**EVALUATION (ELITE TEAM-3)**

*Abduhakimov Fayzullo*

*Jo’rayev Xudoyshukur*

*Jumaniyoz Botirov*

**OVERVIEW**

Evaluation metrics are quantitative measures used to assess the performance of a model or system in various domains, such as machine learning, data analysis, and information retrieval. These metrics provide a way to gauge how well a model is performing and to compare different models or algorithms.

Here are some common evaluation metrics used in different contexts:

* Accuracy: The proportion of correctly classified instances out of the total instances. It is a basic measure of overall correctness but may not be suitable for imbalanced datasets.
* Precision: The ratio of true positive predictions to the total predicted positives. It measures the accuracy of positive predictions and is particularly relevant when the cost of false positives is high.
* Recall (Sensitivity or True Positive Rate): The ratio of true positive predictions to the total actual positives. It quantifies the ability of a model to capture all relevant instances and is crucial when the cost of false negatives is high.
* F1 Score: The harmonic mean of precision and recall. It provides a balance between precision and recall, especially in situations where there is an imbalance between positive and negative instances.
* Specificity: The ratio of true negative predictions to the total actual negatives. It is essential when the focus is on correctly identifying negative instances.
* Area Under the Receiver Operating Characteristic (ROC) Curve (AUC-ROC): A performance measurement for the classification problems at various threshold settings. It plots the true positive rate against the false positive rate, providing a comprehensive view of the model's performance across different thresholds.
* Mean Squared Error (MSE): Commonly used in regression problems, it measures the average squared difference between predicted and actual values.
* Root Mean Squared Error (RMSE): The square root of the mean squared error, providing a measure of the average magnitude of errors.
* Mean Absolute Error (MAE): The average absolute difference between predicted and actual values.
* Confusion Matrix: A table that describes the performance of a classification model, presenting the counts of true positive, true negative, false positive, and false negative predictions.

The choice of evaluation metric depends on the specific goals and characteristics of the problem at hand. It is often advisable to consider multiple metrics to gain a comprehensive understanding of a model's performance.

**EVALUATION OF OUR RESEARCH PROJECT**

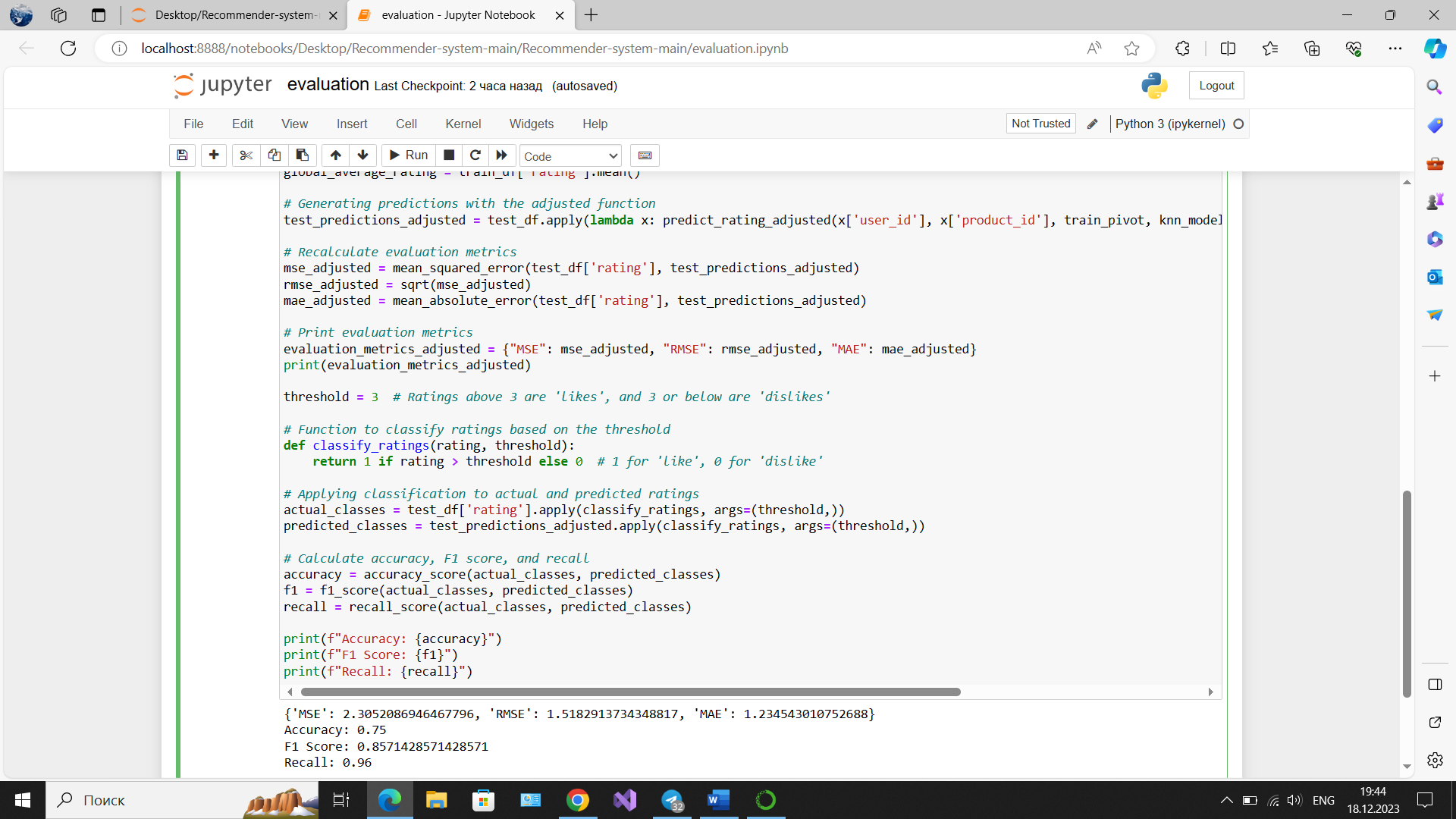


Figure 1. Evaluation metrics

As can be seen from Figure 1, the results are as below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| MSE | RMSE | MAE | Accuracy | F1 Score | Recall |
| 2.3 | 1.5 | 1.2 | 0.75 | 0.86 | 0.96 |

*MSE –* Mean Squared Error.

*RMSE –* Root Mean Squared Error.

MSE and RMSE measure the average squared difference between the actual ratings and the predicted ratings. RMSE is the square root of MSE. Lower values indicate better performance.

*MAE – Mean Absolute Error.* This metric measures the average absolute difference between the actual ratings and the predicted ratings. Like MSE, lower values indicate better accuracy.

*Precision and Recall.* Precision measures the proportion of recommended items that are relevant, while recall measures the proportion of relevant items that are recommended. These are particularly useful when the recommendation system is treated as a binary classification problem (relevant vs. non-relevant).

*F1-Score* - the harmonic mean of precision and recall. It's a single metric that balances both the precision and recall.

To calculate F1-Score, Recall, Confusion Matrix and Accuracy, we first needed to transform the rating prediction problem into a binary classification problem. This involves setting a threshold to categorize ratings into two classes like ‘likes’ or ‘dislikes’. Due to the fact that ratings are on a scale from 1 to 5, we chose a threshold such that ratings above 3 were considered "likes" and ratings 3 or below were "dislikes." Once this classification was done, we were able to calculate accuracy, F1 score, and recall (see Figure 1). Besides, we decided to calculate confusion matrix as well.

Code:

*conf\_matrix = confusion\_matrix(actual\_classes, predicted\_classes)*

*# Plotting the confusion matrix using seaborn*

*plt.figure(figsize=(8, 6))*

*sns.heatmap(conf\_matrix, annot=True, fmt='g', cmap='Blues')*

*plt.title('Confusion Matrix')*

*plt.xlabel('Predicted Labels')*

*plt.ylabel('Actual Labels')*

*plt.xticks(ticks=[0.5, 1.5], labels=['Dislike', 'Like'])*

*plt.yticks(ticks=[0.5, 1.5], labels=['Dislike', 'Like'])*

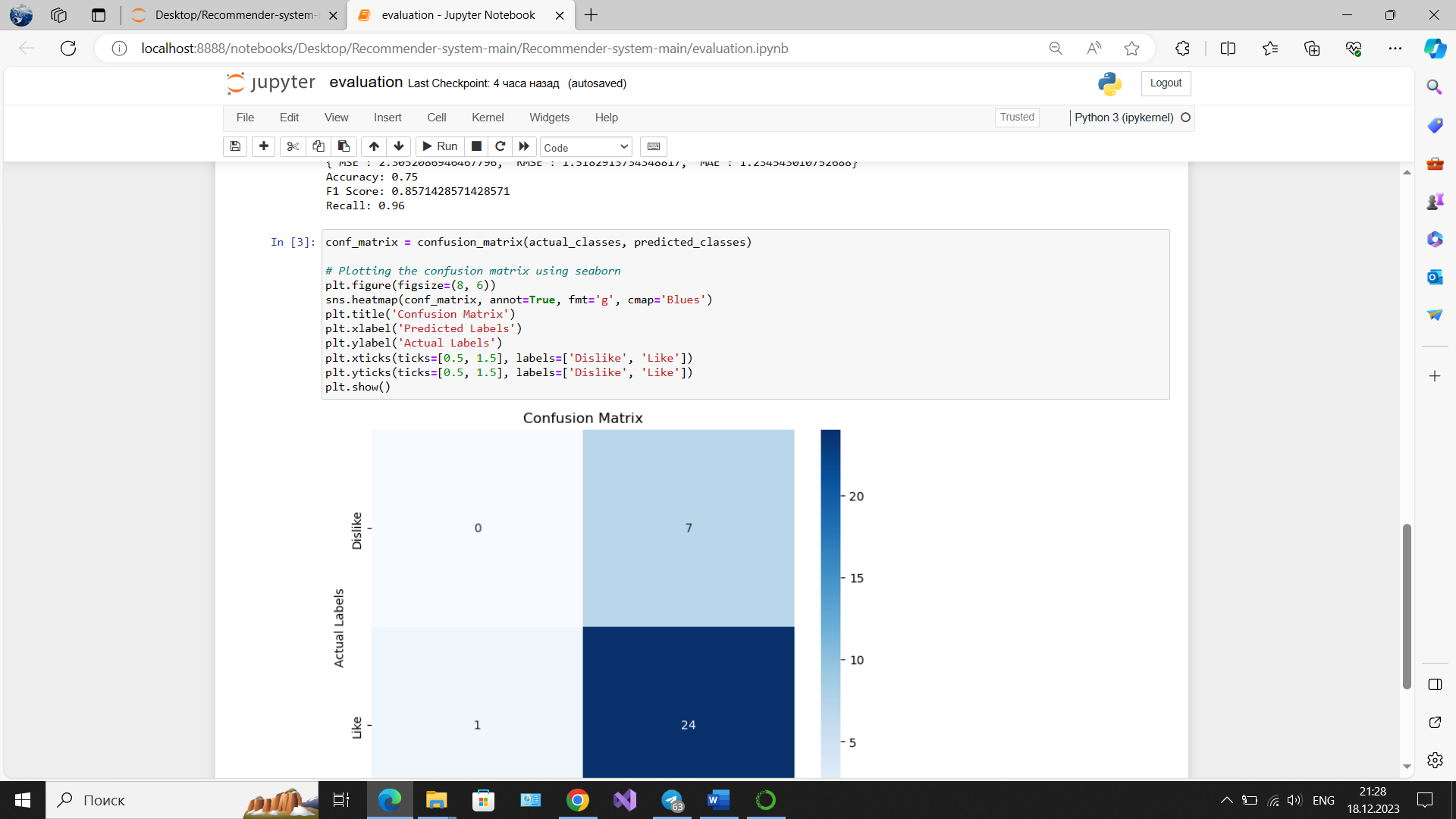
*plt.show()*

Figure 2. Confusion Matrix

Let’s compare our KNN method to SVM method:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Method | MSE | RMSE | MAE | Accuracy | F1 Score | Recall |
| **KNN** | 2.3 | 1.5 | 1.2 | 0.75 | 0.86 | 0.96 |
| **SVM** | 0.03 | 0.17 | 0.02 | 0.99 | 0.97 | 0.95 |

Note that while implementing SVM method, we had to implement normalization in order to calculate accuracy, f1-score, etc. Overall, KNN performs slightly better than SVM method. There is an article related to recommender systems based on support vector machines [1] in which the accuracy of the model is below seventy per cent.

[1] Min, S. H., & Han, I. (2005, July). Recommender systems using support vector machines. In *International Conference on Web Engineering* (pp. 387-393). Berlin, Heidelberg: Springer Berlin Heidelberg.

**This is our evaluation code:**

*import matplotlib.pyplot as plt*

*import seaborn as sns*

*import pandas as pd*

*import numpy as np*

*from sklearn.model\_selection import train\_test\_split*

*from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, confusion\_matrix*

*from math import sqrt*

*from scipy.sparse import csr\_matrix*

*from sklearn.neighbors import NearestNeighbors*

*from sklearn.metrics import accuracy\_score, f1\_score, recall\_score*

*import ast*

*# Load and prepare the ratings data*

*with open('ratings.py', 'r') as file:*

*ratings\_content = file.read()*

*ratings\_list = ast.literal\_eval(ratings\_content.split('=')[1].strip())*

*ratings\_df = pd.DataFrame(ratings\_list)*

*# Split the data into training and testing sets*

*train\_df, test\_df = train\_test\_split(ratings\_df, test\_size=0.2, random\_state=42)*

*# Function to prepare data for KNN*

*def prepare\_data\_for\_knn(data):*

*pivot\_data = data.pivot(index='product\_id', columns='user\_id', values='rating').fillna(0)*

*matrix\_data = csr\_matrix(pivot\_data.values)*

*return pivot\_data, matrix\_data*

*train\_pivot, train\_matrix = prepare\_data\_for\_knn(train\_df)*

*test\_pivot, test\_matrix = prepare\_data\_for\_knn(test\_df)*

*# Initialize and train the KNN model*

*knn\_model = NearestNeighbors(metric='cosine', algorithm='brute')*

*knn\_model.fit(train\_matrix)*

*# Adjusted predict\_rating function*

*def predict\_rating\_adjusted(user\_id, product\_id, pivot\_data, model, global\_avg\_rating):*

*if user\_id in pivot\_data.columns and product\_id in pivot\_data.index:*

*user\_idx = list(pivot\_data.columns).index(user\_id)*

*product\_idx = list(pivot\_data.index).index(product\_id)*

*distances, indices = model.kneighbors(pivot\_data.iloc[product\_idx, :].values.reshape(1, -1), n\_neighbors=7)*

*neighbor\_ratings = [pivot\_data.iloc[indices.flatten()[i], user\_idx] for i in range(1, len(distances.flatten())) if pivot\_data.iloc[indices.flatten()[i], user\_idx] > 0]*

*if neighbor\_ratings:*

*return np.mean(neighbor\_ratings)*

*else:*

*user\_avg = pivot\_data.iloc[:, user\_idx][pivot\_data.iloc[:, user\_idx] > 0].mean()*

*product\_avg = pivot\_data.iloc[product\_idx, :][pivot\_data.iloc[product\_idx, :] > 0].mean()*

*return user\_avg if not np.isnan(user\_avg) else product\_avg if not np.isnan(product\_avg) else global\_avg\_rating*

*else:*

*return global\_avg\_rating*

*# Calculate the global average rating*

*global\_average\_rating = train\_df['rating'].mean()*

*# Generating predictions with the adjusted function*

*test\_predictions\_adjusted = test\_df.apply(lambda x: predict\_rating\_adjusted(x['user\_id'], x['product\_id'], train\_pivot, knn\_model, global\_average\_rating), axis=1)*

*# Recalculate evaluation metrics*

*mse\_adjusted = mean\_squared\_error(test\_df['rating'], test\_predictions\_adjusted)*

*rmse\_adjusted = sqrt(mse\_adjusted)*

*mae\_adjusted = mean\_absolute\_error(test\_df['rating'], test\_predictions\_adjusted)*

*# Print evaluation metrics*

*evaluation\_metrics\_adjusted = {"MSE": mse\_adjusted, "RMSE": rmse\_adjusted, "MAE": mae\_adjusted}*

*print(evaluation\_metrics\_adjusted)*

*threshold = 3 # Ratings above 3 are 'likes', and 3 or below are 'dislikes'*

*# Function to classify ratings based on the threshold*

*def classify\_ratings(rating, threshold):*

*return 1 if rating > threshold else 0 # 1 for 'like', 0 for 'dislike'*

*# Applying classification to actual and predicted ratings*

*actual\_classes = test\_df['rating'].apply(classify\_ratings, args=(threshold,))*

*predicted\_classes = test\_predictions\_adjusted.apply(classify\_ratings, args=(threshold,))*

*# Calculate accuracy, F1 score, and recall*

*accuracy = accuracy\_score(actual\_classes, predicted\_classes)*

*f1 = f1\_score(actual\_classes, predicted\_classes)*

*recall = recall\_score(actual\_classes, predicted\_classes)*

*print(f"Accuracy: {accuracy}")*

*print(f"F1 Score: {f1}")*

*print(f"Recall: {recall}")*

SVM method:

*import pandas as pd*

*import numpy as np*

*from sklearn.decomposition import TruncatedSVD*

*from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, accuracy\_score, f1\_score, recall\_score*

*from scipy.sparse import csr\_matrix*

*from math import sqrt*

*import ast*

*# Load and prepare the ratings data*

*with open('ratings.py', 'r') as file:*

*ratings\_content = file.read()*

*ratings\_list = ast.literal\_eval(ratings\_content.split('=')[1].strip())*

*ratings\_df = pd.DataFrame(ratings\_list)*

*# Prepare the user-item matrix*

*pivot\_table = ratings\_df.pivot\_table(index='user\_id', columns='product\_id', values='rating').fillna(0)*

*X = csr\_matrix(pivot\_table.values)*

*# Apply SVD*

*svd = TruncatedSVD(n\_components=20, random\_state=42)*

*X\_reduced = svd.fit\_transform(X)*

*# Inverse transform to get rating predictions*

*X\_pred = svd.inverse\_transform(X\_reduced)*

*# Flatten the matrices for calculating regression metrics*

*original\_ratings = X.toarray().flatten()*

*predicted\_ratings = X\_pred.flatten()*

*# Calculate RMSE, MSE, and MAE*

*mse = mean\_squared\_error(original\_ratings, predicted\_ratings)*

*rmse = sqrt(mse)*

*mae = mean\_absolute\_error(original\_ratings, predicted\_ratings)*

*# Define a threshold for classification*

*threshold = 3*

*actual\_classes = [1 if rating > threshold else 0 for rating in original\_ratings]*

*predicted\_classes = [1 if rating > threshold else 0 for rating in predicted\_ratings]*

*# Calculate accuracy, F1 score, and recall*

*accuracy = accuracy\_score(actual\_classes, predicted\_classes)*

*f1 = f1\_score(actual\_classes, predicted\_classes)*

*recall = recall\_score(actual\_classes, predicted\_classes)*

*# Print evaluation metrics*

*print(f'RMSE: {rmse}')*

*print(f'MSE: {mse}')*

*print(f'MAE: {mae}')*

*print(f'Accuracy: {accuracy}')*

*print(f'F1 Score: {f1}')*

*print(f'Recall: {recall}')*