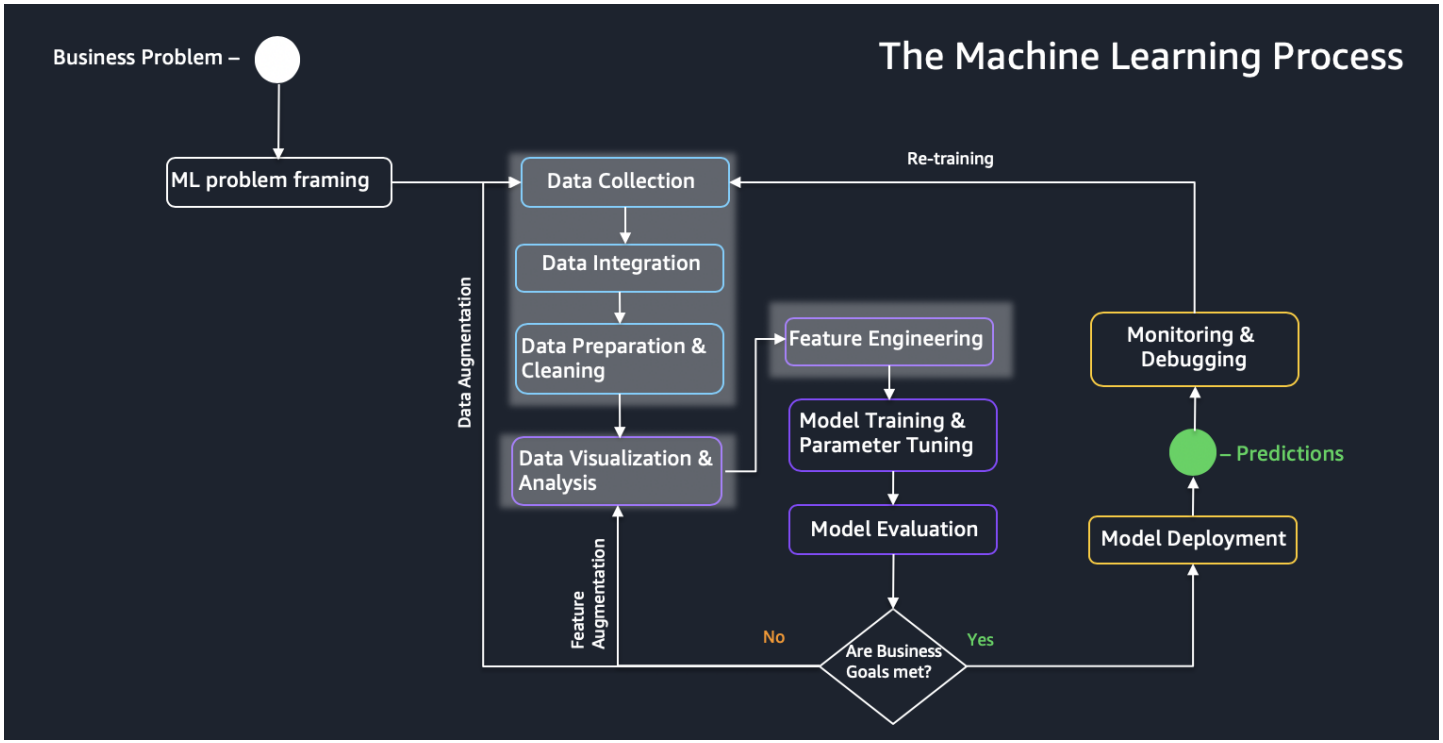
Option 3: Amazon SageMaker Processing

⚠ DO NOT perform this part if you have already executed the feature engineering with Amazon Data Wrangler (Option 1) or with Numpy and Pandas (Option 2).

Introduction

For the data scientists and ML engineers who are more experienced with running preprocessing, postprocessing and model evaluation workloads with Python scripts or custom containers, Amazon SageMaker Processing introduces a new Python SDK that lets you do exactly this. Processing jobs accept data from Amazon S3 as input and store data into Amazon S3 as output.



To use SageMaker Processing, simply supply a Python data preprocessing script. For this example, we're using a SageMaker prebuilt Scikit-learn container, which includes many common functions and libraries for processing data. There are few limitations on what kinds of code and operations you can run, and only a minimal contract: input and output data must be placed in specified directories:

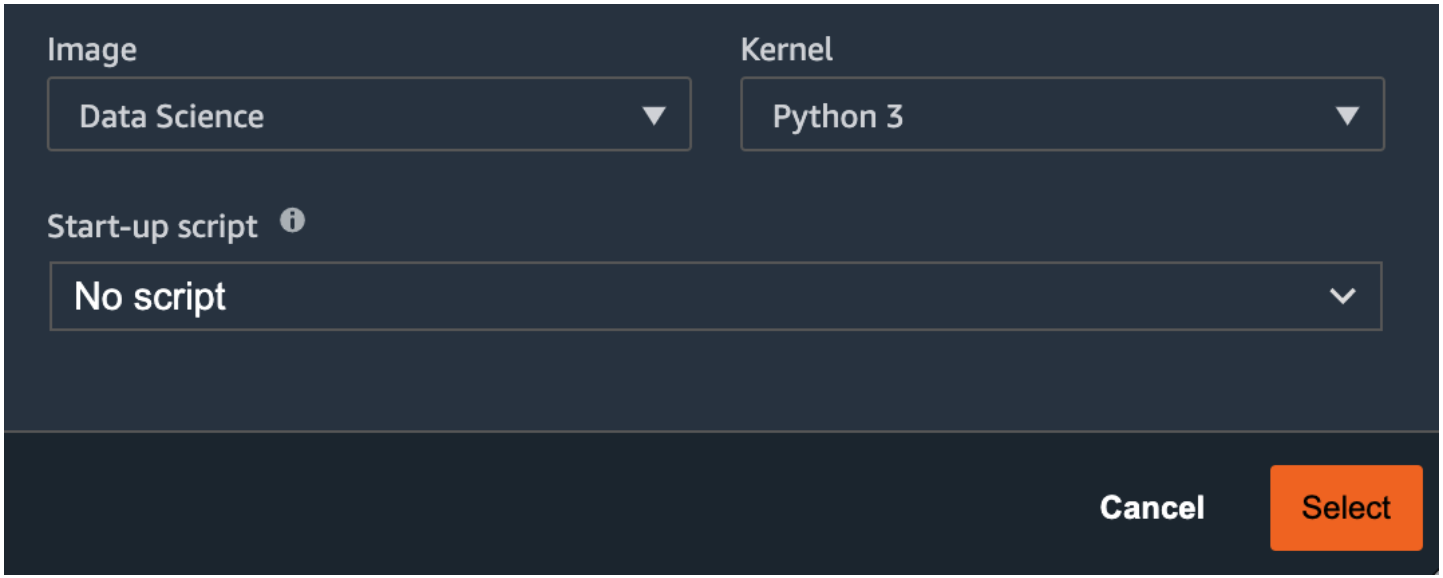
- **/opt/ml/processing/input/** for input data
- **/opt/ml/processing/output/** for output data

If this is done, SageMaker Processing automatically loads the input data from S3 and uploads transformed data back to S3 when the job is complete.

In this lab, we'll import the dataset and transform it with SageMaker Processing, which can be used to process terabytes of data in a SageMaker-managed cluster separate from the instance running your notebook server. In a typical SageMaker workflow, notebooks are only used for prototyping and can be run on relatively inexpensive and less powerful instances, while processing, training and model hosting tasks are run on separate, more powerful SageMaker-managed instances.

Getting Started

1. Click on this **amazon-sagemaker-immersion-day** folder and then double click on the **processing_xgboost.ipynb** notebook.
2. If you are prompted to choose a Kernel, choose the “Python 3 (Data Science)” kernel and click “Select”.



3. Run the different cells until the one that starts with **%%writefile preprocessing.py**. This cell will write locally the script that will be used as source by SageMaker Processing.
4. A few things are noteworthy from this script:
   1. the **ArgumentParser** makes it easier to be able to run this script anywhere, whether it's in the AWS cloud or locally on your machine. Nothing in this script is specific to SageMaker, except from the default values of **--filepath** and **--outputpath**
   2. by setting the default values of a few of the arguments in the **ArgumentParser**, we ensure that the paths are by default compliant with SageMaker expected paths
   3. If you want to use dynamic values, you will be able to provide them when creating the **Processor**

```
parser = argparse.ArgumentParser()
# Data, model, and output directories
# model_dir is always passed in from SageMaker. By default this is a S3 path under the default bucket.
parser.add_argument('--filepath', type=str, default='/opt/ml/processing/input/')
parser.add_argument('--filename', type=str, default='bank-additional-full.csv')
parser.add_argument('--outputpath', type=str, default='/opt/ml/processing/output/')
```

5. Before starting the SageMaker Processing job, we instantiate a SKLearnProcessor object. This object allows you to specify the instance type to use in the job, as well as how many instances. If you wanted to use a PySpark cluster, you can use the PySparkProcessor , or even a custom ScriptProcessor if you want to provide a script to your own custom container.

```
from sagemaker.sklearn.processing import SKLearnProcessor

sklearn_processor = SKLearnProcessor(
    framework_version="0.23-1",
```

```
    role=get_execution_role(),
    instance_type="ml.m5.large",
    instance_count=1,
    base_job_name='sm-immday-skprocessing'
)
```

6. You can now run this processing job, by specifying the code to use, inputs, outputs, and the arguments, if any. By specifying s3_data_distribution_type="ShardedByS3Key", we assure that, if we are using multiple instances for processing, data is distributed across them, to ensure parallelization of the processing job.

```
sklearn_processor.run(
    code='preprocessing.py',
    # arguments = ['arg1', 'arg2'],
    inputs=[
        ProcessingInput(
            source=input_source,
            destination="/opt/ml/processing/input",
            s3_input_mode="File",
            s3_data_distribution_type="ShardedByS3Key"
        )
    ],
    outputs=[
        ProcessingOutput(
            output_name="train_data",
            source="/opt/ml/processing/output/train",
            destination=train_path,
        ),
        ProcessingOutput(output_name="validation_data", source="/opt/ml/processing/output/validation", destination=validation_path
        ProcessingOutput(output_name="test_data", source="/opt/ml/processing/output/test", destination=test_path),
    ]
)
```

7. This cell can take a few minutes to run (from 5 to 7). Once it's done, you can check that the Processing job has generated its outputs in the train_path S3 location by running:

```
!aws s3 ls $train_path/
2021-06-10 17:39:22    3545009 train.csv
```

8. Now your data is ready! You can proceed with Lab 2 to train your XGBoost model.