For Deliverable Three, we opted for an XGBoost model due to its compatibility with our dataset. This choice is motivated by two key factors. Firstly, our dataset comprises exclusively of numeric values, aligning with XGBoost's requirement for numeric input. Secondly, given that our task involves binary classification, XGBoost stands out as a excellent algorithm for such scenarios.

Once the model selection was finalized, we initiated the data pre-processing phase to mitigate potential biases stemming from the dataset encoding. An illustrative instance of this lies in the "Education" feature, originally labeled 0-6. To address this, we transformed it into seven distinct features for education, each represented by a binary value (0 or 1). The following code snippet demonstrates the application of this one-hot encoding technique. Additional details on our data pre-processing steps are available in the "Data Pre-process.ipynb" file within this repository.

[13]:
```
encoder=OneHotEncoder(sparse_output=False)
df_encoded = pd.DataFrame (encoder.fit_transform(df[['EDUCATION']]))
df_encoded.columns = encoder.get_feature_names_out(['EDUCATION'])
df.drop(['EDUCATION'] ,axis=1, inplace=True)
df_OH= pd.concat([df, df_encoded ], axis=1)
df = df_OH
df.head(5)
```

[13]:

	ID	LIMIT_BAL	SEX	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	...	PAY_AMT5	PAY_AMT6	default payment next month	EDUCATION_0	EDUCATION_1	EDUCATION_2	EDUCATION_3	EDUCATION_4	EDU(
0	1	20000	2	1	24	2	2	-1	-1	-2	...	0	0	1	0.0	0.0	1.0	0.0	0.0	
1	2	120000	2	2	26	-1	2	0	0	0	...	0	2000	1	0.0	0.0	1.0	0.0	0.0	
2	3	90000	2	2	34	0	0	0	0	0	...	1000	5000	0	0.0	0.0	1.0	0.0	0.0	
3	4	50000	2	1	37	0	0	0	0	0	...	1069	1000	0	0.0	0.0	1.0	0.0	0.0	
4	5	50000	1	1	57	-1	0	-1	0	0	...	689	679	0	0.0	0.0	1.0	0.0	0.0	

5 rows × 31 columns

After completing the data pre-processing phase, our focus shifted to constructing the model. We leveraged Amazon SageMaker in conjunction with S3 buckets for this purpose. The selection of SageMaker was motivated by our project's ultimate objective, which involves creating an API to interact with a user interface. SageMaker endpoints were chosen to facilitate hosting the model seamlessly.

Additionally, in the model-building process, we made a deliberate decision to forgo hyperparameter tuning. XGBoost hyperparameter tuning is known to be a challenging task, and we determined that the associated costs might not justify the benefits, especially considering the need to allocate budget resources towards hosting a frontend web application and an API for user communication. Furthermore, our model's performance is deemed satisfactory for our informal application, eliminating the imperative for extensive hyperparameter tuning.

For a glimpse into some of our results, please refer to the following section.

[15]:
```
predictions = predictor.predict(predict_data.values).decode('utf-8')
predictions = [float(prediction) for prediction in predictions.split(',')]
actual_labels = test_data.iloc[:, 0].values
predicted_labels = [1 if prediction > 0.5 else 0 for prediction in predictions]
accuracy = sum(actual_labels == predicted_labels) / len(actual_labels)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

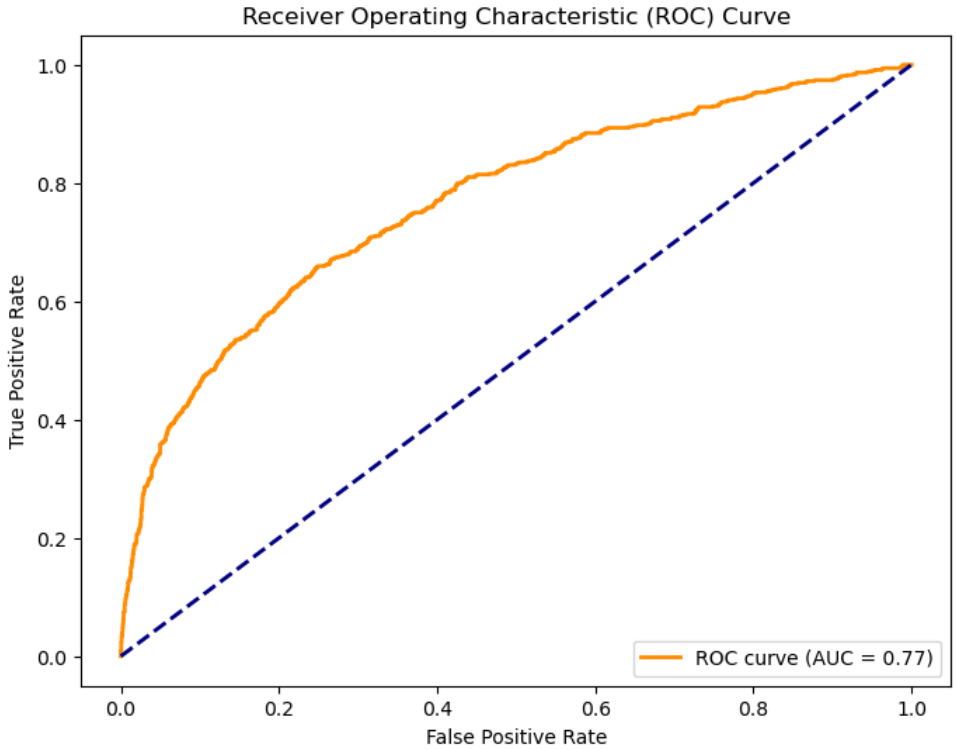
```
Accuracy: 81.23%
```

[16]:
```
# MSE
true_labels = test_data.iloc[:, 0].values
squared_errors = (predictions - true_labels) ** 2
mse = np.mean(squared_errors)
print(f'Mean Squared Error (MSE): {mse:.4f}')
```

```
Mean Squared Error (MSE): 0.1412
```

[17]:
```
# Precision
binary_predictions = [1 if prediction > 0.5 else 0 for prediction in predictions]
precision = precision_score(true_labels, binary_predictions)
print(f'Precision: {precision:.4f}')
```

```
Precision: 0.6596
```



To explore the code employed in crafting our XGBoost model, refer to the "Model Training and Eval.ipynb" file within this repository.