

In this Lab, we will explore Amazon SageMaker as a powerful cloud native tool for training and deploying machine learning models into production.

- The official SageMaker SDK documentation: <https://sagemaker.readthedocs.io>

## 1. Data processing using SageMaker

First, we will familiarize ourselves with the dataset we will use to run our SageMaker Jobs.

### Problem description:

Find the best strategies to improve for the next marketing campaign. How can the financial institution (a bank) have greater effectiveness for future marketing campaigns? To answer it, we need to analyze the latest marketing campaign carried out by the bank and identify patterns that will help us draw conclusions in order to develop future strategies.

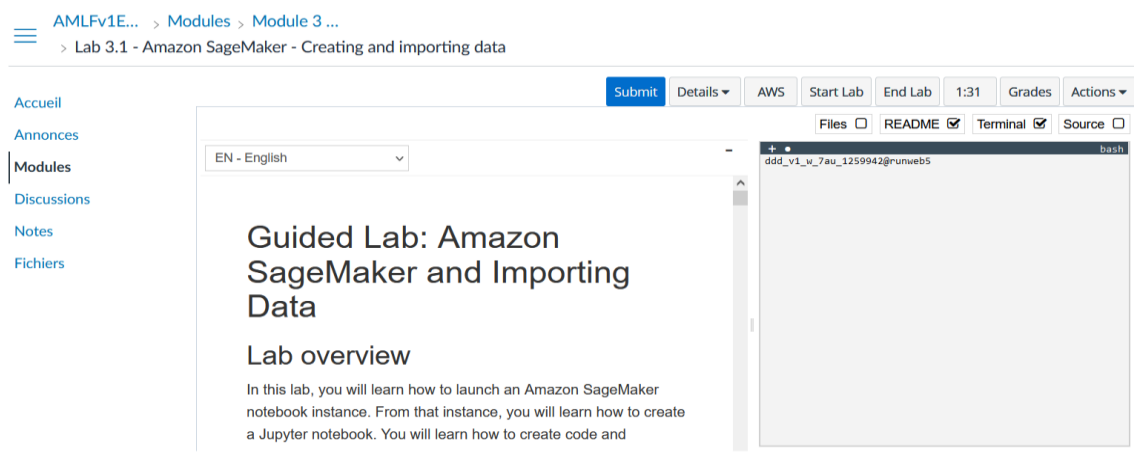
### Source :

[Moro et al., 2014] S. Moro, P. Cortez and P. Rita. A Data-Driven Approach to Predict the Success of Bank Telemarketing. Decision Support Systems, Elsevier, 62:22-31, June 2014]

In the lab folder, you will find a notebook called “Bank-Marketing.ipynb” in which I have prepared a set of processing to carry out in order to familiarize you with the dataset.

Run it in your local environment and look at the output of each entry. Then access your Amazon Academy space and more specifically the “AWS Academy Machine Learning Foundations” course

Then click on “Lab 3.1 - Amazon SageMaker - Creating and importing data” then on “Start Lab”.

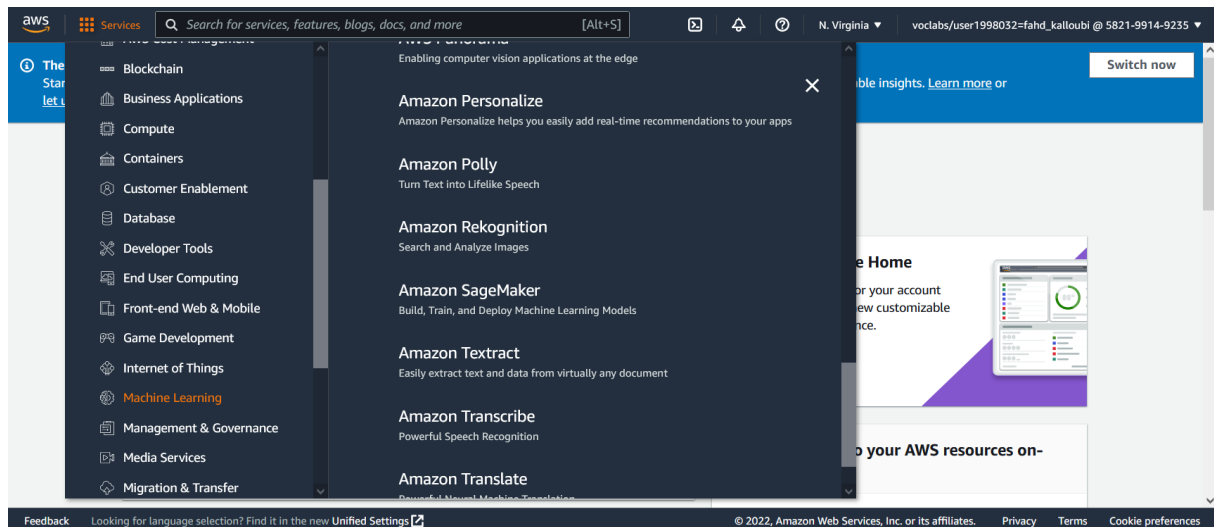


The screenshot shows the AWS Academy Lab interface. On the left, there is a sidebar with navigation links: Accueil, Annonces, Modules (selected), Discussions, Notes, and Fichiers. The main content area displays the title 'Guided Lab: Amazon SageMaker and Importing Data' and a 'Lab overview' section. The overview text states: 'In this lab, you will learn how to launch an Amazon SageMaker notebook instance. From that instance, you will learn how to create a Jupyter notebook. You will learn how to create code and'. On the right side of the interface, there is a terminal window with the prompt 'ddd\_v1\_w\_7au\_125942@runweb5' and a 'bash' prompt. Above the terminal, there are tabs for 'Files', 'README' (checked), 'Terminal' (checked), and 'Source'. At the top of the interface, there is a navigation bar with a breadcrumb trail: 'AMLFv1E... > Modules > Module 3 ... > Lab 3.1 - Amazon SageMaker - Creating and importing data'. The navigation bar also includes buttons for 'Submit', 'Details', 'AWS', 'Start Lab', 'End Lab', '1:31', 'Grades', and 'Actions'.

A window will then appear indicating the status of your lab, if the status is “ready” then close it and click on AWS.

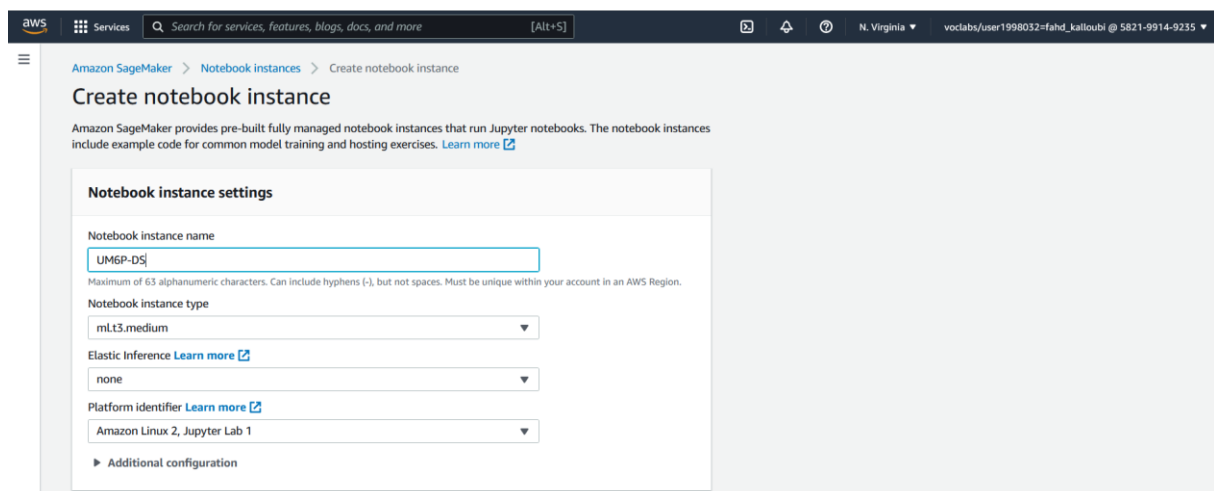
You will then be redirected to the Amazon web services console.

While on the console, click on “services” and then “Machine Learning” and finally on “Amazon SageMaker” as shown in the figure below.



Once on the home page, click on “Notebook >> Notebook instances” in the menu on the left and then on the “create notebook instance” button.

Give your instance a name as shown in the figure below:



When the status of the instance changes to “InService”, click “Open JupyterLab”.

In the JupyterLab integrated environment, create a notebook called “Bank-Marketing-processing.ipynb”.

1. We will start by downloading and extracting our data

```
%%sh
wget -N https://sagemaker-sample-data-us-west-2.s3-us-west-2.amazonaws.com/autopilot/direct_marketing/bank-additional.zip
```

```
unzip -o bank-additional.zip
```

2. We will then load it with Pandas

```
import pandas as pd
data = pd.read_csv('./bank-additional/bank-additionalfull.csv')
print(data.shape)
data[:5]
```

3. Now let's upload the dataset to Amazon S3. We will use a default “Bucket” automatically created by SageMaker in the region we are running in.

```
prefix = 'sagemaker/DEMO-smprocessing/input'

input_data = sagemaker.Session().upload_data(path='./bank-additional/bank-
additional-full.csv', key_prefix=prefix)
```

4. Since SageMaker Processing takes care of all infrastructure issues, we can focus on the script itself. We also don't have to worry about Amazon S3: SageMaker Processing will automatically copy the input dataset from S3 to the container, and the processed datasets from the container to S3. Container paths are provided when we configure the Job itself. Here is what we will use:

- The input data set: /opt/ml/processing/input
- The processed training set: /opt/ml/processing/train
- The processed test set: /opt/ml/processing/test

In the lab folder, you will find the processing script named “preprocessing.py”. This script will load the dataset, perform some preprocessing and save the processed dataset

5. Processing script execution: Coming back to our notebook, we use the SKLearnProcessor object from the SageMaker SDK to configure the processing task:

- We define which version of scikit-learn we want to use and what our infrastructure requirements are. Here we opt for an ml.m5.xlarge instance:

```
from sagemaker.sklearn.processing import SKLearnProcessor
role = sagemaker.get_execution_role()
sklearn_processor = SKLearnProcessor(framework_version='0.20.0',
                                     role=role,
                                     instance_type='ml.m5.xlarge',
                                     instance_count=1)
```

- Then we simply run the Job, passing the script name, the dataset input path in S3, the user-defined dataset paths into the SageMaker processing environment.

```
from sagemaker.processing import ProcessingInput, ProcessingOutput

sklearn_processor.run(code='preprocessing.py',
```

```

inputs=[ProcessingInput(
    source=input_data,
    destination='/opt/ml/processing/input'),
outputs=[ProcessingOutput(output_name='train_data',
    source='/opt/ml/processing/train'),
    ProcessingOutput(output_name='test_data',
    source='/opt/ml/processing/test')]
)

```

6. After a few minutes, the job ends and we can see the script outputs:

```

Reading input data from /opt/ml/processing/input/bank-additional-
full.csv
Positive samples: 4639
Negative samples: 36537
Ratio: 7.88
Running preprocessing and feature engineering transformations
Train data shape after preprocessing: (32940, 58)
Test data shape after preprocessing: (8236, 58)
Saving training features to
/opt/ml/processing/train/train_features.csv
Saving test features to /opt/ml/processing/test/test_features.csv
Saving training labels to /opt/ml/processing/train/train_labels.csv
Saving test labels to /opt/ml/processing/test/test_labels.csv

```

7. Finally, we can display the job to display the location of the processed datasets

```

preprocessing_job_description = sklearn_processor.jobs[-1].describe()
output_config =
preprocessing_job_description['ProcessingOutputConfig']
for output in output_config['Outputs']:
print(output['S3Output']['S3Uri'])

```

8. In a terminal, we can use the AWS CLI to retrieve the processed training and test sets located at the previous path. To do this, go to the “Launcher” tab and click on “Terminal” and then run the following two commands to copy the files from S3 to the local machine as well as two other commands to view the first records:

```

aws s3 cp s3://sagemaker-us-east-1-582199149235/sagemaker-scikit-learn-2022-06-25-11-23-07-573/output/train_data/train_features.csv .
aws s3 cp s3://sagemaker-us-east-1-582199149235/sagemaker-scikit-learn-2022-06-25-11-23-07-573/output/train_data/train_labels.csv .

head -1 train_features.csv
head -1 train_labels.csv

```

```

sh-4.2$ aws s3 cp s3://sagemaker-us-east-1-582199149235/sagemaker-scikit-learn-2022-06-25-11-23-07-573/output/train_data .
fatal error: An error occurred (404) when calling the HeadObject operation: Key "sagemaker-scikit-learn-2022-06-25-11-23-07-573/output/train_data" does not exist
sh-4.2$ aws s3 cp s3://sagemaker-us-east-1-582199149235/sagemaker-scikit-learn-2022-06-25-11-23-07-573/output/train_data/train_features.csv .
download: s3://sagemaker-us-east-1-582199149235/sagemaker-scikit-learn-2022-06-25-11-23-07-573/output/train_data/train_features.csv to ./train_features.csv
sh-4.2$ aws s3 cp s3://sagemaker-us-east-1-582199149235/sagemaker-scikit-learn-2022-06-25-11-23-07-573/output/train_data/train_labels.csv .
download: s3://sagemaker-us-east-1-582199149235/sagemaker-scikit-learn-2022-06-25-11-23-07-573/output/train_data/train_labels.csv to ./train_labels.csv
sh-4.2$ ls
anaconda3  LICENSE                               Nvidia_Cloud_EULA.pdf  SageMaker              sample-notebooks-1656155130  tools              train_labels.csv
examples   nvidia-acknowledgements  README                sample-notebooks      src                          train_features.csv  tutorials
sh-4.2$

```

## 2. Training a model using SageMaker SDK with built-in algorithms

After exploring how you can preprocess a dataset using Amazon SageMaker. In this part, we will train a regression model on the Boston Housing dataset (<https://www.kaggle.com/code/prasadperera/the-boston-housing-dataset>).

### 2.1 Data preparation

The algorithms built into Amazon SageMaker expect the dataset to be in a certain format, such as CSV, protobuf, or libsvm. Supported formats are listed in the algorithm documentation. In the workshop folder you will find the “housing.csv” dataset and the “Linear Learner on Boston Housing.ipynb” notebook, import them into your environment. In the following, we will explain each cell separately.

#### 1. Data loading :

```
import pandas as pd

dataset = pd.read_csv('housing.csv')
print(dataset.shape)
dataset[:5]
```

#### 2. Reading the official SageMaker documentation

(<https://docs.aws.amazon.com/sagemaker/latest/dg/cdf-training.html>), we can notice that Amazon SageMaker requires that the CSV file should not contain the header record and that the target variable must be at the beginning.

```
dataset = pd.concat([dataset['medv'], dataset.drop(['medv'], axis=1)],
axis=1)
```

#### 3. Let's divide our dataset into 90% training and 10% testing

```
from sklearn.model_selection import train_test_split
training_dataset, validation_dataset = train_test_split(dataset,
test_size=0.1)

print(training_dataset.shape)
print(validation_dataset.shape)
```

#### 4. We will save these two partitions in two csv files without headers and without indices. Notice that both files are created in the workspace on the left.

```
training_dataset.to_csv('training_dataset.csv', index=False, header=False)
validation_dataset.to_csv('validation dataset.csv', index=False, header=False)
```

#### 5. Now we need to upload these two files to S3 in two separate folders. We will use the default “Bucket” bucket created by SageMaker in the region where we are running our notebook. We can find out the name by running “sagemaker.Session.default\_bucket()”

```
import sagemaker

print(sagemaker. version )
```

```

sess = sagemaker.Session()
bucket = sess.default_bucket()

prefix = 'boston-housing'
training_data_path = sess.upload_data(path='training_dataset.csv',
key_prefix=prefix + '/input/training')
validation_data_path = sess.upload_data(path='validation_dataset.csv',
key_prefix=prefix + '/input/validation')

print(training_data_path)
print(validation_data_path)

```

- Both Buckets will look like this (the account number will be different in your case)

```

s3://sagemaker-eu-west-1-123456789012/boston-housing/
input/training/training_dataset.csv

s3://sagemaker-eu-west-1-123456789012/boston-housing/
input/validation/validation_dataset.csv

```

## 2.2 Configure the training job

The Estimator object (`sagemaker.estimator.Estimator`) is the cornerstone of training a model in SageMaker. It allows you to select the appropriate algorithm, define your training infrastructure needs, etc.

- SageMaker algorithms are packaged in Docker containers. Using `boto3` and the `image_uris.retrieve()` API, we can easily find the name of the Linear Learner algorithm in the region in which we are running our job:

```

import boto3
from sagemaker import image_uris

region = boto3.Session().region_name
container = image_uris.retrieve('linear-learner', region)
print(container)

```

- Now that we know the container name, we can configure our training Job with the Estimator object:

```

from sagemaker.estimator import Estimator

role = sagemaker.get_execution_role()

ll_estimator = Estimator(container,
    role=role,
    instance_count=1,
    instance_type='ml.m5.large',
    output_path='s3://{}/{}/output'.format(bucket, prefix)
)

```

- Next, we need to define the hyper parameters: in order to tune the parameters we need to read the algorithm documentation and respect the mandatory parameters only, then we need to quickly check the optional parameters for default values that might conflict with your dataset. Let's look at the documentation and see which hyperparameters are required ([https://docs.aws.amazon.com/sagemaker/latest/dg/ll\\_hyperparameters.html](https://docs.aws.amazon.com/sagemaker/latest/dg/ll_hyperparameters.html)). It turns out that there is only one: `predictor_type` which defines the type of problem

that Linear Learner trains on (regression, binary classification or multiclass classification).

Also we can see that the default value for `mini_batch_size` is 1000: this won't work well with our dataset of 506 samples, so let's set it to 32. We can also see that the `normalize_data` parameter is set to true by default, which makes it unnecessary to normalize the data ourselves

```
ll_estimator.set_hyperparameters(  
    predictor_type='regressor',  
    mini_batch_size=32)
```

4. Now let's define data channels: A channel is a named source of data passed to a SageMaker estimator. All built-in algorithms need at least one training channel, and many also support additional channels for validation and testing. We will define two channels: one for training and the other for validation

```
training_data_channel =  
sagemaker.TrainingInput(s3_data=training_data_path,content_type='text/csv')  
validation_data_channel =  
sagemaker.TrainingInput(s3_data=validation_data_path,content_type='text/csv')  
  
ll_data = {'train': training_data_channel, 'validation':  
validation_data_channel}
```

## 2.3 Running the training job

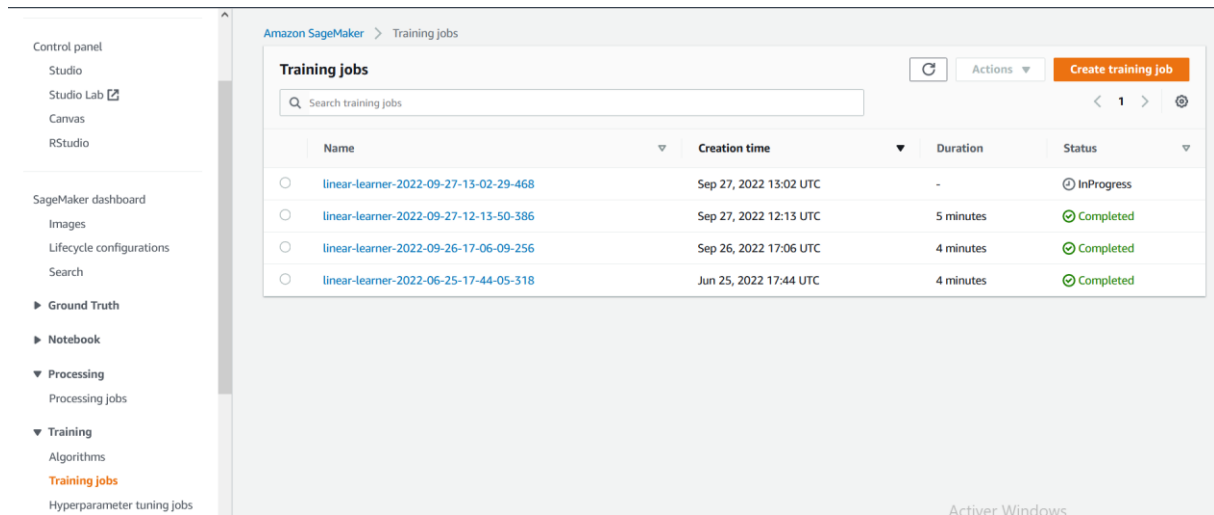
1. We will pass the data channel dictionary to the `fit()` method.

```
ll_estimator.fit(ll_data)
```

The training logs will be visible in the notebook. The first lines show that the infrastructure is provisioned

```
2022-09-27 13:02:29 Starting - Starting the training job...  
2022-09-27 13:02:45 Starting - Preparing the instances for  
trainingProfilerReport-1664283749: InProgress  
.....  
2022-09-27 13:03:55 Downloading - Downloading input data.....  
2022-09-27 13:04:51 Training - Downloading the training image.....  
2022-09-27 13:06:28 Training - Training image download completed.  
Training in progress..Docker entrypoint called with argument(s): train
```

Once the job is executed, it appears in the SageMaker console in the “Training jobs” section as shown in the figure below:



- Looking at the output location in our S3 bucket, we see the model artifact:

```
%bash -s "$ll_estimator.output_path"
aws s3 ls --recursive $1
```

## 2.4 Deploy our model

- It is recommended to create identifiable and unique endpoint names. However, we could also let SageMaker create one for us during deployment:

```
from time import strftime, gmtime
timestamp = strftime('%d-%H-%M-%S', gmtime())

endpoint_name = 'linear-learner-demo-'+timestamp
print(endpoint_name)
```

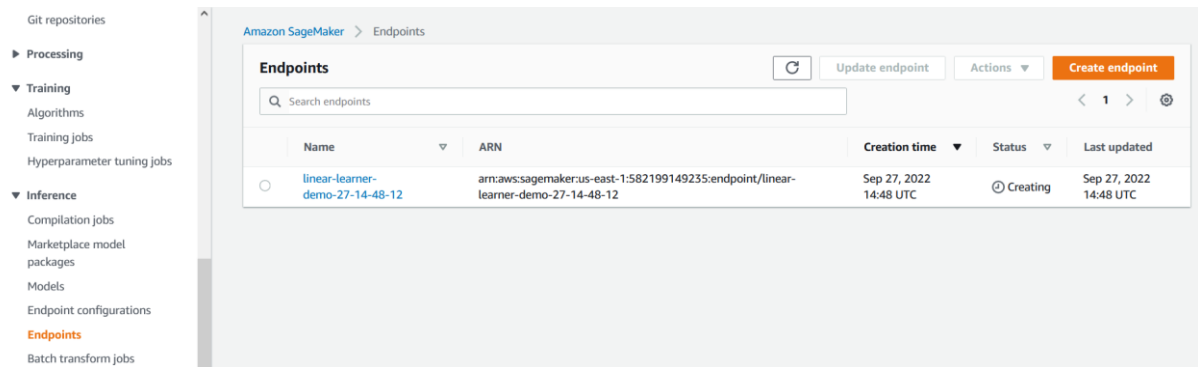
So the endpoint will be named: linear-learner-demo-27-14-48-12

- We deploy the model using the `deploy()` API. As this is a test endpoint, we use the smallest instance available, `ml.t2.medium`. In the eu-west-1 region, it will only cost us \$0.07 per hour:

```
ll_predictor = ll_estimator.deploy(endpoint_name=endpoint_name,
                                   initial_instance_count=1,
                                   instance_type='ml.t2.medium')
```

Once the endpoint is created, you can consult it in the “inferere > endpoint” section as shown in the figure below:





3. A few minutes later, the terminal is in service. We can use the `predict()` API to send it a CSV sample for prediction. We set the content type and serialization accordingly: built-in functions are available and we use them as is:

```
test_sample =
'0.00632,18.00,2.310,0,0.5380,6.5750,65.20,4.0900,1,296.0,15.30,4.98'

#ll_predictor.content_type = 'text/csv'
ll_predictor.serializer = sagemaker.serializers.CSVSerializer()
ll_predictor.deserializer = sagemaker.deserializers.CSVDeserializer()

response = ll_predictor.predict(test_sample)
print(response)
```

We can also predict the output of several samples:

```
test_samples =
['0.00632,18.00,2.310,0,0.5380,6.5750,65.20,4.0900,1,296.0,15.30,4.98',
'0.02731,0.00,7.070,0,0.4690,6.4210,78.90,4.9671,2,242.0,17.80,9.14']

response = ll_predictor.predict(test_samples)
print(response)
print(ll_predictor.endpoint_name)
```

We can also predict using the endpoint directly using the “`invoke_endpoint()`” function of the “runtime” object.

```
runtime = boto3.Session().client(service_name='runtime.sagemaker')

response = runtime.invoke_endpoint(EndpointName=endpoint_name,
                                   ContentType='text/csv',
                                   Body=test_sample)

print(response['Body'].read())
```

Finally, and to avoid any additional charges, we can delete the endpoint:

```
ll_predictor.delete_endpoint()
```

### **To do :**

1. Train XGboost model by tuning its hyperparameters. Try to create a new notebook for this task.