MIS 451 Assignment 5:

Implementing a Machine Learning pipeline with Amazon SageMaker

Name: Tran Phuong Thao

IRN: 2132300447

Overview

This comprehensive lab series guides students through the end-to-end process of building, training, and deploying machine learning models using Amazon SageMaker. The series is divided into seven focused labs, each addressing a crucial step in the machine learning workflow. Starting with data creation and import, students will progress through data exploration, preprocessing, model training, deployment, performance evaluation, and hyperparameter tuning. By the end of these labs, students will have a thorough understanding of how to utilize Amazon SageMaker for machine learning projects.

Objectives

- By the end of this lab series, students will be able to:
- Create and import data into Amazon SageMaker.
- Perform exploratory data analysis to understand and visualize the data.
- Encode categorical data to prepare it for machine learning models.
- Train machine learning models using SageMaker's built-in algorithms.
- Deploy trained models to SageMaker endpoints for real-time inference.
- Generate and interpret model performance metrics to evaluate accuracy and effectiveness.
- Conduct hyperparameter tuning to optimize model performance.

Requirements:

Use your account at https://awsacademy.instructure.com/ and complete these Labs:

Lab 3.1 - Amazon SageMaker - Creating and importing data

Importing the data

```
[1]: import warnings, requests, zipfile, io
     warnings.simplefilter('ignore')
     import pandas as pd
     from scipy.io import arff
```

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

Lab 3.2 - Amazon SageMaker - Exploring Data

Context: You work for a healthcare provider and want to improve the detection of abnormalities in orthopedic patients.

Business problem: You are tasked with solving this problem by using machine learning (ML). You have access to a dataset that contains six biomechanical features and a target of normal or abnormal. You can use this dataset to train an ML model to predict if a patient will have an abnormality.

Importing the data


```
[1]: import warnings, requests, zipfile, io
     warnings.simplefilter('ignore')
     import pandas as pd
     from scipy.io import arff
```

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


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

⇨ Firstly, importing the data

```
[4]: df.shape
```

```
[4]: (310, 7)
```

You will now get a list of the columns.

```
[5]: df.columns
```

```
[5]: Index(['pelvic_incidence', 'pelvic_tilt', 'lumbar_lordosis_angle',
            'sacral_slope', 'pelvic_radius', 'degree_spondylolisthesis', 'class'],
           dtype='object')
```

You can see the six biomechanical features, and the target column is named *class*.

What column types do you have?

```
[6]: df.dtypes
```

```
[6]: pelvic_incidence            float64
     pelvic_tilt                 float64
     lumbar_lordosis_angle       float64
     sacral_slope                float64
     pelvic_radius               float64
     degree_spondylolisthesis    float64
     class                        object
     dtype: object
```

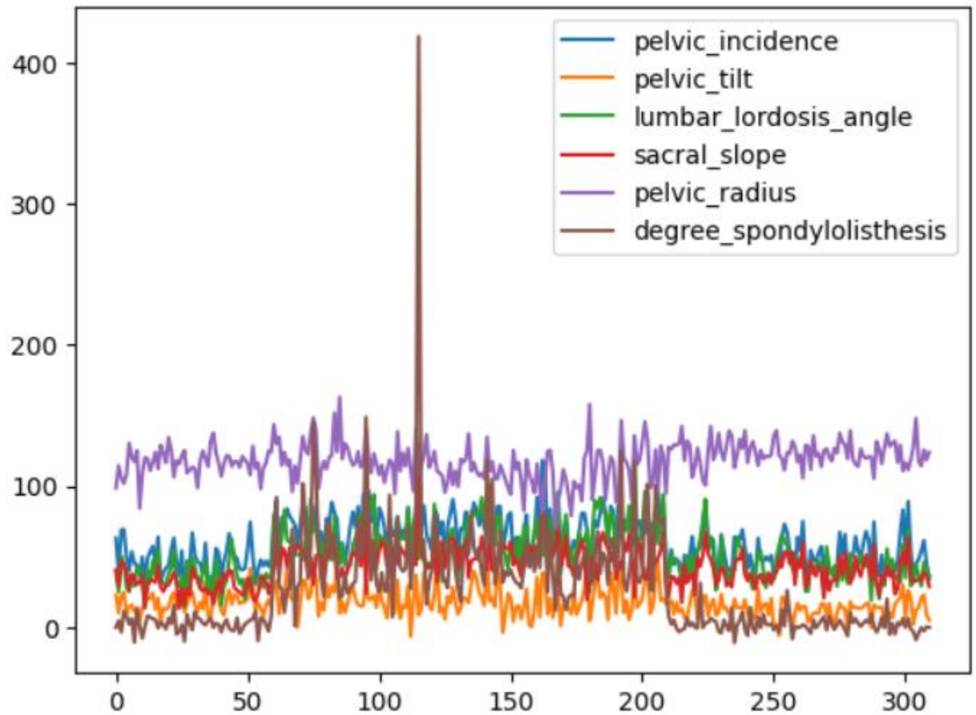
You have six floats for the biomechanical features, but the target is a class.

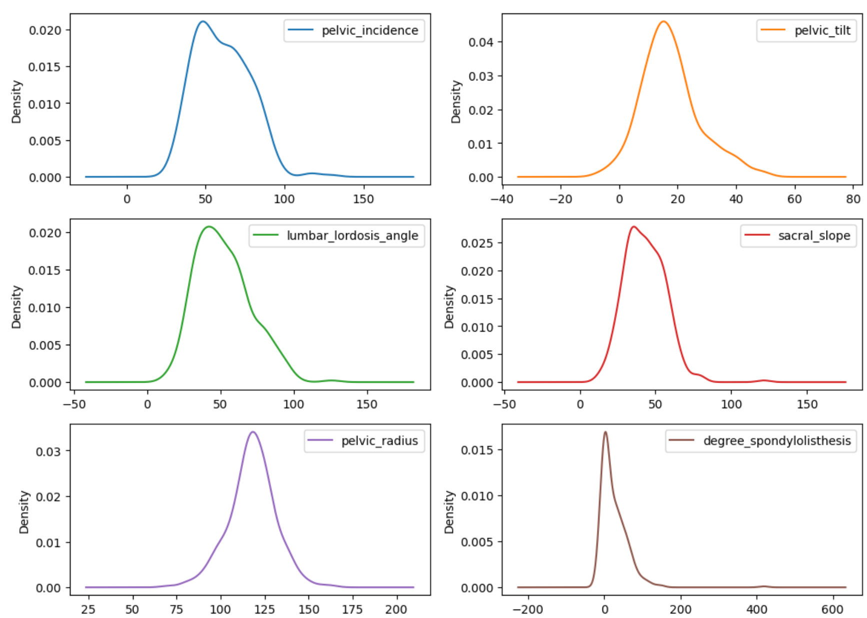
⇨ Exploring the data: the dataset has 310 rows and 7 columns (6 features + 1 class). I used df.dtypes to see types of data. It helps me a lot when the ML problem, because ML will work well with numberical variables. In this case, we can see that all of variables are numerical, except class which is the target column of the dataset.

```
df.describe()
```

	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis
count	310.000000	310.000000	310.000000	310.000000	310.000000	310.000000
mean	60.496653	17.542822	51.930930	42.953831	117.920655	26.296694
std	17.236520	10.008330	18.554064	13.423102	13.317377	37.559027
min	26.147921	-6.554948	14.000000	13.366931	70.082575	-11.058179
25%	46.430294	10.667069	37.000000	33.347122	110.709196	1.603727
50%	58.691038	16.357689	49.562398	42.404912	118.268178	11.767934
75%	72.877696	22.120395	63.000000	52.695888	125.467674	41.287352
max	129.834041	49.431864	125.742385	121.429566	163.071041	418.543082

```
df.plot(kind='density',subplots=True,layout=(4,2),figsize=(12,12),sharex=False)
plt.show()
```





```
df['degree_spondylolisthesis'].plot.density()
```

```
<Axes: ylabel='Density'>
```

A density plot smooths out the curve. It looks like there might be an increase around **400**. Visualize the data with a *histogram*.

```
df['degree_spondylolisthesis'].plot.hist()
```

```
<Axes: ylabel='Frequency'>
```

By using a *box plot*, you can see if there any outliers.

```
df['degree_spondylolisthesis'].plot.box()
```

```
<Axes: ylabel='Frequency'>
```

```
df['class'].value_counts()
```

```
class
b'Abnormal'    210
b'Normal'      100
Name: count, dtype: int64
```

It loks like you have about 1/3 *Normal* and 2/3 *Abnormal*. This result should be fine, but if you could get more data, you would want to try and balance the numbers more.

The class values aren't going to work for your ML model, so you will convert this column to a numeric value. You can use a *mapper* for this task.

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

Now, you can plot the *degree_spondylolisthesis* against the target.

```
df.plot.scatter(y='degree_spondylolisthesis',x='class')
```

```
<Axes: xlabel='class', ylabel='degree_spondylolisthesis'>
```

```
df.groupby('class').boxplot(fontsize=20,rot=90,figsize=(20,10),patch_artist=True)
```

```
0         Axes(0.1,0.15;0.363636x0.75)
1    Axes(0.536364,0.15;0.363636x0.75)
dtype: object
```

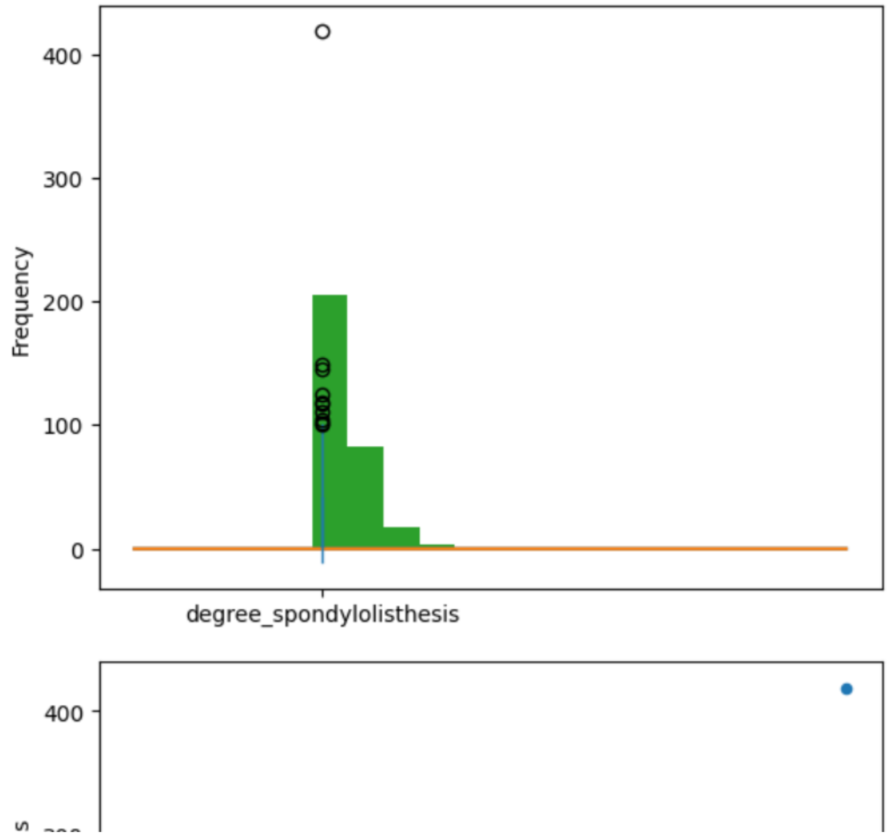
Using the **corr** function, you can create a correlation matrix for the entire dataset.

```
corr_matrix = df.corr()
corr_matrix["class"].sort_values(ascending=False)
```

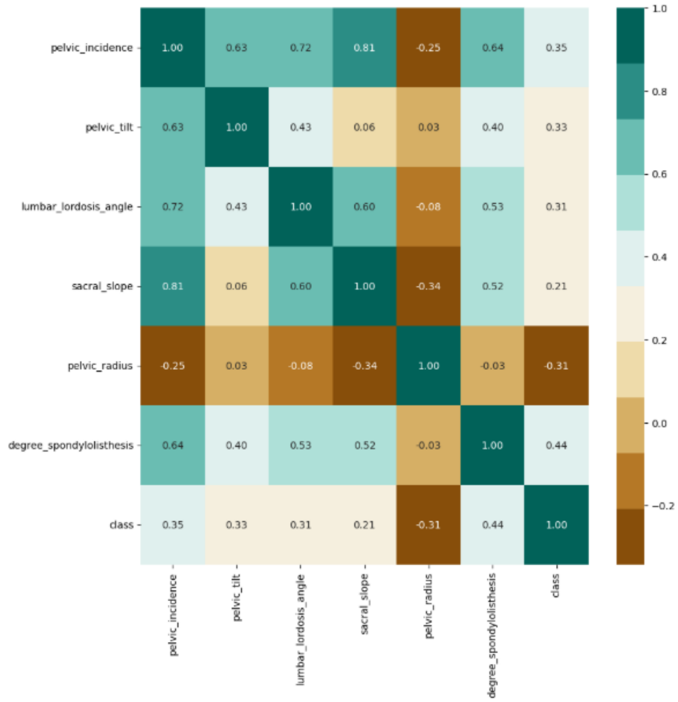
```
class                       1.000000
degree_spondylolisthesis    0.443687
pelvic_incidence            0.353336
pelvic_tilt                 0.326063
lumbar_lordosis_angle       0.312484
sacral_slope                0.210602
pelvic_radius              -0.309857
Name: class, dtype: float64
```

You can also plot out this data.

```
pd.plotting.scatter_matrix(df,figsize=(12,12))
plt.show()
```



```python
import seaborn as sns
# Plot figsize
fig, ax = plt.subplots(figsize=(10, 10))
# Generate Color Map
# colormap = sns.diverging_palette(220, 10, as_cmap=True)
colormap = sns.color_palette("BrBG", 10)
# Generate Heat Map, allow annotations and place floats in map
sns.heatmap(corr_matrix, cmap=colormap, annot=True, fmt=".2f")
#ax.set_yticklabels(column_names);
plt.show()
```



Lab 3.3 - Amazon SageMaker - Encoding Categorical Data

```python
import pandas as pd

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
```

Next, load the dataset into a pandas DataFrame.

The data doesn't contain a header, so you will define those column names in a variable that's named `col_names` to the attributes listed in the dataset description.

```python
url = "imports-85.csv"
col_names=['symboling','normalized-losses','fuel-type','aspiration','num-of-doors','body-style','drive-wheels','engine-location','wheel-base',
                              'length','width','height','curb-weight','engine-type','num-of-cylinders','engine-size',
                              'fuel-system','bore','stroke','compression-ratio','horsepower','peak-rpm','city-mpg','highway-mpg','price']

df_car = pd.read_csv(url,sep=',',names = col_names ,na_values="?",  header=None)
```

First, to see the number of rows (instances) and columns (features), you will use `shape`.

```
df_car.shape
```

```
(205, 25)
```

Next, examine the data by using the `head` method.

```
df_car.head(5)
```

	symboling	normalized-losses	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	length	width	height	curb-weight	engine-type	num-of-cylinders	engine-size	fuel-system	bore	stro
0	3	NaN	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	48.8	2548	dohc	four	130	mpfi	3.47	2.
1	3	NaN	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	48.8	2548	dohc	four	130	mpfi	3.47	2.
2	1	NaN	gas	std	two	hatchback	rwd	front	94.5	171.2	65.5	52.4	2823	ohcv	six	152	mpfi	2.68	3.
3	2	164.0	gas	std	four	sedan	fwd	front	99.8	176.6	66.2	54.3	2337	ohc	four	109	mpfi	3.19	3.
4	2	164.0	gas	std	four	sedan	4wd	front	99.4	176.6	66.4	54.3	2824	ohc	five	136	mpfi	3.19	3.

There are 25 columns. Some of the columns have numerical values, but many of them contain text.

```
df_car.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 25 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   symboling          205 non-null    int64
 1   normalized-losses  164 non-null    float64
 2   fuel-type          205 non-null    object
 3   aspiration         205 non-null    object
 4   num-of-doors       203 non-null    object
 5   body-style         205 non-null    object
 6   drive-wheels       205 non-null    object
 7   engine-location    205 non-null    object
 8   wheel-base         205 non-null    float64
 9   length             205 non-null    float64
 10  width              205 non-null    float64
 11  height             205 non-null    float64
 12  curb-weight        205 non-null    int64
 13  engine-type        205 non-null    object
 14  num-of-cylinders   205 non-null    object
 15  engine-size        205 non-null    int64
 16  fuel-system        205 non-null    object
 17  bore               201 non-null    float64
 18  stroke             201 non-null    float64
 19  compression-ratio  205 non-null    float64
 20  horsepower         203 non-null    float64
 21  peak-rpm           203 non-null    float64
 22  city-mpg           205 non-null    int64
 23  highway-mpg        205 non-null    int64
 24  price              201 non-null    float64
dtypes: float64(11), int64(5), object(9)
```

```
df_car.columns
```

```
Index(['symboling', 'normalized-losses', 'fuel-type', 'aspiration', 'num-of-doors', 'body-style', 'drive-wheels', 'engine-location', 'wheel-bas
e', 'length', 'width', 'height', 'curb-weight', 'engine-type', 'num-of-cylinders', 'engine-size', 'fuel-system', 'bore', 'stroke', 'compression
-ratio', 'horsepower', 'peak-rpm', 'city-mpg', 'highway-mpg', 'price'], dtype='object')
```

```
df_car = df_car[[ 'aspiration', 'num-of-doors',  'drive-wheels',  'num-of-cylinders']].copy()
```

You now have four columns. These columns all contain text values.

```
df_car.head()
```

	aspiration	num-of-doors	drive-wheels	num-of-cylinders
0	std	two	rwd	four
1	std	two	rwd	four
2	std	two	rwd	six
3	std	four	fwd	four
4	std	four	4wd	five

Most machine learning algorithms require inputs that are numerical values.

- The **num-of-cylinders** and **num-of-doors** features have an ordinal value. You could convert the values of these features into their numerical counterparts.
- However, **aspiration** and **drive-wheels** don't have an ordinal value. These features must be converted differently.

You will explore the ordinal features first.

Step 2: Encoding ordinal features

In this step, you will use a mapper function to convert the ordinal features into ordered numerical values.

Start by getting the new column types from the DataFrame:

```
df_car.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 4 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   aspiration        205 non-null    object
 1   num-of-doors      203 non-null    object
 2   drive-wheels      205 non-null    object
 3   num-of-cylinders  205 non-null    object
dtypes: object(4)
memory usage: 6.5+ KB
```

First, determine what values the ordinal columns contain.

Starting with the **num-of-doors** feature, you can use `value_counts` to discover the values.

```
df_car['num-of-doors'].value_counts()
```

```
four    114
two      89
Name: num-of-doors, dtype: int64
```

```
door_mapper = {"two": 2,
               "four": 4}
```

You can then use the `replace` method from pandas to generate a new numerical column based on the **num-of-doors** column.

```
df_car['doors'] = df_car["num-of-doors"].replace(door_mapper)
```

When you display the DataFrame, you should see the new column on the right. It contains a numerical representation of the number of doors.

```
df_car.head()
```

	aspiration	num-of-doors	drive-wheels	num-of-cylinders	doors
0	std	two	rwd	four	2.0
1	std	two	rwd	four	2.0
2	std	two	rwd	six	2.0
3	std	four	fwd	four	4.0
4	std	four	4wd	five	4.0

Repeat the process with the **num-of-cylinders** column.

First, get the values.

```
df_car['num-of-cylinders'].value_counts()
```

```
four      159
six        24
five       11
eight       5
two         4
three       1
twelve      1
Name: num-of-cylinders, dtype: int64
```

Next, create the mapper.

```
cylinder_mapper = {"two":2,
                   "three":3,
                   "four":4,
                   "five":5,
                   "six":6,
                   "eight":8,
                   "twelve":12}
```

Apply the mapper by using the `replace` method.

```
df_car['cylinders'] = df_car['num-of-cylinders'].replace(cylinder_mapper)
```

```
df_car.head()
```

	aspiration	num-of-doors	drive-wheels	num-of-cylinders	doors	cylinders
0	std	two	rwd	four	2.0	4
1	std	two	rwd	four	2.0	4
2	std	two	rwd	six	2.0	6
3	std	four	fwd	four	4.0	4
4	std	four	4wd	five	4.0	5

```
df_car['drive-wheels'].value_counts()
```

```
fwd    120
rwd     76
4wd      9
Name: drive-wheels, dtype: int64
```

Use the `get_dummies` method to add new binary features to the DataFrame.

```
df_car = pd.get_dummies(df_car,columns=['drive-wheels'])
```

```
df_car.head()
```

	aspiration	num-of-doors	num-of-cylinders	doors	cylinders	drive-wheels_4wd	drive-wheels_fwd	drive-wheels_rwd
0	std	two	four	2.0	4	0	0	1
1	std	two	four	2.0	4	0	0	1
2	std	two	six	2.0	6	0	0	1
3	std	four	four	4.0	4	0	1	0
4	std	four	five	4.0	5	1	0	0

```
df_car['aspiration'].value_counts()
```

```
std      168
turbo     37
Name: aspiration, dtype: int64
```

```
df_car = pd.get_dummies(df_car,columns=['aspiration'], drop_first=True)
```

```
df_car.head()
```

	num-of-doors	num-of-cylinders	doors	cylinders	drive-wheels_4wd	drive-wheels_fwd	drive-wheels_rwd	aspiration_turbo
0	two	four	2.0	4	0	0	1	0
1	two	four	2.0	4	0	0	1	0
2	two	six	2.0	6	0	0	1	0
3	four	four	4.0	4	0	1	0	0
4	four	five	4.0	5	1	0	0	0

Lab 3.4 - Amazon SageMaker - Training a model

```
import warnings, requests, zipfile, io
warnings.simplefilter('ignore')
import pandas as pd
from scipy.io import arff
import boto3
```

```
f_zip = 'http://archive.ics.uci.edu/ml/machine-learning-databases/00212/vertebral_column_data.zip'
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)
```

```
df.shape
```

```
(310, 7)
```

Next, get a list of the columns.

```
df.columns
```

```
Index(['pelvic_incidence', 'pelvic_tilt', 'lumbar_lordosis_angle',
       'sacral_slope', 'pelvic_radius', 'degree_spondylolisthesis', 'class'],
      dtype='object')
```

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

You should see that the **class** is now the first column.

```
df.columns
```

```
Index(['class', 'pelvic_incidence', 'pelvic_tilt', 'lumbar_lordosis_angle',
       'sacral_slope', 'pelvic_radius', 'degree_spondylolisthesis'],
      dtype='object')
```

```
from sklearn.model_selection import train_test_split
train, test_and_validate = train_test_split(df, test_size=0.2, random_state=42, stratify=df['class'])
```

Next, split the *test_and_validate* dataset into two equal parts.

```
test, validate = train_test_split(test_and_validate, test_size=0.5, random_state=42, stratify=test_and_validate['class'])
```

Examine the three datasets.

```
print(train.shape)
print(test.shape)
print(validate.shape)
```

```
(248, 7)
(31, 7)
(31, 7)
```

Now, check the distribution of the classes.

```
print(train['class'].value_counts())
print(test['class'].value_counts())
print(validate['class'].value_counts())
```

```
1    168
0     80
Name: class, dtype: int64
1    21
0    10
Name: class, dtype: int64
1    21
0    10
Name: class, dtype: int64
```

```
bucket='c159597a4098424l10394480t1w181649032969-labbucket-jrohftkumznv'

prefix='lab3'

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

import os

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)).put(Body=csv_buffer.getvalue())
```

Use the function that you created to upload the three datasets.

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

Step 3: Training the model

Now that the data in Amazon S3, you can train a model.

The first step is to get the XGBoost container URI.

```
import boto3
from sagemaker.image_uris import retrieve
container = retrieve('xgboost',boto3.Session().region_name,'1.0-1')
```

```
sagemaker.config INFO - Not applying SDK defaults from location: /etc/xdg/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /home/ec2-user/.config/sagemaker/config.yaml
```

Next, you must set some *hyperparameters* for the model. Because this is the first time you are training the model, you can use some values to get started.

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

Use the **estimator** function to set up the model. Here are a few parameters of interest:

- **instance_count** - This defines how many instances will be used for training. You will use *one* instance.
- **instance_type** - This defines the instance type for training. In this case, it's *ml.m4.xlarge*.

```
import sagemaker
s3_output_location="s3://{}/{}/output/".format(bucket,prefix)
xgb_model=sagemaker.estimator.Estimator(container,
                                       sagemaker.get_execution_role(),
                                       instance_count=1,
                                       instance_type='ml.m4.xlarge',
                                       output_path=s3_output_location,
                                        hyperparameters=hyperparams,
                                        sagemaker_session=sagemaker.Session())
```

```
Couldn't call 'get_role' to get Role ARN from role name c159597a4098424l10394480t1w1-SageMakerExecutionRole-hAMbPYuyDBxL to get Role path.
```

The estimator needs *channels* to feed data into the model. For training, the *train_channel* and *validate_channel* will be used.

```
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}
```

Running **fit** will train the model.

Note: This process can take up to 5 minutes.

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

```
INFO:sagemaker:Creating training-job with name: sagemaker-xgboost-2025-05-24-08-38-30-809
```

```
2025-05-24 08:38:32 Starting - Starting the training job.
2025-05-24 08:38:46 Starting - Preparing the instances for training....
2025-05-24 08:39:07 Downloading - Downloading input data.....
2025-05-24 08:39:37 Downloading - Downloading the training image........
2025-05-24 08:40:28 Training - Training image download completed. Training in progress....
2025-05-24 08:40:49 Uploading - Uploading generated training model.
2025-05-24 08:41:02 Completed - Training job completed
```

After the training is complete, you are ready to test and evaluate the model. However, you will do testing and validation in later labs.

Lab 3.5 - Amazon SageMaker - Deploying a model

```
bucket='c159597a4098426l10407094t1w905685017570-labbucket-ovhovpgnst2h'
```

```
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
```

```
sagemaker.config INFO - Not applying SDK defaults from location: /etc/xdg/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /home/ec2-user/.config/sagemaker/config.yaml
```

```
f_zip = 'http://archive.ics.uci.edu/ml/machine-learning-databases/00212/vertebral_column_data.zip'
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, stratify=df['class'])
test, validate = train_test_split(test_and_validate, test_size=0.5, random_state=42, stratify=test_and_validate['class'])

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)).put(Body=csv_buffer.getvalue())

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"}

s3_output_location="s3://{}/{}/output/".format(bucket,prefix)
xgb_model=sagemaker.estimator.Estimator(container,
                                        sagemaker.get_execution_role(),
                                        instance_count=1,
                                        instance_type='ml.m4.xlarge',
                                        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)

print('ready for hosting!')
```

```
INFO:sagemaker:Creating training-job with name: sagemaker-xgboost-2025-05-25-10-03-42-474

2025-05-25 10:03:43 Starting - Starting the training job..
2025-05-25 10:03:58 Starting - Preparing the instances for training....
2025-05-25 10:04:25 Downloading - Downloading input data.....
2025-05-25 10:04:55 Downloading - Downloading the training image........
2025-05-25 10:05:41 Training - Training image download completed. Training in progress....
2025-05-25 10:06:02 Uploading - Uploading generated training model..
2025-05-25 10:06:15 Completed - Training job completed
ready for hosting!
```

Step 1: Hosting the model

Now that you have a trained model, you can host it by using Amazon SageMaker hosting services.

The first step is to deploy the model. Because you have a model object, *xgb_model*, you can use the **deploy** method. For this lab, you will use a single ml.m4.xlarge instance.

```
]: xgb_predictor = xgb_model.deploy(initial_instance_count=1,
                 serializer = sagemaker.serializers.CSVSerializer(),
                 instance_type='ml.m4.xlarge')
```

```
INFO:sagemaker:Creating model with name: sagemaker-xgboost-2025-05-25-10-06-31-036
INFO:sagemaker:Creating endpoint-config with name sagemaker-xgboost-2025-05-25-10-06-31-036
INFO:sagemaker:Creating endpoint with name sagemaker-xgboost-2025-05-25-10-06-31-036
-----------!
```

Step 2: Performing predictions

Now that you have a deployed model, you will run some predictions.

First, review the test data and re-familiarize yourself with it.

```
[6]: test.shape
```

```
[6]: (31, 7)
```

You have 31 instances, with seven attributes. The first five instances are:

```
[7]: test.head(5)
```

[7]:

	class	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis
136	1	88.024499	39.844669	81.774473	48.179830	116.601538	56.766083
230	0	65.611802	23.137919	62.582179	42.473883	124.128001	-4.083298
134	1	52.204693	17.212673	78.094969	34.992020	136.972517	54.939134
130	1	50.066786	9.120340	32.168463	40.946446	99.712453	26.766697
47	1	41.352504	16.577364	30.706191	24.775141	113.266675	-4.497958

```
row = test.iloc[0:1,1:]
row.head()
```

	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis
136	88.024499	39.844669	81.774473	48.17983	116.601538	56.766083

You can convert this to a comma-separated values (CSV) file, and store it in a string buffer.

```
batch_X_csv_buffer = io.StringIO()
row.to_csv(batch_X_csv_buffer, header=False, index=False)
test_row = batch_X_csv_buffer.getvalue()
print(test_row)
```

```
88.0244989,39.84466878,81.77447308,48.17983012,116.6015376,56.76608323
```

Now, you can use the data to perform a prediction.

```
xgb_predictor.predict(test_row)
```

```
b'0.9966071844100952'
```

```
test.head(5)
```

	class	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis
136	1	88.024499	39.844669	81.774473	48.179830	116.601538	56.766083
230	0	65.611802	23.137919	62.582179	42.473883	124.128001	-4.083298
134	1	52.204693	17.212673	78.094969	34.992020	136.972517	54.939134
130	1	50.066786	9.120340	32.168463	40.946446	99.712453	26.766697
47	1	41.352504	16.577364	30.706191	24.775141	113.266675	-4.497958

Step 3: Terminating the deployed model

To delete the endpoint, use the **delete_endpoint** function on the predictor.

```
xgb_predictor.delete_endpoint(delete_endpoint_config=True)
```

```
INFO:sagemaker:Deleting endpoint configuration with name: sagemaker-xgboost-2025-05-25-10-06-31-036
INFO:sagemaker:Deleting endpoint with name: sagemaker-xgboost-2025-05-25-10-06-31-036
```

```
batch_X = test.iloc[:,1:];
batch_X.head()
```

	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis
136	88.024499	39.844669	81.774473	48.179830	116.601538	56.766083
230	65.611802	23.137919	62.582179	42.473883	124.128001	-4.083298
134	52.204693	17.212673	78.094969	34.992020	136.972517	54.939134
130	50.066786	9.120340	32.168463	40.946446	99.712453	26.766697
47	41.352504	16.577364	30.706191	24.775141	113.266675	-4.497958

Next, write your data to a CSV file.

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

Last, before you perform a transform, configure your transformer with the input file, output location, and instance type.


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

Last, before you perform a transform, configure your transformer with the input file, output location, and instance type.


```
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.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()
```

```
INFO:sagemaker:Creating model with name: sagemaker-xgboost-2025-05-25-10-12-48-316
INFO:sagemaker:Creating transform job with name: sagemaker-xgboost-2025-05-25-10-12-48-999
.............................
```

```
s3 = boto3.client('s3')
obj = s3.get_object(Bucket=bucket, Key="{}/batch-out/{}".format(prefix,'batch-in.csv.out'))
target_predicted = pd.read_csv(io.BytesIO(obj['Body'].read()),sep=',',names=['class'])
target_predicted.head(5)
```

You can use a function to convert the probabilty into either a *0* or a *1*.

The first table output will be the *predicted values*, and the second table output is the *original test data*.

```
def binary_convert(x):
    threshold = 0.65
    if x > threshold:
        return 1
    else:
        return 0

target_predicted['binary'] = target_predicted['class'].apply(binary_convert)

print(target_predicted.head(10))
test.head(10)
```

Note: The *threshold* in the **binary_convert** function is set to *.65*.

Lab 3.6 - Amazon SageMaker - Generating model performance metrics

```
bucket='c159597a4098428l10407590t1w536722728253-labbucket-jlpr6oqbmwcq'
```

```
import warnings, requests, zipfile, io
warnings.simplefilter('ignore')
import pandas as pd
from scipy.io import arff

import os
import boto3
import sagemaker
import numpy as np
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
from sagemaker.image_uris import retrieve
from sklearn.model_selection import train_test_split
```

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

```
f_zip = 'http://archive.ics.uci.edu/ml/machine-learning-databases/00212/vertebral_column_data.zip'
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, stratify=df['class'])
test, validate = train_test_split(test_and_validate, test_size=0.5, random_state=42, stratify=test_and_validate['class'])

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)).put(Body=csv_buffer.getvalue())

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"}

s3_output_location="s3://{}/{}/output/".format(bucket,prefix)
xgb_model=sagemaker.estimator.Estimator(container,
                                        sagemaker.get_execution_role(),
                                        instance_count=1,
                                        instance_type='ml.m4.xlarge',
                                        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.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()

s3 = boto3.client('s3')
obj = s3.get_object(Bucket=bucket, Key="{}/batch-out/{}".format(prefix,'batch-in.csv.out'))
```

```
INFO:sagemaker:Creating training-job with name: sagemaker-xgboost-2025-05-25-18-38-26-438
```

```
def binary_convert(x):
    threshold = 0.3
    if x > threshold:
        return 1
    else:
        return 0

target_predicted_binary = target_predicted['class'].apply(binary_convert)

print(target_predicted_binary.head(5))
test.head(5)
```

```
0    1
1    1
2    1
3    1
4    1
Name: class, dtype: int64
```

	class	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis
136	1	88.024499	39.844669	81.774473	48.179830	116.601538	56.766083
230	0	65.611802	23.137919	62.582179	42.473883	124.128001	-4.083298
134	1	52.204693	17.212673	78.094969	34.992020	136.972517	54.939134
130	1	50.066786	9.120340	32.168463	40.946446	99.712453	26.766697
47	1	41.352504	16.577364	30.706191	24.775141	113.266675	-4.497958

```
test_labels = test.iloc[:,0]
test_labels.head()
```

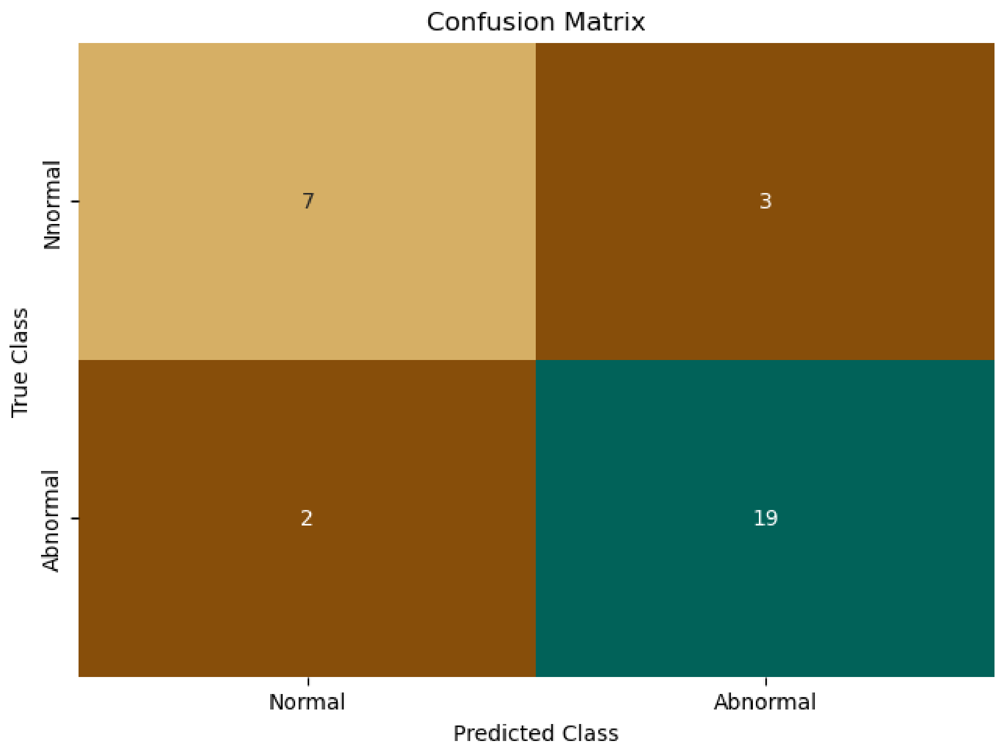
```
136    1
230    0
134    1
130    1
47     1
Name: class, dtype: int64
```

Now, you can use the *scikit-learn* library, which contains a function to create a confusion matrix.

```
from sklearn.metrics import confusion_matrix

matrix = confusion_matrix(test_labels, target_predicted_binary)
df_confusion = pd.DataFrame(matrix, index=['Nnormal','Abnormal'],columns=['Normal','Abnormal'])

df_confusion
```

	Normal	Abnormal
Nnormal	7	3
Abnormal	2	19

```
import seaborn as sns
import matplotlib.pyplot as plt

colormap = sns.color_palette("BrBG", 10)
sns.heatmap(df_confusion, annot=True, cbar=None, cmap=colormap)
plt.title("Confusion Matrix")
plt.tight_layout()
plt.ylabel("True Class")
plt.xlabel("Predicted Class")
plt.show()
```



```
from sklearn.metrics import roc_auc_score, roc_curve, auc

TN, FP, FN, TP = confusion_matrix(test_labels, target_predicted_binary).ravel()

print(f"True Negative (TN) : {TN}")
print(f"False Positive (FP): {FP}")
print(f"False Negative (FN): {FN}")
print(f"True Positive (TP) : {TP}")
```

```
True Negative (TN) : 7
False Positive (FP): 3
False Negative (FN): 2
True Positive (TP) : 19
```

You can now calculate some statistics.

Sensitivity

Sensitivity is also known as *hit rate, recall,* or *true positive rate (TPR)*. It measures the proportion of the actual positives that are correctly identified.

In this example, the sensitivity is *the probablity of detecting an abnormality for patients with an abnormality.*

```
# Sensitivity, hit rate, recall, or true positive rate
Sensitivity  = float(TP)/(TP+FN)*100
print(f"Sensitivity or TPR: {Sensitivity}%")
print(f"There is a {Sensitivity}% chance of detecting patients with an abnormality have an abnormality")
```

```
Sensitivity or TPR: 90.47619047619048%
There is a 90.47619047619048% chance of detecting patients with an abnormality have an abnormality
```

Specificity

The next statistic is *specificity*, which is also known as the *true negative*. It measures the proportion of the actual negatives that are correctly identified.

In this example, the specificity is *the probablity of detecting normal, for patients who are normal.*

```
# Specificity or true negative rate
Specificity  = float(TN)/(TN+FP)*100
print(f"Specificity or TNR: {Specificity}%")
print(f"There is a {Specificity}% chance of detecting normal patients are normal.")
```

```
Specificity or TNR: 70.0%
There is a 70.0% chance of detecting normal patients are normal.
```

Positive and negative predictive values

The *precision*, or *positive predictive value*, is the proportion of positive results.

In this example, the positive predictive value is *the probability that subjects with a positive screening test truly have an abnormality.*

```
# Precision or positive predictive value
Precision = float(TP)/(TP+FP)*100
print(f"Precision: {Precision}%")
print(f"You have an abnormality, and the probablity that is correct is {Precision}%")
```

```
Precision: 86.36363636363636%
You have an abnormality, and the probablity that is correct is 86.36363636363636%
```

The *negative predictive value* is the proportion of negative results.

In this example, the negative predictive value is *the probability that subjects with a negative screening test truly have an abnormality.*

```
# Negative predictive value
NPV = float(TN)/(TN+FN)*100
print(f"Negative Predictive Value: {NPV}%")
print(f"You don't have an abnormality, but there is a {NPV}% chance that is incorrect" )
```

```
Negative Predictive Value: 77.77777777777779%
You don't have an abnormality, but there is a 77.77777777777779% chance that is incorrect
```

False positive rate

The *false positive rate (FPR)* is the probability that a false alarm will be raised, or that *a positive result will be given when the true value is negative.*

```
# Fall out or false positive rate
FPR = float(FP)/(FP+TN)*100
print( f"False Positive Rate: {FPR}%")
print( f"There is a {FPR}% chance that this positive result is incorrect.")
```

```
False Positive Rate: 30.0%
There is a 30.0% chance that this positive result is incorrect.
```

False negative rate

The *false negative rate* -- or *miss rate* -- is *the probability that a true positive will be missed by the test.*

```
# False negative rate
FNR = float(FN)/(TP+FN)*100
print(f"False Negative Rate: {FNR}%")
print(f"There is a {FNR}% chance that this negative result is incorrect.")
```

```
False Negative Rate: 9.523809523809524%
There is a 9.523809523809524% chance that this negative result is incorrect.
```

False discovery rate

In this example, the *false discovery rate* is *the probability of predicting an abnormality when the patient doesn't have one.*

```
# False discovery rate
FDR = float(FP)/(TP+FP)*100
print(f"False Discovery Rate: {FDR}%" )
print(f"You have an abnormality, but there is a {FDR}% chance this is incorrect.")
```

```
False Discovery Rate: 13.636363636363635%
You have an abnormality, but there is a 13.636363636363635% chance this is incorrect.
```

Overall accuracy

How accuracte is your model?

```
# Overall accuracy
ACC = float(TP+TN)/(TP+FP+FN+TN)*100
print(f"Accuracy: {ACC}%")
```

```
Accuracy: 83.87096774193549%
```

In summary, you calculated the following metrics from your model:

```
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}%")
```

```
Sensitivity or TPR: 90.47619047619048%
Specificity or TNR: 70.0%
Precision: 86.36363636363636%
Negative Predictive Value: 77.77777777777779%
False Positive Rate: 30.0%
False Negative Rate: 9.523809523809524%
False Discovery Rate: 13.636363636363635%
Accuracy: 83.87096774193549%
```

Step 4: Calculating the AUC-ROC Curve

The scikit-learn library has functions that can help you compute the *area under the receiver operating characteristic curve (AUC-ROC)*.

- The ROC is a probability curve.
- The AUC tells you how well the model can distinguish between classes.

The AUC can be calculated. As you will see in the next lab, it can be used to measure the performance of the model.

In this example, the higher the AUC, the better the model is at distinguishing between abnormal and normal patients.

Depending on the value you set for the threshold, the AUC can change. You can plot the AUC by using the probability instead of your converted class.

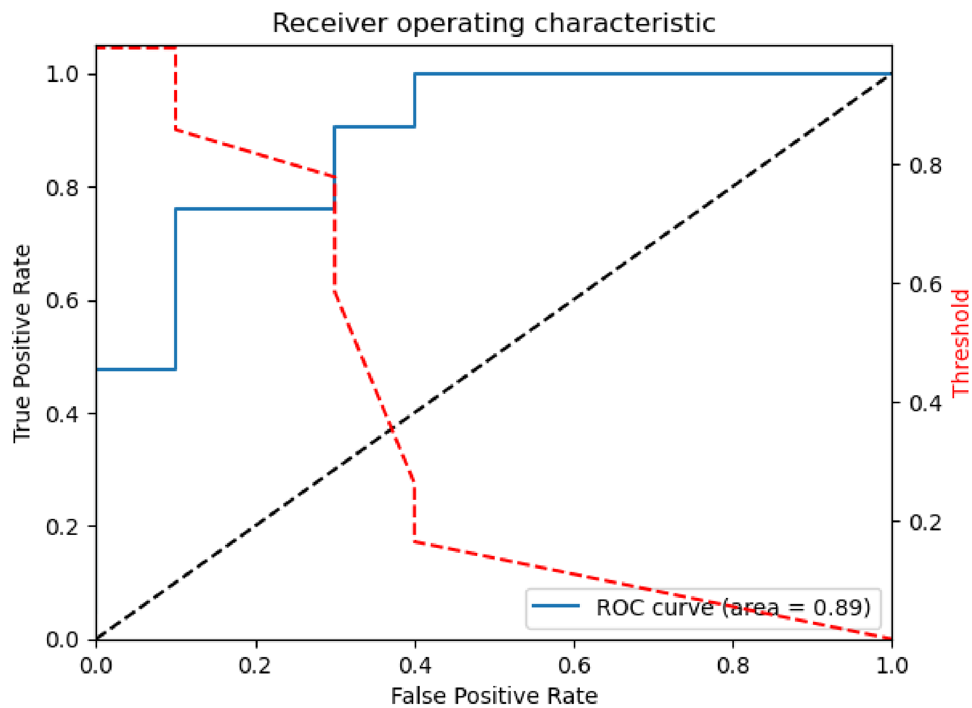
```
test_labels = test.iloc[:,0];
print("Validation AUC", roc_auc_score(test_labels, target_predicted) )
```

```
Validation AUC 0.8904761904761904
```

Typically, the ROC curve is plotted with the TPR against the FPR, where the TPR is on the y-axis and the FPR is on the x-axis.

scikit-learn has the **roc_curve** function to help generate those values to plot.

```
fpr, tpr, thresholds = roc_curve(test_labels, target_predicted)

finite_indices = np.isfinite(thresholds)
fpr_finite = fpr[finite_indices]
tpr_finite = tpr[finite_indices]
thresholds_finite = thresholds[finite_indices]

plt.figure()
plt.plot(fpr_finite, tpr_finite, label='ROC curve (area = %0.2f)' % auc(fpr_finite, tpr_finite))
plt.plot([0, 1], [0, 1], 'k--')  # Dashed diagonal
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")

roc_auc = auc(fpr, tpr)

if thresholds_finite.size > 0:
    ax2 = plt.gca().twinx()
    ax2.plot(fpr_finite, thresholds_finite, markeredgecolor='r', linestyle='dashed', color='r')
    ax2.set_ylabel('Threshold', color='r')
    ax2.set_ylim([thresholds_finite[-1], thresholds_finite[0]])
    ax2.set_xlim([fpr_finite[0], fpr_finite[-1]])

plt.show()
```



+

Lab 3.7 - Amazon SageMaker - Hyperparameter Tuning

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.

[7]:
```
bucket='c159597a4098430l10453832t1w936967379244-labbucket-no4qeaof3lg7'
```

[8]:
```
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
```

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:

```
INFO:sagemaker:Creating model with name: sagemaker-xgboost-2025-05-29-13-43-10-941

INFO:sagemaker:Creating transform job with name: sagemaker-xgboost-2025-05-29-13-43-11-605
...........................
..!
CPU times: user 1.27 s, sys: 82.6 ms, total: 1.35 s
Wall time: 7min 10s
```

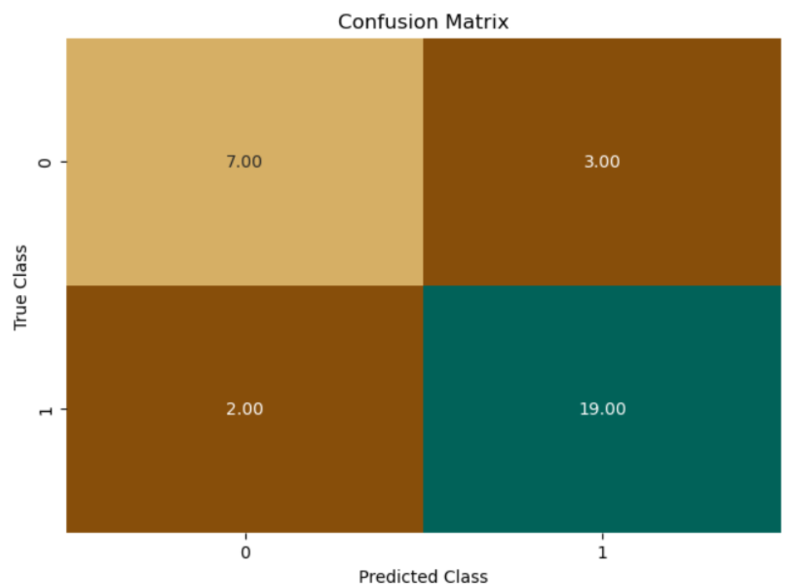
Step 1: Getting model statistics

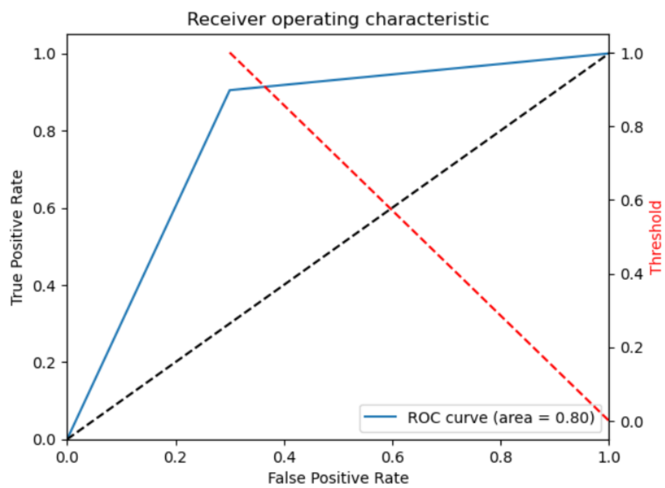
Before you tune the model, re-familiarize yourself with the current model's metrics.

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

[10]:
```python
s3 = boto3.client('s3')
obj = s3.get_object(Bucket=bucket, Key="{}/batch-out/{}".format(prefix,'batch-in.csv.out'))
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]
```

[13]:
```python
plot_confusion_matrix(test_labels, target_predicted_binary)
```





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.

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.

```
[18]: boto3.client('sagemaker').describe_hyper_parameter_tuning_job(
          HyperParameterTuningJobName=tuner.latest_tuning_job.job_name)['HyperParameterTuningJobStatus']
```

```
[18]: '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.

19]:
```
from pprint import pprint
from sagemaker.analytics import HyperparameterTuningJobAnalytics

tuner_analytics = HyperparameterTuningJobAnalytics(tuner.latest_tuning_job.name, sagemaker_session=sagemaker.Session())

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)
```

19]:

	alpha	eta	min_child_weight	num_round	subsample	TrainingJobName	TrainingJobStatus	FinalObjectiveValue	TrainingStartTime	TrainingEndTime	TrainingEla
1	9.947633	0.246740	1.526445	50.0	0.924905	sagemaker-xgboost-250529-1350-009-d40d0842	Completed	0.09677	2025-05-29 14:03:49+00:00	2025-05-29 14:04:23+00:00	
2	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	sagemaker-xgboost-250529-	Completed	0.09677	2025-05-29	2025-05-29	

[28]:
```
attached_tuner = HyperparameterTuner.attach(tuner.latest_tuning_job.name, sagemaker_session=sagemaker.Session())
best_training_job = attached_tuner.best_training_job()
```

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

[29]:
```
from sagemaker.estimator import Estimator
algo_estimator = Estimator.attach(best_training_job)

best_algo_model = algo_estimator.create_model(env={'SAGEMAKER_DEFAULT_INVOCATIONS_ACCEPT':"text/csv"})
```

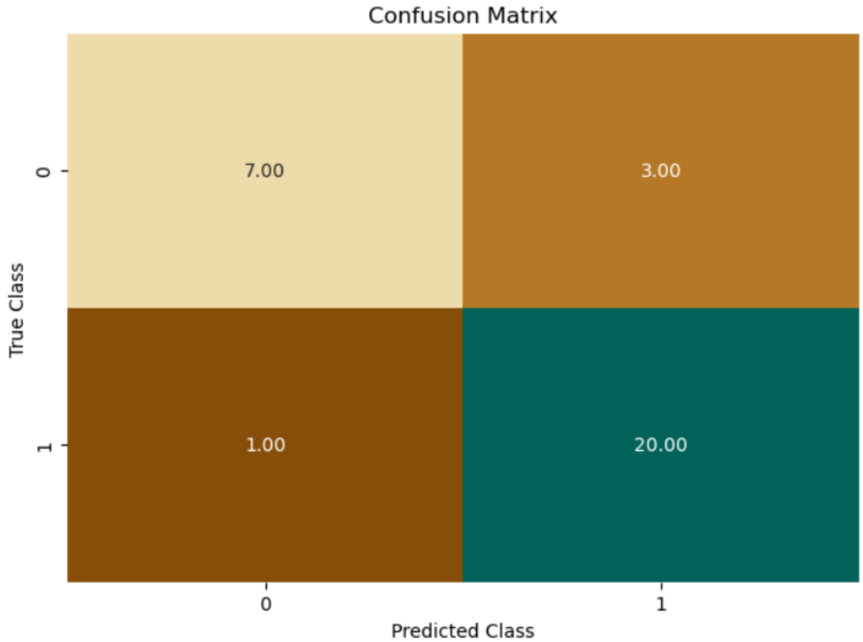
```
2025-05-29 13:56:41 Starting - Found matching resource for reuse
2025-05-29 13:56:41 Downloading - Downloading the training image
2025-05-29 13:56:41 Training - Training image download completed. Training in progress.
2025-05-29 13:56:41 Uploading - Uploading generated training model
2025-05-29 13:56:41 Completed - Resource reused by training job: sagemaker-xgboost-250529-1350-004-cb95fb6c
```

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.

[*]:
```
%%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',
```

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

[35]:
```
plot_confusion_matrix(test_labels, best_target_predicted_binary)
```

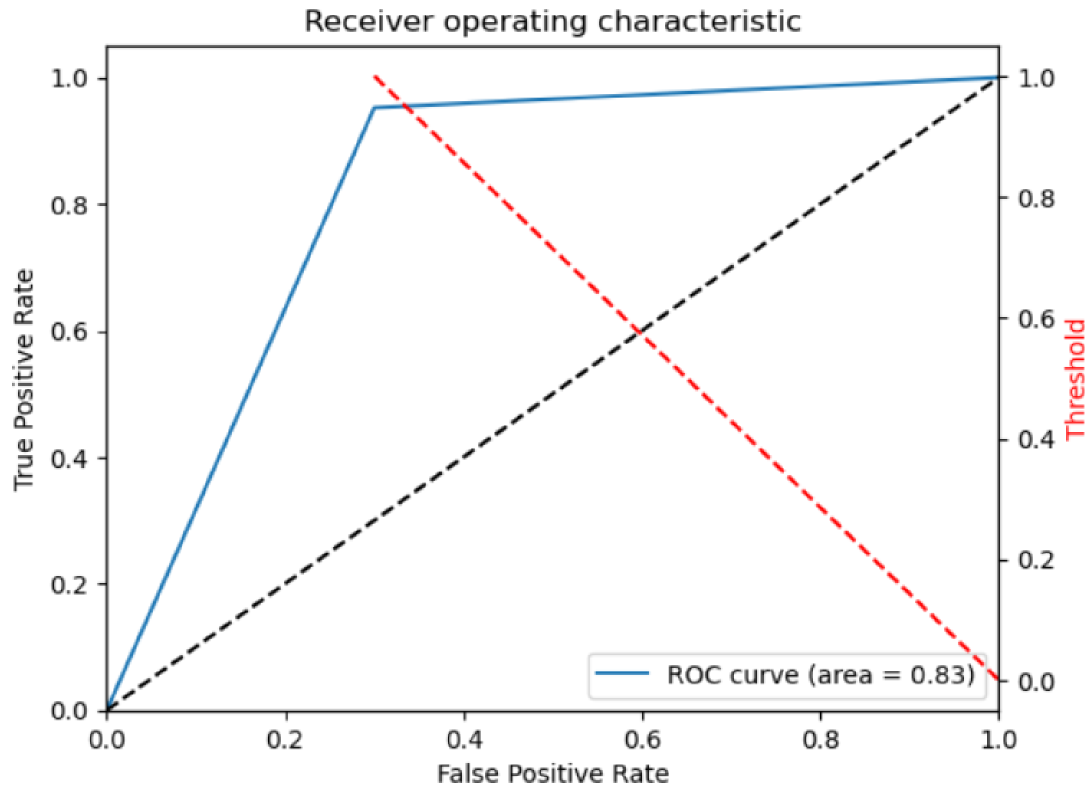


Plot the ROC chart.

[36]:
```
plot_roc(test_labels, best_target_predicted_binary)
```

```
Sensitivity or TPR: 95.23809523809523%
Specificity or TNR: 70.0%
Precision: 86.95652173913044%
Negative Predictive Value: 87.5%
False Positive Rate: 30.0%
False Negative Rate: 4.761904761904762%
False Discovery Rate: 13.043478260869565%
Accuracy: 87.09677419354838%
Validation AUC 0.8261904761904761
```

```
Traceback (most recent call last)
in <module>:1
```



Deliverables

- A detailed report that includes sections corresponding to the steps outlined above.
- Code files containing the data analysis, model code, and any additional scripts used.

Assessment

Your work will be evaluated based on the provided rubric (RUBRIC No. 3)

<u>**End**</u>