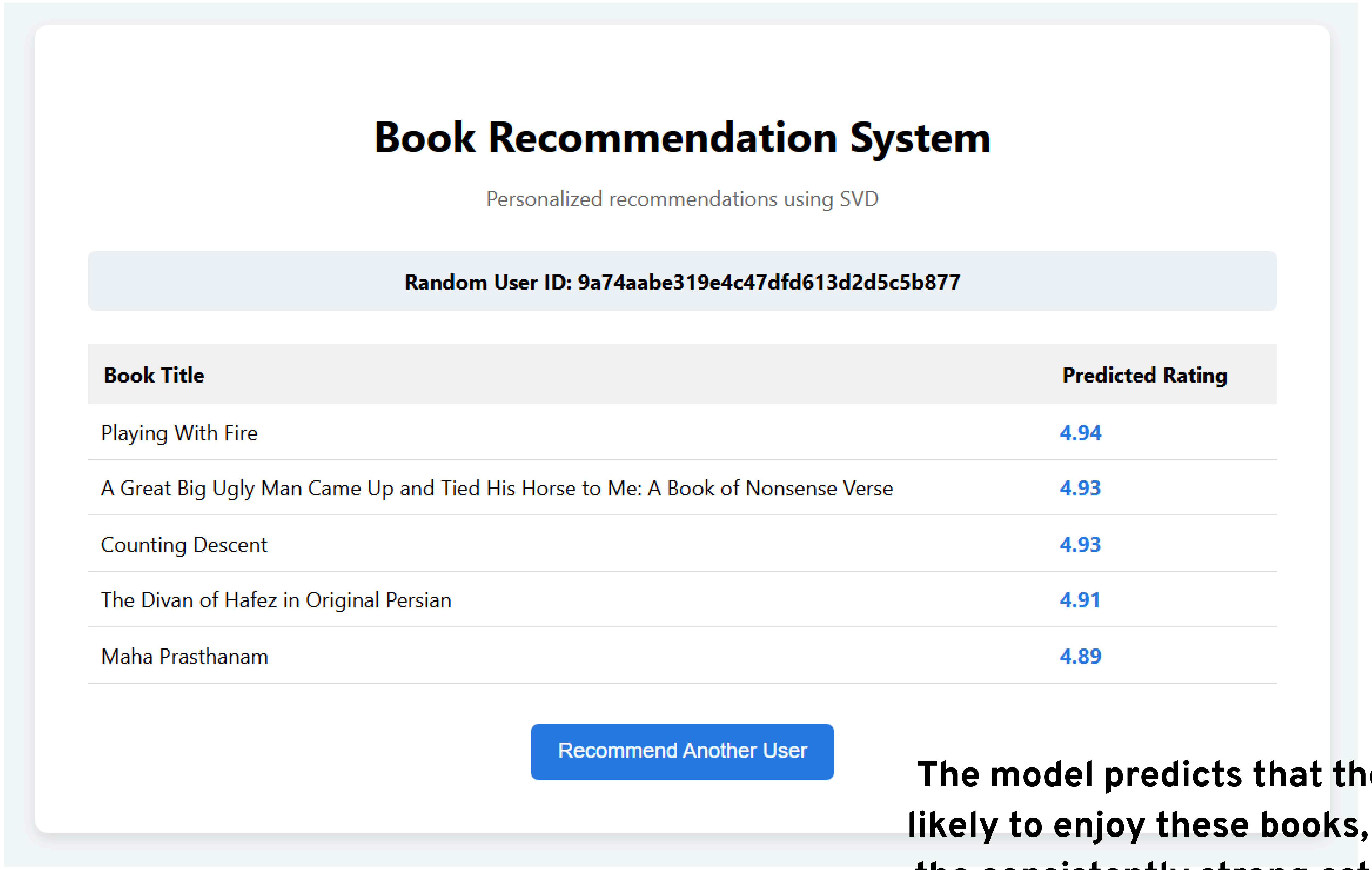
candidate_books contains all books that the user has not rated yet.

predictions contains the estimated ratings for each book in candidate_books predicted by the SVD model.

top_predictions contains the highest predicted ratings from predictions, sorted in descending order, limited to top_n books.

recommended extracts the book IDs and their predicted ratings from top_predictions to prepare for creating a DataFrame with titles.

Book Recommendation System with SVD



Book Recommendation System

Personalized recommendations using SVD

Random User ID: 9a74aabe319e4c47dfd613d2d5c5b877

Book Title	Predicted Rating
Playing With Fire	4.94
A Great Big Ugly Man Came Up and Tied His Horse to Me: A Book of Nonsense Verse	4.93
Counting Descent	4.93
The Divan of Hafez in Original Persian	4.91
Maha Prasthanam	4.89

The model predicts that the user is highly likely to enjoy these books, as indicated by the consistently strong estimated ratings.