Lab 3.7 - Student Notebook

Lab setup

Because this solution is split across several labs in the module, you run the following cells so that you can load the data and train the model to be deployed.

Note: The setup can take up to 5 minutes to complete.

Importing the data, and training, testing and validating the model

By running the following cells, the data will be imported, and the model will be trained, tested and validated and ready for use.

Note: **The following cells represent the key steps in the previous labs.**

In order to tune the model it must be ready, then you can tweak the mdoel with hyperparameters later in step 2.

In [1]:
```
bucket='c169682a4380825l111216536t1w526205326733-labbucket-wpgdpf2be4p8'
```

In [2]:
```
import time
start = time.time()
import warnings, requests, zipfile, io
warnings.simplefilter('ignore')
import pandas as pd
from scipy.io import arff

import os
import boto3
import sagemaker
from sagemaker.image_uris import retrieve
from sklearn.model_selection import train_test_split

from sklearn.metrics import roc_auc_score, roc_curve, auc, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

```
sagemaker.config INFO - Not applying SDK defaults from location: /etc/xdg/sagem
aker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /home/ec2-use
r/.config/sagemaker/config.yaml
Matplotlib is building the font cache; this may take a moment.
```

Note: **The following cell takes approximately 10 minutes to complete. Observe the code and how it processes, this will help you to better understand what is**

going on in the background. Keep in mind that this cell completes all the steps you did in previous labs in this module including:

- **Importing the data**
- **Loading the data into a dataframe**
- **Splitting the data into training, test and validation datasets**
- **Uploading the split datasets to S3**
- **Training, testing and validating the model with the datasets**

In [3]:
```
%%time

def plot_roc(test_labels, target_predicted_binary):
    TN, FP, FN, TP = confusion_matrix(test_labels, target_predicted_binary).rave
    # Sensitivity, hit rate, recall, or true positive rate
    Sensitivity  = float(TP)/(TP+FN)*100
    # Specificity or true negative rate
    Specificity  = float(TN)/(TN+FP)*100
    # Precision or positive predictive value
    Precision = float(TP)/(TP+FP)*100
    # Negative predictive value
    NPV = float(TN)/(TN+FN)*100
    # Fall out or false positive rate
    FPR = float(FP)/(FP+TN)*100
    # False negative rate
    FNR = float(FN)/(TP+FN)*100
    # False discovery rate
    FDR = float(FP)/(TP+FP)*100
    # Overall accuracy
    ACC = float(TP+TN)/(TP+FP+FN+TN)*100

    print(f"Sensitivity or TPR: {Sensitivity}%")
    print(f"Specificity or TNR: {Specificity}%")
    print(f"Precision: {Precision}%")
    print(f"Negative Predictive Value: {NPV}%")
    print( f"False Positive Rate: {FPR}%")
    print(f"False Negative Rate: {FNR}%")
    print(f"False Discovery Rate: {FDR}%" )
    print(f"Accuracy: {ACC}%")

    test_labels = test.iloc[:,0];
    print("Validation AUC", roc_auc_score(test_labels, target_predicted_binary)

    fpr, tpr, thresholds = roc_curve(test_labels, target_predicted_binary)
    roc_auc = auc(fpr, tpr)

    plt.figure()
    plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % (roc_auc))
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc="lower right")

    # create the axis of thresholds (scores)
    ax2 = plt.gca().twinx()
    ax2.plot(fpr, thresholds, markeredgecolor='r',linestyle='dashed', color='r')
    ax2.set_ylabel('Threshold',color='r')
    ax2.set_ylim([thresholds[-1],thresholds[0]])
    ax2.set_xlim([fpr[0],fpr[-1]])

    print(plt.figure())

def plot_confusion_matrix(test_labels, target_predicted):
    matrix = confusion_matrix(test_labels, target_predicted)
    df_confusion = pd.DataFrame(matrix)
    colormap = sns.color_palette("BrBG", 10)
    sns.heatmap(df_confusion, annot=True, fmt='.2f', cbar=None, cmap=colormap)
    plt.title("Confusion Matrix")
    plt.tight_layout()
    plt.ylabel("True Class")
    plt.xlabel("Predicted Class")
    plt.show()


f_zip = 'http://archive.ics.uci.edu/ml/machine-learning-databases/00212/vertebra
r = requests.get(f_zip, stream=True)
Vertebral_zip = zipfile.ZipFile(io.BytesIO(r.content))
Vertebral_zip.extractall()

data = arff.loadarff('column_2C_weka.arff')
df = pd.DataFrame(data[0])

class_mapper = {b'Abnormal':1,b'Normal':0}
df['class']=df['class'].replace(class_mapper)

cols = df.columns.tolist()
cols = cols[-1:] + cols[:-1]
df = df[cols]

train, test_and_validate = train_test_split(df, test_size=0.2, random_state=42,
test, validate = train_test_split(test_and_validate, test_size=0.5, random_state

prefix='lab3'

train_file='vertebral_train.csv'
test_file='vertebral_test.csv'
validate_file='vertebral_validate.csv'

s3_resource = boto3.Session().resource('s3')
def upload_s3_csv(filename, folder, dataframe):
    csv_buffer = io.StringIO()
    dataframe.to_csv(csv_buffer, header=False, index=False )
    s3_resource.Bucket(bucket).Object(os.path.join(prefix, folder, filename)).pu

upload_s3_csv(train_file, 'train', train)
upload_s3_csv(test_file, 'test', test)
upload_s3_csv(validate_file, 'validate', validate)

container = retrieve('xgboost',boto3.Session().region_name,'1.0-1')

hyperparams={"num_round":"42",
             "eval_metric": "auc",
             "objective": "binary:logistic",
             "silent" : 1}

s3_output_location="s3://{}/{}/output/".format(bucket,prefix)
```

```
xgb_model=sagemaker.estimator.Estimator(container,
                                        sagemaker.get_execution_role(),
                                        instance_count=1,
                                        instance_type='ml.m5.2xlarge',
                                        output_path=s3_output_location,
                                         hyperparameters=hyperparams,
                                         sagemaker_session=sagemaker.Session())

train_channel = sagemaker.inputs.TrainingInput(
    "s3://{}/{}/train/".format(bucket,prefix,train_file),
    content_type='text/csv')

validate_channel = sagemaker.inputs.TrainingInput(
    "s3://{}/{}/validate/".format(bucket,prefix,validate_file),
    content_type='text/csv')

data_channels = {'train': train_channel, 'validation': validate_channel}

xgb_model.fit(inputs=data_channels, logs=False)

batch_X = test.iloc[:,1:];

batch_X_file='batch-in.csv'
upload_s3_csv(batch_X_file, 'batch-in', batch_X)

batch_output = "s3://{}/{}/batch-out/".format(bucket,prefix)
batch_input = "s3://{}/{}/batch-in/{}".format(bucket,prefix,batch_X_file)

xgb_transformer = xgb_model.transformer(instance_count=1,
                                        instance_type='ml.m5.2xlarge',
                                        strategy='MultiRecord',
                                        assemble_with='Line',
                                        output_path=batch_output)

xgb_transformer.transform(data=batch_input,
                          data_type='S3Prefix',
                          content_type='text/csv',
                          split_type='Line')
xgb_transformer.wait(logs=False)
```

```
INFO:sagemaker:Creating training-job with name: sagemaker-xgboost-2025-08-14-14
-05-33-905
2025-08-14 14:05:38 Starting - Starting the training job..
2025-08-14 14:05:53 Starting - Preparing the instances for training..
2025-08-14 14:06:09 Downloading - Downloading input data...
2025-08-14 14:06:24 Downloading - Downloading the training image.......
2025-08-14 14:07:05 Training - Training image download completed. Training in p
rogress......
2025-08-14 14:07:36 Uploading - Uploading generated training model.
2025-08-14 14:07:49 Completed - Training job completed
INFO:sagemaker:Creating model with name: sagemaker-xgboost-2025-08-14-14-07-50-
904
INFO:sagemaker:Creating transform job with name: sagemaker-xgboost-2025-08-14-1
4-07-51-410
............................
..!
CPU times: user 1.57 s, sys: 163 ms, total: 1.73 s
Wall time: 7min 46s
```

Step 1: Getting model statistics

Before you tune the model, re-familiarize yourself with the current model's metrics.

The setup performed a batch prediction, so you must read in the results from Amazon Simple Storage Service (Amazon S3).

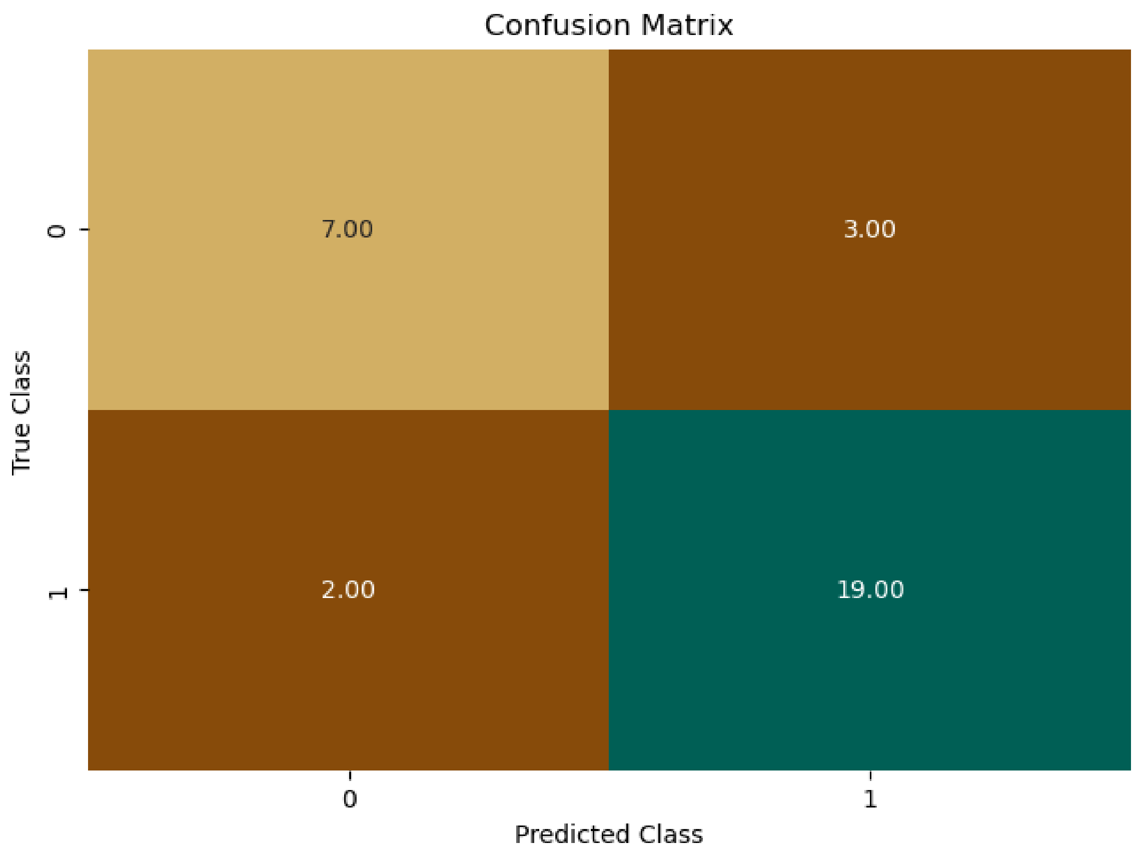
In [4]:
```
s3 = boto3.client('s3')
obj = s3.get_object(Bucket=bucket, Key="{}/batch-out/{}".format(prefix,'batch-in
target_predicted = pd.read_csv(io.BytesIO(obj['Body'].read()),names=['class'])

def binary_convert(x):
    threshold = 0.5
    if x > threshold:
        return 1
    else:
        return 0

target_predicted_binary = target_predicted['class'].apply(binary_convert)
test_labels = test.iloc[:,0]
```

Plot the confusion matrix and the receiver operating characteristic (ROC) curve for the original model.

In [5]:
```
plot_confusion_matrix(test_labels, target_predicted_binary)
```



In [11]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
```

```
import numpy as np

def plot_roc(y_true, y_pred):
    """
    Plots ROC curve safely.
    y_true: array-like of shape (n_samples,) -> true binary labels (0 or 1)
    y_pred: array-like of shape (n_samples,) -> predicted scores or probabilitie
    """
    # Convert inputs to numpy arrays
    y_true = np.array(y_true)
    y_pred = np.array(y_pred)

    # Ensure both classes are present in y_true
    if len(np.unique(y_true)) < 2:
        print("ROC curve is undefined — only one class present in labels.")
        return

    # Compute ROC curve
    try:
        fpr, tpr, _ = roc_curve(y_true, y_pred)
    except ValueError as e:
        print(f"Error computing ROC: {e}")
        return

    # Replace NaN/Inf with safe numbers
    fpr = np.nan_to_num(fpr, nan=0.0, posinf=1.0, neginf=0.0)
    tpr = np.nan_to_num(tpr, nan=0.0, posinf=1.0, neginf=0.0)

    # Compute AUC
    roc_auc = auc(fpr, tpr)

    # Plot ROC
    plt.figure(figsize=(6, 6))
    plt.plot(fpr, tpr, color="blue", lw=2, label=f"AUC = {roc_auc:.2f}")
    plt.plot([0, 1], [0, 1], color="red", lw=2, linestyle="--")
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.0])
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("Receiver Operating Characteristic (ROC)")
    plt.legend(loc="lower right")
    plt.grid(True)
    plt.show()
```

This plot gives you a starting point. Make a note of the *Validation area under the curve (AUC)*. You will use it later to check your tuned model to see if it's better.

Step 2: Creating a hyperparameter tuning job

A hyperparameter tuning job can take several hours to complete, depending on the value ranges that you provide. To simplify this task, the parameters used in this step are a subset of the recommended ranges. They were tuned to give good results in this lab, without taking multiple hours to complete.

For more information about the parameters to tune for XGBoost, see Tune an XGBoost Model in the AWS Documentation.

Because this next cell can take approximately 45 minutes to complete, go ahead and run the cell. You will examine what's happening, and why these hyperparameter ranges were chosen.

In [10]:
```
%%time
from sagemaker.tuner import IntegerParameter, CategoricalParameter, ContinuousPa

xgb = sagemaker.estimator.Estimator(container,
                                    role=sagemaker.get_execution_role(),
                                    instance_count= 1, # make sure you have limi
                                    instance_type='ml.m4.xlarge',
                                    output_path='s3://{}/{}/output'.format(bucke
                                    sagemaker_session=sagemaker.Session())


xgb.set_hyperparameters(eval_metric='error@.40',
                        objective='binary:logistic',
                        num_round=42)

hyperparameter_ranges = {'alpha': ContinuousParameter(0, 100),
                         'min_child_weight': ContinuousParameter(1, 5),
                         'subsample': ContinuousParameter(0.5, 1),
                         'eta': ContinuousParameter(0.1, 0.3),
                         'num_round': IntegerParameter(1,50)
                         }

objective_metric_name = 'validation:error'
objective_type = 'Minimize'

tuner = HyperparameterTuner(xgb,
                            objective_metric_name,
                            hyperparameter_ranges,
                            max_jobs=10, # Set this to 10 or above depending upo
                            max_parallel_jobs=1,
                            objective_type=objective_type,
                            early_stopping_type='Auto')

tuner.fit(inputs=data_channels, include_cls_metadata=False)
tuner.wait()
```

```
WARNING:sagemaker.estimator:No finished training job found associated with this
estimator. Please make sure this estimator is only used for building workflow c
onfig
WARNING:sagemaker.estimator:No finished training job found associated with this
estimator. Please make sure this estimator is only used for building workflow c
onfig
INFO:sagemaker:Creating hyperparameter tuning job with name: sagemaker-xgboost-
250814-1418
```

```
.......................................................................
!
CPU times: user 975 ms, sys: 68.2 ms, total: 1.04 s
Wall time: 14min 11s
```

First, you will create the model that you want to tune.

```
xgb = sagemaker.estimator.Estimator(container,

role=sagemaker.get_execution_role(),
                                    instance_count= 1, # make
sure you have limit set for these instances
                                    instance_type='ml.m4.xlarge',

output_path='s3://{}/{}/output'.format(bucket, prefix),

sagemaker_session=sagemaker.Session())

xgb.set_hyperparameters(eval_metric='[error@.40]',
                        objective='binary:logistic',
                        num_round=42)
```

Notice that the *eval_metric* of the model was changed to *error@.40*, with a goal of minimizing that value.

error **is the binary classification error rate. It's calculated as *#(wrong cases)/#(all cases)*. For predictions, the evaluation will consider the instances that have a prediction value larger than 0.4 to be positive instances, and the others as negative instances.**

Next, you must specify the hyperparameters that you want to tune, in addition to the ranges that you must select for each parameter.

The hyperparameters that have the largest effect on XGBoost objective metrics are:

- **alpha**
- **min_child_weight**
- **subsample**
- **eta**
- **num_round**

The recommended tuning ranges can be found in the AWS Documentation at Tune an XGBoost Model.

For this lab, you will use a *subset* of values. These values were obtained by running the tuning job with the full range, then minimizing the range so that you can use fewer iterations to get better performance. Though this practice isn't strictly realistic, it prevents you from waiting several hours in this lab for the tuning job to complete.

```
hyperparameter_ranges = {'alpha': ContinuousParameter(0, 100),
                         'min_child_weight':
ContinuousParameter(1, 5),
                         'subsample': ContinuousParameter(0.5,
1),
                         'eta': ContinuousParameter(0.1, 0.3),
                         'num_round': IntegerParameter(1,50)
                         }
```

You must specify how you are rating the model. You could use several different objective metrics, a subset of which applies to a binary classifcation problem. Because the evaluation metric is error**, you set the objective to *error*.**

```
objective_metric_name = 'validation:error'
objective_type = 'Minimize'
```

Finally, you run the tuning job.

```
tuner = HyperparameterTuner(xgb,
                            objective_metric_name,
                            hyperparameter_ranges,
                            max_jobs=10, # Set this to 10 or
above depending upon budget & available time.
                            max_parallel_jobs=1,
                            objective_type=objective_type,
                            early_stopping_type='Auto')

tuner.fit(inputs=data_channels, include_cls_metadata=False)
tuner.wait()
```

Wait until the training job is finished. It might take up to 45 minutes. While you are waiting, observe the job status in the console, as described in the following instructions.

To monitor hyperparameter optimization jobs:

1. **In the AWS Management Console, on the** Services **menu, choose** Amazon SageMaker**.**
2. **Choose** Training > Hyperparameter tuning jobs**.**
3. **You can check the status of each hyperparameter tuning job, its objective metric value, and its logs.**

After the training job is finished, check the job and make sure that it completed successfully.

In [12]:
```
boto3.client('sagemaker').describe_hyper_parameter_tuning_job(
    HyperParameterTuningJobName=tuner.latest_tuning_job.job_name)['HyperParamete
```

Out[12]:
```
'Completed'
```

Step 3: Investigating the tuning job results

Now that the job is complete, there should be 10 completed jobs. One of the jobs should be marked as the best.

You can examine the metrics by getting *HyperparameterTuningJobAnalytics* and loading that data into a pandas DataFrame.

In [13]:
```
from pprint import pprint
from sagemaker.analytics import HyperparameterTuningJobAnalytics

tuner_analytics = HyperparameterTuningJobAnalytics(tuner.latest_tuning_job.name,

df_tuning_job_analytics = tuner_analytics.dataframe()

# Sort the tuning job analytics by the final metrics value
df_tuning_job_analytics.sort_values(
    by=['FinalObjectiveValue'],
    inplace=True,
    ascending=False if tuner.objective_type == "Maximize" else True)

# Show detailed analytics for the top 20 models
df_tuning_job_analytics.head(20)
```

Out[13]:

	alpha	eta	min_child_weight	num_round	subsample	TrainingJobName	Training
0	0.000000	0.193526	4.863646	41.0	0.706201	sagemaker-xgboost-250814-1418-010-0c3547e6	C
2	3.103832	0.132827	1.159917	50.0	0.501484	sagemaker-xgboost-250814-1418-008-ec24d2cb	C
3	7.025730	0.168570	1.778131	50.0	0.983446	sagemaker-xgboost-250814-1418-007-4a668bdf	C
4	0.000000	0.255618	4.987248	50.0	0.769845	sagemaker-xgboost-250814-1418-006-3b51d4c5	C
1	3.467956	0.298108	3.397213	50.0	0.641344	sagemaker-xgboost-250814-1418-009-929c449e	C
5	7.603463	0.269997	4.200754	27.0	0.517949	sagemaker-xgboost-250814-1418-005-43894485	C
9	20.365008	0.128987	4.881620	37.0	0.801484	sagemaker-xgboost-250814-1418-001-065a2f5b	C
7	27.517449	0.124204	4.174839	43.0	0.920480	sagemaker-xgboost-250814-1418-003-72178386	
6	18.787796	0.208230	1.943879	3.0	0.809734	sagemaker-xgboost-250814-1418-004-dc2eb0a6	C
8	95.276137	0.262731	2.891484	16.0	0.549368	sagemaker-xgboost-250814-1418-002-19942f24	

You should be able to see the hyperparameters that were used for each job, along with the score. You could use those parameters and create a model, or you can get the best model from the hyperparameter tuning job.

In [14]:
```
attached_tuner = HyperparameterTuner.attach(tuner.latest_tuning_job.name, sagema
best_training_job = attached_tuner.best_training_job()
```

Now, you must attach to the best training job and create the model.

In [15]:
```
from sagemaker.estimator import Estimator
algo_estimator = Estimator.attach(best_training_job)

best_algo_model = algo_estimator.create_model(env={'SAGEMAKER_DEFAULT_INVOCATION
```

```
2025-08-14 14:30:20 Starting - Found matching resource for reuse
2025-08-14 14:30:20 Downloading - Downloading the training image
2025-08-14 14:30:20 Training - Training image download completed. Training in p
rogress.
2025-08-14 14:30:20 Uploading - Uploading generated training model
2025-08-14 14:30:20 Completed - Resource reused by training job: sagemaker-xgbo
ost-250814-1418-008-ec24d2cb
```

Then, you can use the transform method to perform a batch prediction by using your testing data. Remember that the testing data is data that the model has never seen before.

In [16]:
```
%%time
batch_output = "s3://{}/{}/batch-out/".format(bucket,prefix)
batch_input = "s3://{}/{}/batch-in/{}".format(bucket,prefix,batch_X_file)

xgb_transformer = best_algo_model.transformer(instance_count=1,
                                       instance_type='ml.m4.xlarge',
                                       strategy='MultiRecord',
                                       assemble_with='Line',
                                       output_path=batch_output)


xgb_transformer.transform(data=batch_input,
                         data_type='S3Prefix',
                         content_type='text/csv',
                         split_type='Line')
xgb_transformer.wait(logs=False)
```

```
INFO:sagemaker:Creating model with name: sagemaker-xgboost-2025-08-14-14-33-00-
353
INFO:sagemaker:Creating transform job with name: sagemaker-xgboost-2025-08-14-1
4-33-00-854
.......................................
..!
CPU times: user 779 ms, sys: 24.9 ms, total: 804 ms
Wall time: 6min 58s
```

Get the predicted target and the test labels of the model.

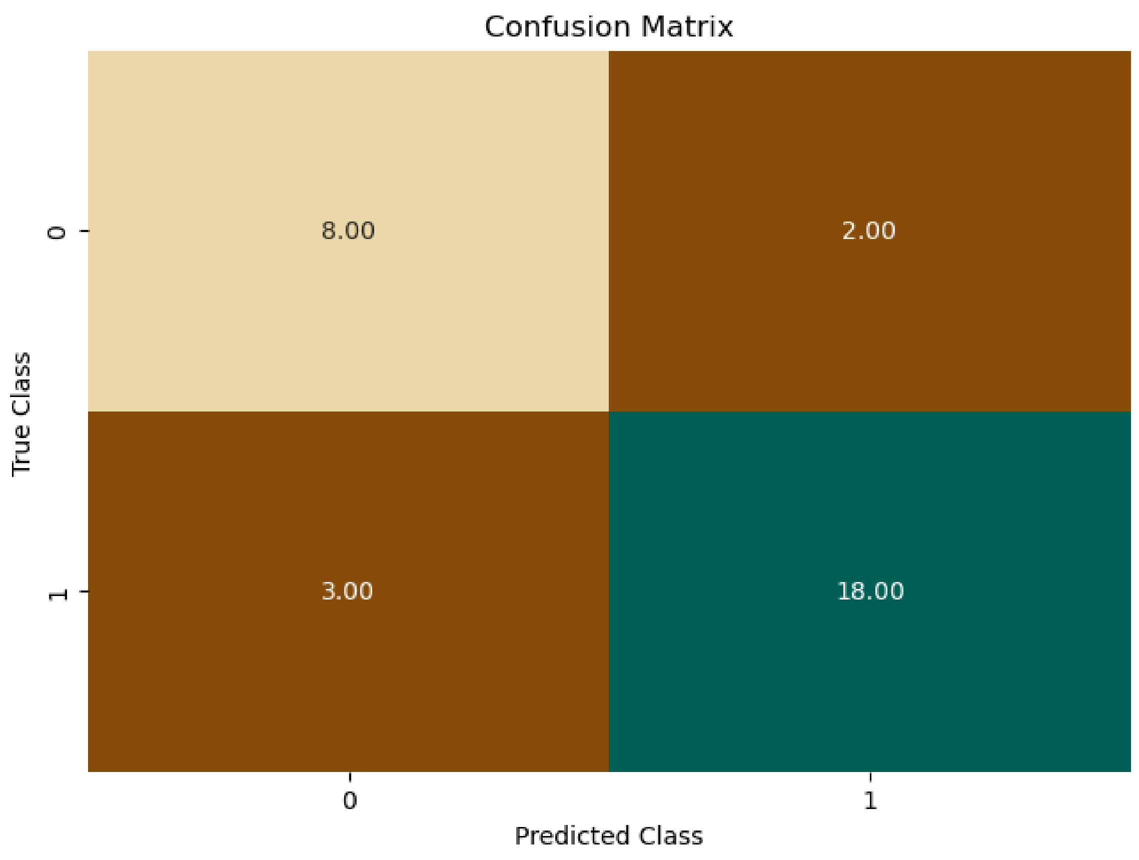
In [17]:
```
s3 = boto3.client('s3')
obj = s3.get_object(Bucket=bucket, Key="{}/batch-out/{}".format(prefix,'batch-in
best_target_predicted = pd.read_csv(io.BytesIO(obj['Body'].read()),names=['class

def binary_convert(x):
    threshold = 0.5
    if x > threshold:
        return 1
    else:
        return 0

best_target_predicted_binary = best_target_predicted['class'].apply(binary_conve
test_labels = test.iloc[:,0]
```

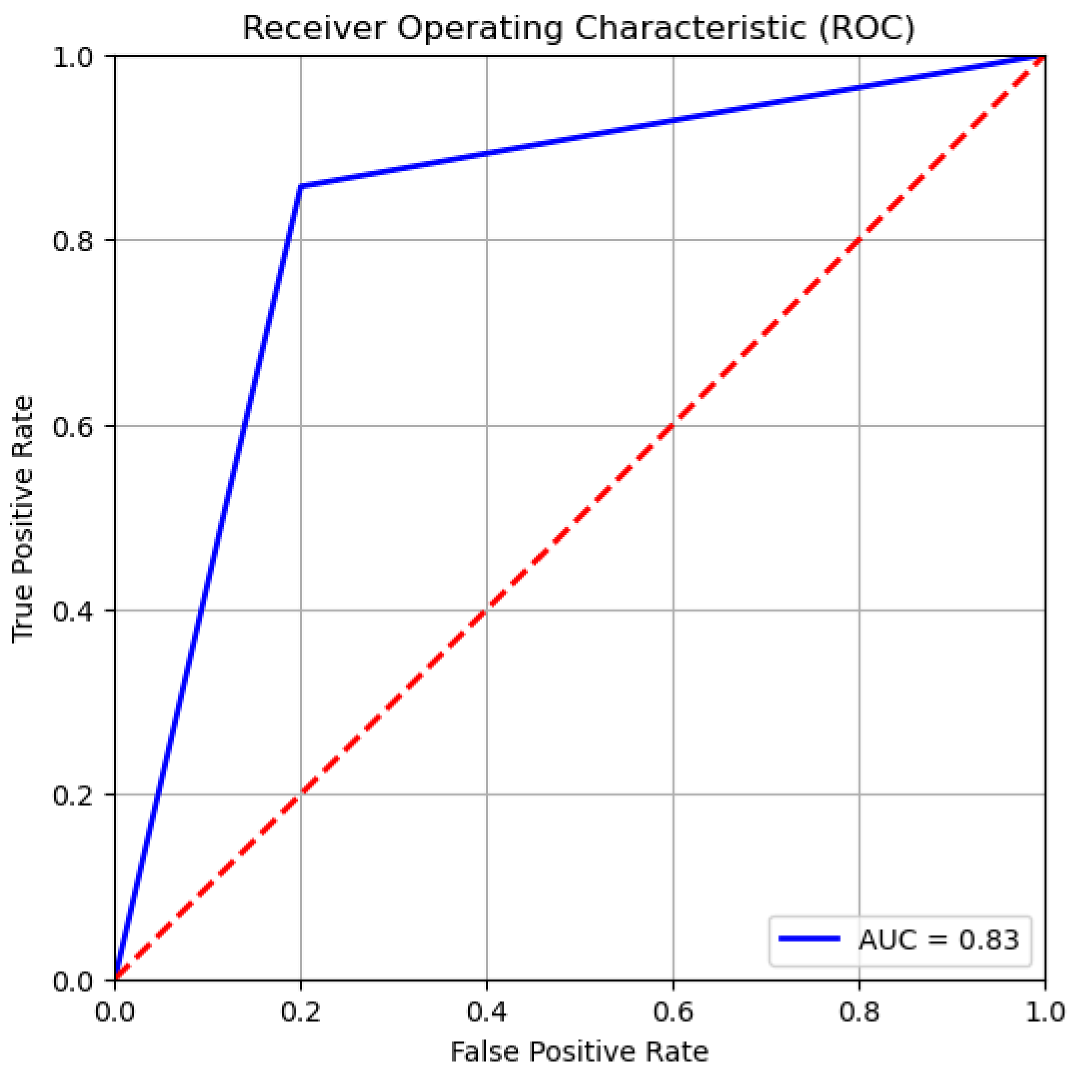
Plot a confusion matrix for your `best_target_predicted` and `test_labels`.

```
In [18]: plot_confusion_matrix(test_labels, best_target_predicted_binary)
```



Plot the ROC chart.

```
In [19]: plot_roc(test_labels, best_target_predicted_binary)
```



Question: How do these results differ from the original? Are these results better or worse?

You might not always see an improvement. There are a few reasons for this result:

- **The model might already be good from the initial pass (what counts as *good* is subjective).**
- **You don't have a large amount of data to train with.**
- **You are using a *subset* of the hyperparameter tuning ranges to save time in this lab.**

Increasing the hyperparameter ranges (as recommended by the documentation) and running more than 30 jobs will typically improve the model. However, this process will take 2-3 hours to complete.

Congratulations!

You have completed this lab, and you can now end the lab by following the lab guide instructions.

```
In [ ]:
```