Lab 2. Train, Tune and Deploy XGBoost

⚠ Make sure you have performed **Lab 1** (Option 1, 2 **or** 3) before beginning this lab.

Overview

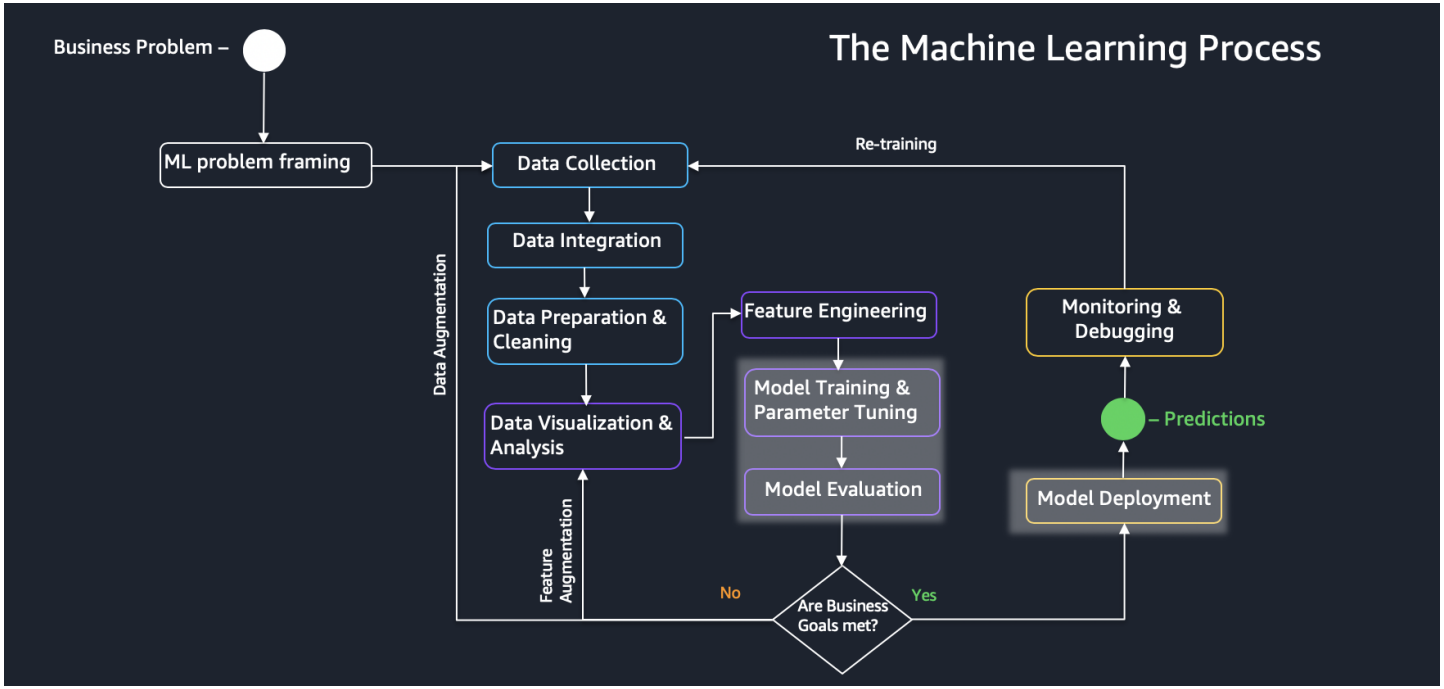
In this lab, you will learn how to use Amazon SageMaker to build, train, and deploy a machine learning (ML) model. We will use the popular XGBoost ML algorithm for this exercise. Amazon SageMaker is a modular, fully managed machine learning service that enables developers and data scientists to build, train, and deploy ML models at scale.

Taking ML models from concept to production is typically complex and time-consuming. You have to manage large amounts of data to train the model, choose the best algorithm for training it, manage the compute capacity while training it, and then deploy the model into a production environment. Amazon SageMaker reduces this complexity by making it much easier to build and deploy ML models. After you choose the right algorithms and frameworks from the wide range of choices available, SageMaker manages all of the underlying infrastructure to train your model at petabyte scale, and deploy it to production.

In this tutorial, you will assume the role of a machine learning developer working at a bank. You have been asked to develop a machine learning model to predict whether a customer will enroll for a certificate of deposit (CD). The model will be trained on the marketing dataset that contains information on customer demographics, responses to marketing events, and external factors.

The data has been labeled for your convenience and a column in the dataset identifies whether the customer is enrolled for a product offered by the bank. A version of this dataset is publicly available from the ML repository curated by the University of California, Irvine. This tutorial implements a supervised machine learning model, since the data is labeled. (Unsupervised learning occurs when the datasets are not labeled.)

In this tutorial, you will learn about the highlighted sections:



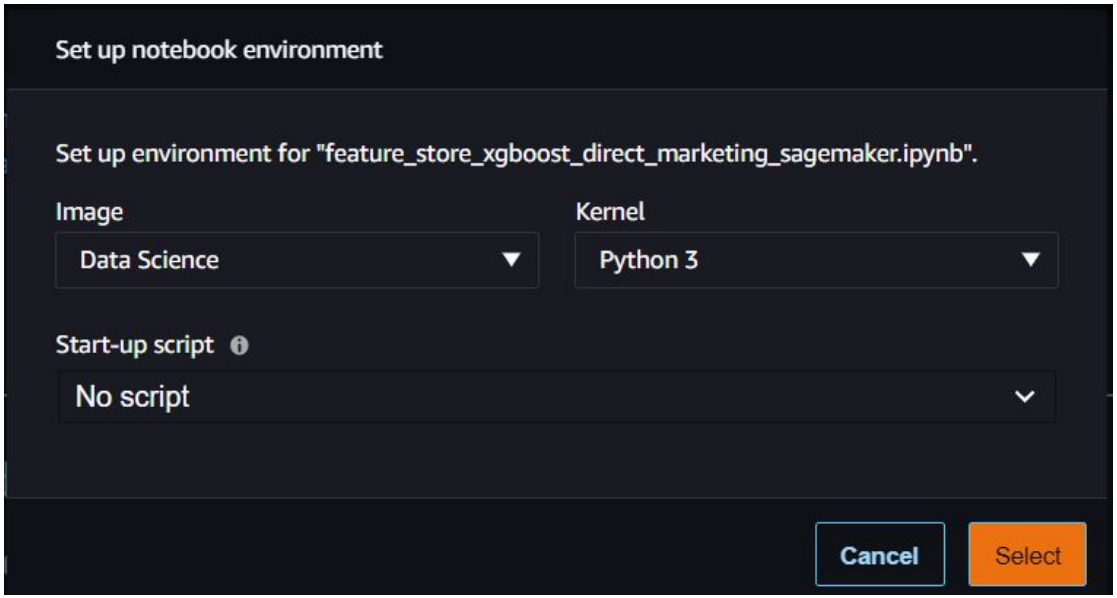
- Train & Tune the model
- Deploy the model
- Evaluate your ML model's performance
- Automatic model tuning

Export Data to S3 *(Optional)*

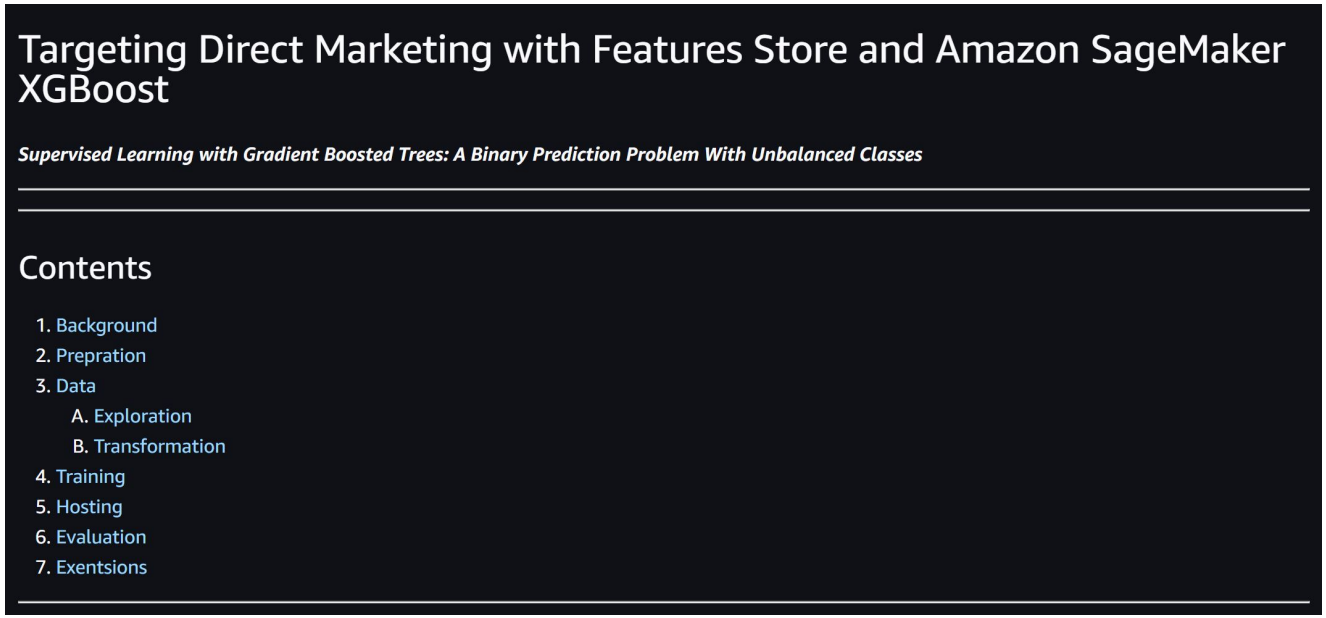
Important
Only run this section if you completed **Lab1 Option1 (Amazon SageMaker Data Wrangler and Feature Store)**, else skip to the **Train & Tune the model** section below

You will retreive the transformed data stored in Feature Store from the Lab1 Option1 section and put it into Amazon s3 to train our model. Let's do this by following the steps below:

1. In your SageMaker Studio Notebook, go to the folders on the top left pane, click on **amazon-sagemaker-immersion-day** folder and then double click on the last file: **feature_store_xgboost_direct_marketing_sagemaker.ipynb** notebook.
2. If you are prompted to choose a Kernel, select the “Python 3 (Data Science)” kernel and click “Select”. If not, verify that the right kernel has been automatically selected.



3. You will then have the notebook opened. You can verify the Kernel CPU and Memory states on the top right of the notebook.



4. Execute the first two cells by pressing **Shift+Enter** in each of the cells. While code runs in the cell, an * appears between the square brackets as pictured in the first screenshot to the right. After a few seconds, the code execution will complete, the * will be replaced with a number.

This code will import some libraries and define a few environment variables in your Jupyter notebook environment.

```
# cell 01
import sagemaker
bucket=sagemaker.Session().default_bucket()
prefix = 'sagemaker/DEMO-xgboost-dm'

# Define IAM role
import boto3
import re
from sagemaker import get_execution_role

role = get_execution_role()
```

Now let's bring in the Python libraries that we'll use throughout the analysis

```
# cell 02
import numpy as np                                # For matrix operations and numerical processing
import pandas as pd                               # For munging tabular data
import matplotlib.pyplot as plt                   # For charts and visualizations
from IPython.display import Image                 # For displaying images in the notebook
from IPython.display import display               # For displaying outputs in the notebook
from time import gmtime, strftime                 # For labeling SageMaker models, endpoints, etc.
import sys                                        # For writing outputs to notebook
import math                                       # For ceiling function
import json                                       # For parsing hosting outputs
import os                                         # For manipulating filepath names
import sagemaker
import zipfile     # Amazon SageMaker's Python SDK provides many helper functions
```

5. In order to be able to do the training in the next section below, we need to retrieve the transformed dataset (including the right features) from Amazon Feature Store and put it into a Pandas dataframe.

> ⚠ Replace the **YOUR FEATURE GROUP NAME** placeholder by your feature group name before executing the cell.

Data

Let's start by downloading the direct marketing dataset from the sample data s3 bucket.

[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

Transformation

The transformations steps were made in DataWrangler & Features Store

Get the data from FS

```
# cell 03
from sagemaker.session import Session
from sagemaker.feature_store.feature_group import FeatureGroup

region = boto3.Session().region_name
boto_session = boto3.Session(region_name=region)

sagemaker_client = boto_session.client(service_name='sagemaker', region_name=region)
featurestore_runtime = boto_session.client(service_name='sagemaker-featurestore-runtime', region_name=region)

feature_store_session = Session(
    boto_session=boto_session,
    sagemaker_client=sagemaker_client,
    sagemaker_featurestore_runtime_client=featurestore_runtime
)

feature_group_name = "YOUR FEATURE GROUP NAME"
feature_group = FeatureGroup(name=feature_group_name, sagemaker_session=feature_store_session)
```

Once your feature group name has been placed in the cell, you can execute it as well as the following cells. You will be able to retrieve the dataset from **Amazon Feature Store** as a **Pandas dataframe**.

```
# cell 03
from sagemaker.session import Session
from sagemaker.feature_store.feature_group import FeatureGroup

region = boto3.Session().region_name
boto_session = boto3.Session(region_name=region)

sagemaker_client = boto_session.client(service_name='sagemaker', region_name=region)
featurestore_runtime = boto_session.client(service_name='sagemaker-featurestore-runtime', region_name=region)

feature_store_session = Session(
    boto_session=boto_session,
    sagemaker_client=sagemaker_client,
    sagemaker_featurestore_runtime_client=featurestore_runtime
)

feature_group_name = "FG-ImmersionDay-8as334e4"
feature_group = FeatureGroup(name=feature_group_name, sagemaker_session=feature_store_session)
```

```
# cell 04
# Build SQL query to features group
fs_query = feature_group.athena_query()
fs_table = fs_query.table_name
query_string = 'SELECT * FROM "'+fs_table+'"'
print('Running ' + query_string)
```

```
Running SELECT * FROM "immersionday1-1679785381"
```

```
# cell 05
# Run Athena query. The output is loaded to a Pandas dataframe.
fs_query.run(query_string=query_string, output_location='s3://'+bucket+'/'+prefix+'/fs_query_results/')
fs_query.wait()
model_data = fs_query.as_dataframe()
```

```
# cell 06
model_data = model_data.drop(['fs_id', 'fs_time', 'write_time', 'api_invocation_time', 'is_deleted'], axis=1)
```

```
# cell 07
model_data
```

	age	campaign	pdays	previous	no_previous_contact	not_working	contact_cellular	contact_telephone	day_of_week_thu	day_of_week_wed	...	month_nov	month_oct	month_mar	month_sep	month_dec	poutcome_nonexistent	poutcome_su
0	0.583291	0.122006	999.000000	0.000000	1	0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	1.0	
1	0.634757	0.000000	999.000000	0.000000	1	0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	1.0	
2	0.428890	0.122006	999.000000	0.000000	1	1	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	1.0	
3	0.686224	0.122006	999.000000	0.000000	1	0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	1.0	
4	0.171556	0.122006	999.000000	0.000000	1	0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	1.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
69372	0.236387	0.098246	3.805254	0.437144	0	0	1.0	0.0	0.0	0.0	...	0.0	1.0	0.0	0.0	0.0	0.0	
69373	0.171079	0.122006	999.000000	0.000000	1	0	1.0	0.0	1.0	0.0		1.0	0.0	0.0	0.0	0.0	1.0	

6. We will split our data set into 3 channels: train, test, validation set:

```
# cell 08
train_data, validation_data, test_data = np.split(model_data.sample(frac=1, random_state=1729), [int(0.7 * len(model_data)), int(0.9 * len(model_data))])
```

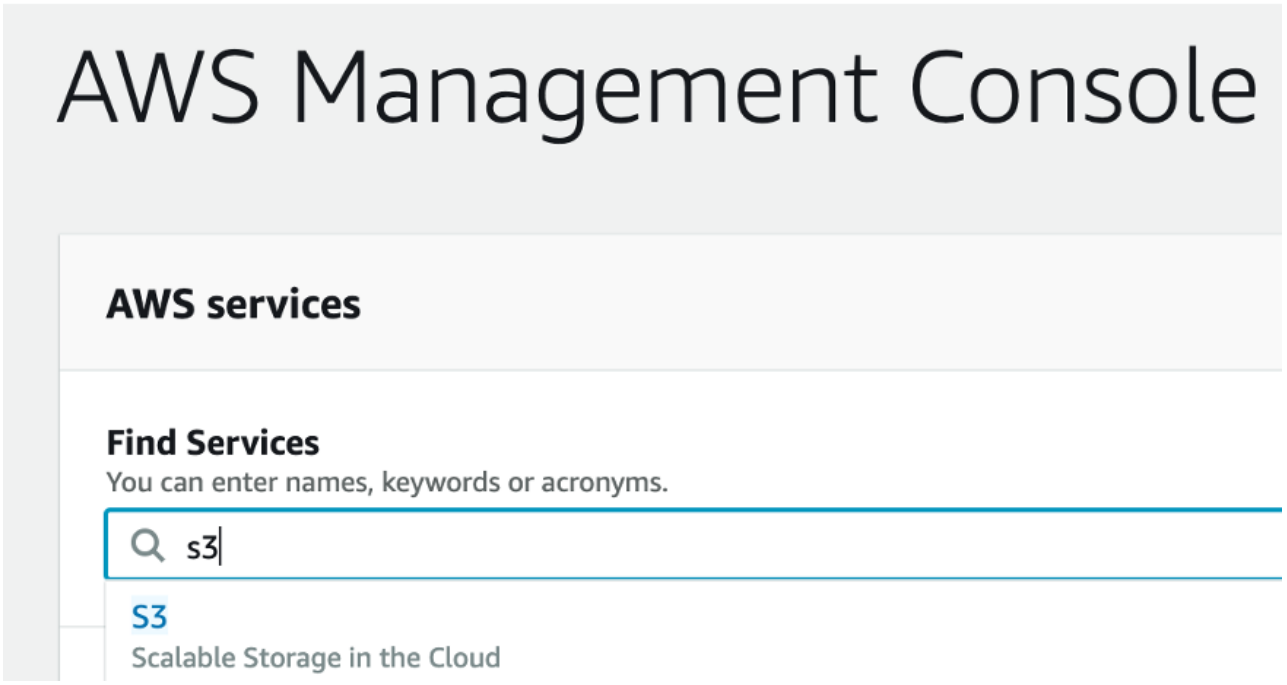
7. Amazon SageMaker XGBoost algorithm expects data to be in libSVM or CSV format (without header) and the first column must be the target variable. So in this step we transform the data accordingly

```
# cell 09
pd.concat([train_data['y_yes'], train_data.drop(['y_no', 'y_yes'], axis=1)], axis=1).to_csv('train.csv', index=False, header=False)
pd.concat([validation_data['y_yes'], validation_data.drop(['y_no', 'y_yes'], axis=1)], axis=1).to_csv('validation.csv', index=False, header=False)
```

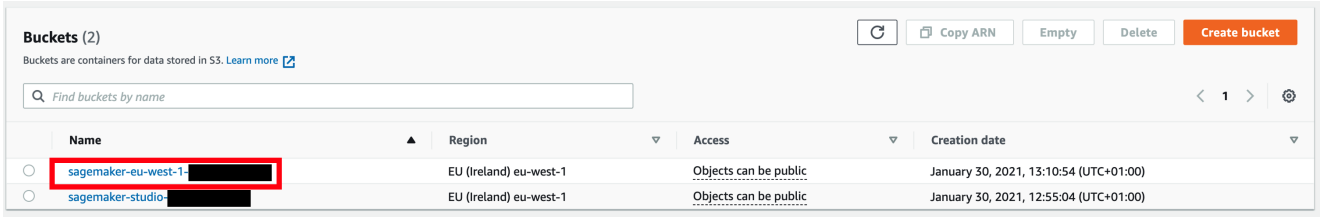
8. Now we will upload the final data into Amazon S3 bucket.

```
# cell 10
boto3.Session().resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'train/train.csv')).upload_file('train.csv')
boto3.Session().resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'validation/validation.csv')).upload_file('validation.csv')
```

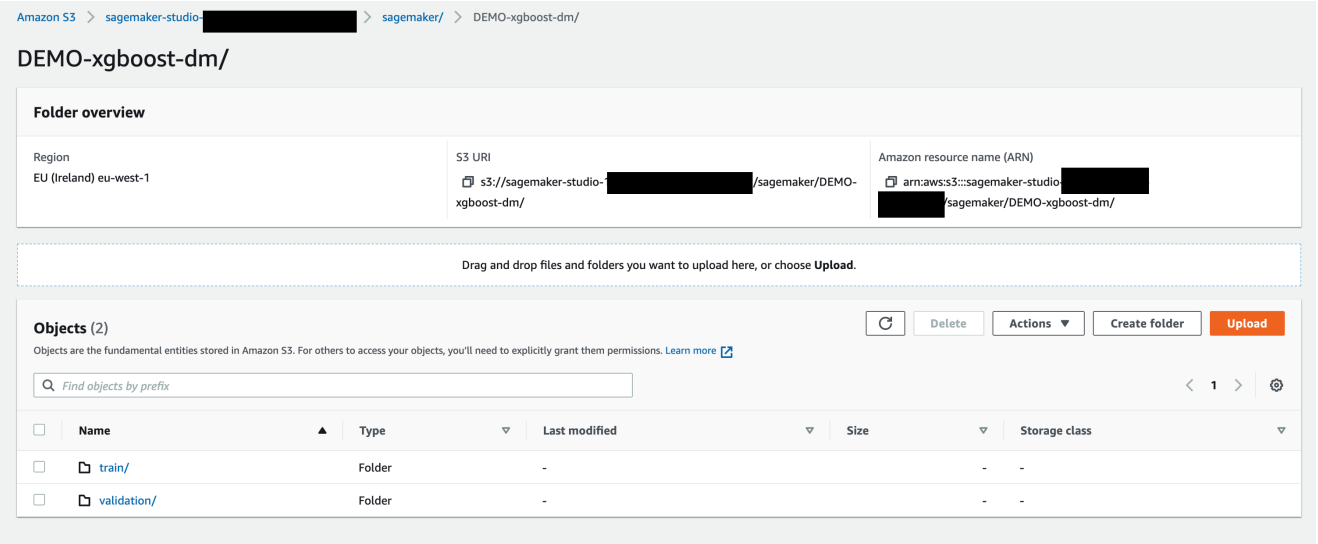
9. Now check the S3 bucket through the console to make sure you have uploaded the train.csv and validation.csv. In the AWS Console Main Page, type **S3**.



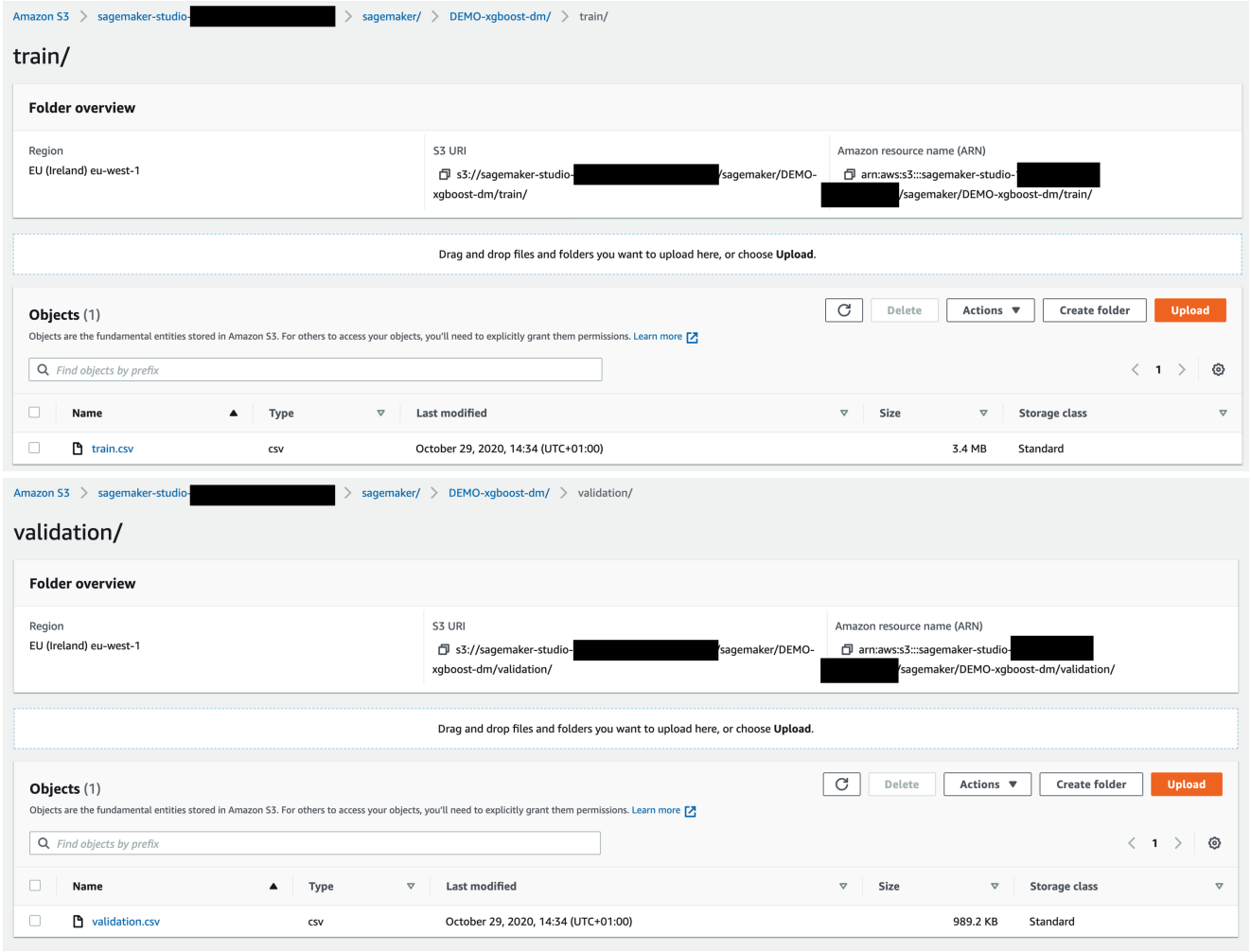
10. Click on your bucket name. If two buckets are present, take the one including your region name (**us-est-1** as an example in the image below).



11. Click on **sagemaker/** and then on **DEMO-xgboost-dm/**. You will see a **train/** and **validation/** folder.



12. You can have a look inside of each folder to make sure that the **train.csv** file is there as well as the **validation.csv** file.



Use this notebook to complete the next section of this lab (**Train & Tune the model**).

Train & Tune the model

In this step, you will train your machine learning model with the training dataset, which you uploaded in Amazon S3 bucket in the previous Lab (Lab 1 - Feature Engineering). **To run this lab, you need to have Lab 1 (Option 1, 2 or 3) completed as a prerequisite. You will continue to work in the notebook that you have used in Lab 1 (just after the section "End of Lab 1")**.

End of Lab 1

Training

Now we know that most of our features have skewed distributions, some are highly correlated with one another, and some appear to have non-linear relationships with our target variable. Also, for targeting future prospects, good predictive accuracy is preferred to being able to explain why that prospect was targeted. Taken together, these aspects make gradient boosted trees a good candidate algorithm.

There are several intricacies to understanding the algorithm, but at a high level, gradient boosted trees works by combining predictions from many simple models, each of which tries to address the weaknesses of the previous models. By doing this the collection of simple models can actually outperform large, complex models. Other Amazon SageMaker notebooks elaborate on gradient boosting trees further and how they differ from similar algorithms.

`xgboost` is an extremely popular, open-source package for gradient boosted trees. It is computationally powerful, fully featured, and has been successfully used in many machine learning competitions. Let's start with a simple `xgboost` model, trained using Amazon SageMaker's managed, distributed training framework.

First we'll need to specify the ECR container location for Amazon SageMaker's implementation of XGBoost.

1. First, we will specify XGBoost ECR container location.

```
# cell 11
container = sagemaker.image_uris.retrieve(region=boto3.Session().region_name, framework='xgboost', version='latest')
```

2. Specify train and validation data set location:

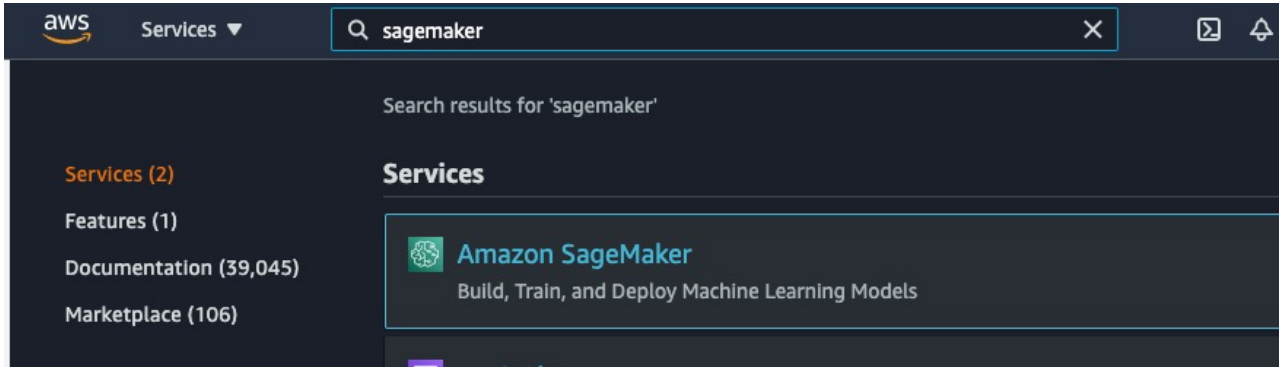
```
# cell 12
s3_input_train = sagemaker.inputs.TrainingInput(s3_data='s3://{}/{}/train'.format(bucket, prefix), content_type='csv')
s3_input_validation = sagemaker.inputs.TrainingInput(s3_data='s3://{}/{}/validation/'.format(bucket, prefix), content_type='csv')
```

3. Now, define the training parameters in SageMaker Estimator. You will also define the tuning hyperparameters in set_hyperparameters and then call the “fit” method to train the model.

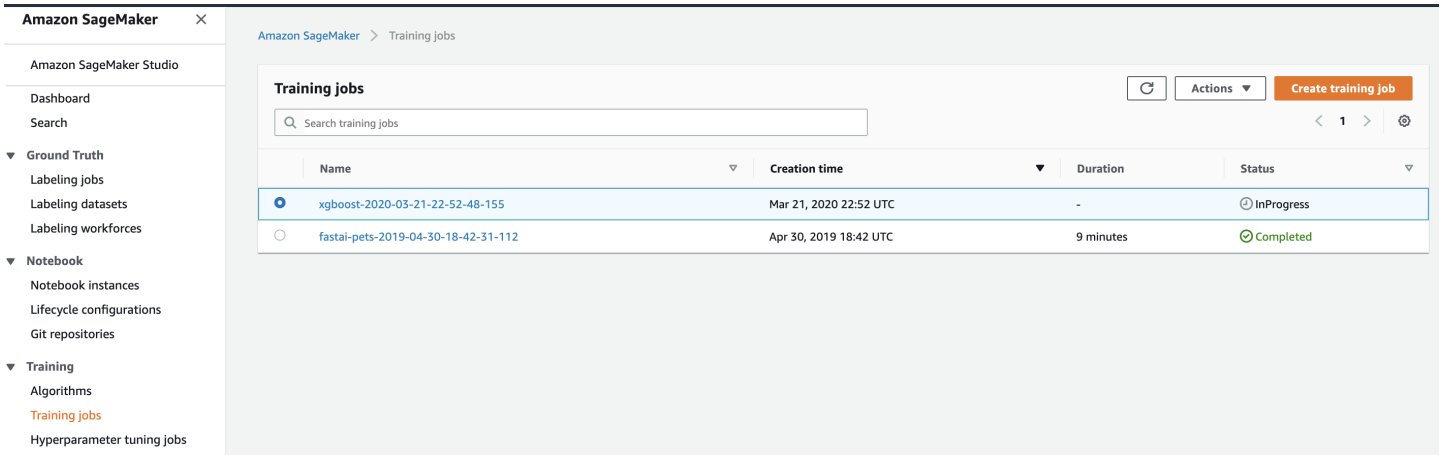
```
# cell 13
sess = sagemaker.Session()

xgb = sagemaker.estimator.Estimator(container,
                                    role,
                                    instance_count=1,
                                    instance_type='ml.m4.xlarge',
                                    output_path='s3://{}/{}/output'.format(bucket, prefix),
                                    sagemaker_session=sess)
xgb.set_hyperparameters(max_depth=5,
                        eta=0.2,
                        gamma=4,
                        min_child_weight=6,
                        subsample=0.8,
                        silent=0,
                        objective='binary:logistic',
                        num_round=100)

xgb.fit({'train': s3_input_train, 'validation': s3_input_validation})
```

```
INFO:sagemaker:Creating training-job with name: xgboost-2023-04-24-17-20-24-046
2023-04-24 17:20:24 Starting - Starting the training job...
2023-04-24 17:20:59 Starting - Preparing the instances for training.........
2023-04-24 17:22:07 Downloading - Downloading input data...
2023-04-24 17:22:37 Training - Downloading the training image...
2023-04-24 17:23:33 Uploading - Uploading generated training modelArguments: train
[2023-04-24:17:23:24:INFO] Running standalone xgboost training.
[2023-04-24:17:23:24:INFO] File size need to be processed in the node: 15.86mb. Available memory size in the node: 8590.14mb
[2023-04-24:17:23:24:INFO] Determined delimiter of CSV input is ','
[17:23:24] S3DistributionType set as FullyReplicated
```

4. You can also check from AWS console to verify the training job started and wait until the status becomes “Completed”. Go to SageMaker from the AWS Console Main Page.



5. Click on the left pane and choose “Training jobs”. You will see the job first in “InProgress” state. Wait for 3-5 mins till the training job completes and the status becomes “Completed”.



You successfully trained the XGBoost model!!

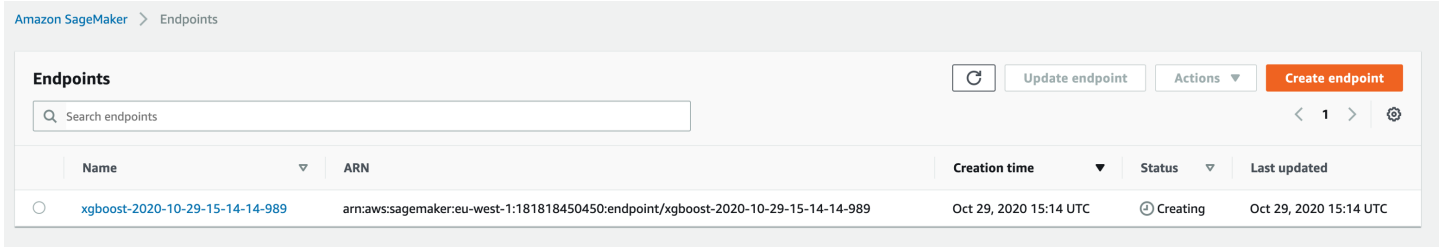
Deploy the model

In this step, you will deploy the trained model to a real-time HTTPS endpoint. This process can take around 6-8 min. You can also choose our newer instance type such as “ml.m5.xlarge” for this deployment.

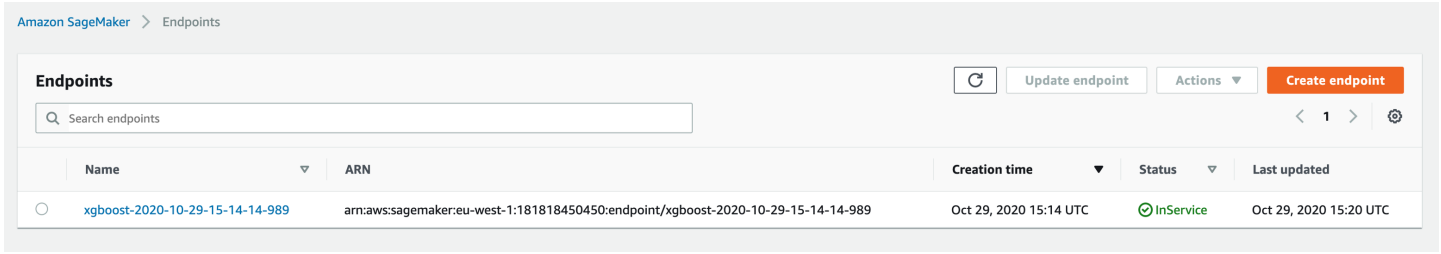
```
# cell 14
xgb_predictor = xgb.deploy(initial_instance_count=1,
                           instance_type='ml.m4.xlarge')
```

```
INFO:sagemaker:Creating model with name: xgboost-2023-04-24-17-24-17-949
INFO:sagemaker:Creating endpoint-config with name xgboost-2023-04-24-17-24-17-949
INFO:sagemaker:Creating endpoint with name xgboost-2023-04-24-17-24-17-949
---------!
```

Now we will check the SageMaker endpoint deployment in the AWS console view as well. In the SageMaker AWS Console, go to “Endpoints” on the left pane. You will see the endpoint in “Creating” state.



It will then transition to “InService” state.



Predict and Evaluate model performance

In this step you will reformat the CSV data, then run the model to create predictions. You will then evaluate model performance using a confusion matrix. In this case, we're predicting whether the customer will subscribe to a term deposit (1) or not (0).

1. First we'll need to determine how we pass data into and receive data from our endpoint. Our data is currently stored as NumPy arrays in memory of our notebook instance. To send it in an HTTP POST request, we'll serialize it as a CSV string and then decode the resulting CSV.

Note: For inference with CSV format, SageMaker XGBoost requires that the data does NOT include the target variable.

```
# cell 15
xgb_predictor.serializer = sagemaker.serializers.CSVSerializer()
```

2. Now, you will create a simple function to:

- Loop over our test dataset.
- Split it into mini-batches of rows.
- Convert those mini-batches to CSV string payloads (notice, we drop the target variable from our dataset first).
- Retrieve mini-batch predictions by invoking the XGBoost endpoint.

- Collect predictions and convert from the CSV output our model provides into a NumPy array.

```
# cell 16
def predict(data, predictor, rows=500 ):
    split_array = np.array_split(data, int(data.shape[0] / float(rows) + 1))
    predictions = ''
    for array in split_array:
        predictions = ','.join([predictions, predictor.predict(array).decode('utf-8')])

    return np.fromstring(predictions[1:], sep=',')

predictions = predict(test_data.drop(['y_no', 'y_yes'], axis=1).to_numpy(), xgb_predictor)
```

3. Now we'll check our confusion matrix to see how well we predicted versus actuals.

```
# cell 17q
pd.crosstab(index=test_data['y_yes'], columns=np.round(predictions), rownames=['actuals'], colnames=['predictions'])
```

predictions	0.0	1.0
actuals		
0.0	3366	137
1.0	435	3001

So, out of 6939 potential customers, model predicted 3138 would subscribe and 3001 of them actually did. We also had 435 subscribers who actually subscribed that model did not predict would.

Disclaimer: This matrix may vary if you change the parameters along the way.

Automatic model Tuning (optional)

Amazon SageMaker automatic model tuning, also known as hyperparameter tuning, finds the best version of a model by running many training jobs on your dataset using the algorithm and ranges of hyperparameters that you specify. It then chooses the hyperparameter values that result in a model that performs the best, as measured by a metric that you choose.

For example, suppose that you want to solve a *binary classification* problem on this marketing dataset. Your goal is to maximize the *area under the curve (auc)* metric of the algorithm by training an XGBoost Algorithm model. You don't know which values of the eta, alpha, min_child_weight, and max_depth hyperparameters to use to train the best model. To find the best values for these hyperparameters, you can specify ranges of values that Amazon SageMaker hyperparameter tuning searches to find the combination of values that results in the training job that performs the best as measured by the objective metric that you chose. Hyperparameter

tuning launches training jobs that use hyperparameter values in the ranges that you specified, and returns the training job with highest auc.

1. We will tune four hyperparameters in this example:

- *eta*: Step size shrinkage used in updates to prevent overfitting. After each boosting step, you can directly get the weights of new features. The eta parameter actually shrinks the feature weights to make the boosting process more conservative.
- *min_child_weight*: Minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than min_child_weight, the building process gives up further partitioning. In linear regression models, this simply corresponds to a minimum number of instances needed in each node. The larger the algorithm, the more conservative it is.
- *alpha*: L1 regularization term on weights. Increasing this value makes models more conservative.
- *max_depth*: Maximum depth of a tree. Increasing this value makes the model more complex and likely to be overfitted.

```
# cell 18
from sagemaker.tuner import IntegerParameter, CategoricalParameter, ContinuousParameter, HyperparameterTuner
hyperparameter_ranges = {'eta': ContinuousParameter(0, 1),
                         'min_child_weight': ContinuousParameter(1, 10),
                         'alpha': ContinuousParameter(0, 2),
                         'max_depth': IntegerParameter(1, 10)}
```

2. Next we'll specify the objective metric that we'd like to tune and its definition, which includes the regular expression (Regex) needed to extract that metric from the CloudWatch logs of the training job. Since we are using built-in XGBoost algorithm here, it emits two predefined metrics: validation:auc and train:auc, and we elected to monitor validation:auc as you can see below. In this case, we only need to specify the metric name and do not need to provide regex. If you bring your own algorithm, your algorithm emits metrics by itself. In that case, you'll need to add a MetricDefinition object here to define the format of those metrics through regex, so that SageMaker knows how to extract those metrics from your CloudWatch logs.

```
# cell 19
objective_metric_name = 'validation:auc'
```

3. Now, we'll create a HyperparameterTuner object, to which we pass:

   - The XGBoost estimator we created above
   - Our hyperparameter ranges
   - Objective metric name and definition

- Tuning resource configurations such as Number of training jobs to run in total and how many training jobs can be run in parallel.

```
# cell 20
tuner = HyperparameterTuner(xgb,
                            objective_metric_name,
                            hyperparameter_ranges,
                            max_jobs=20,
                            max_parallel_jobs=3)
```

4. Now we can launch a hyperparameter tuning job by calling fit() function. After the hyperparameter tuning job is created, we can go to SageMaker console to track the progress of the hyperparameter tuning job until it is completed.

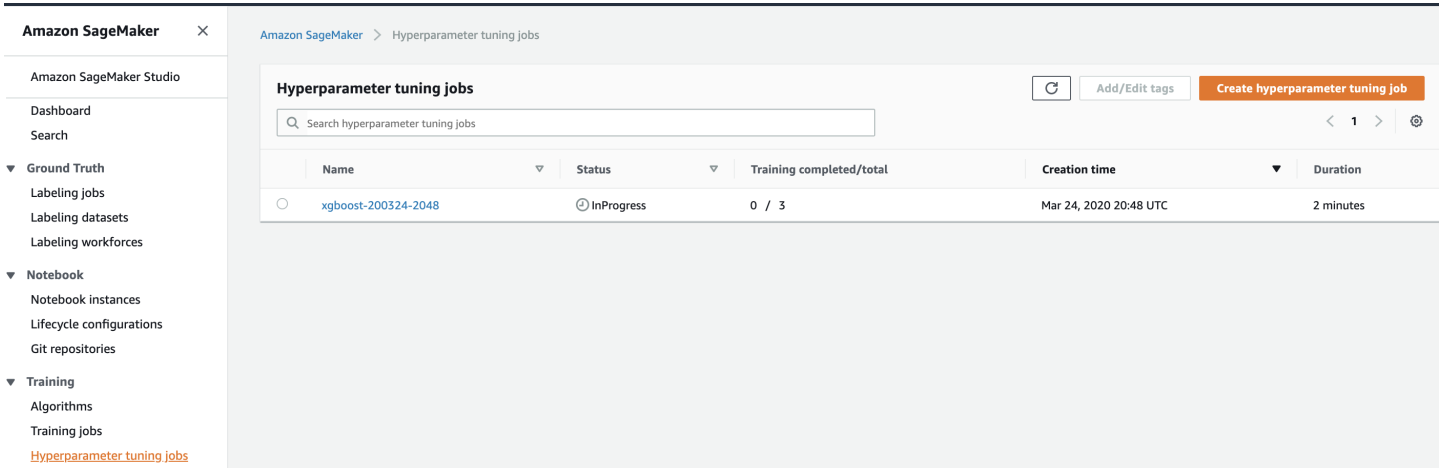
```
# cell 21
tuner.fit({'train': s3_input_train, 'validation': s3_input_validation})
```

5. Let's just run a quick check of the hyperparameter tuning jobs status by using below command. Output should be “InProgress” . That means job started successfully.

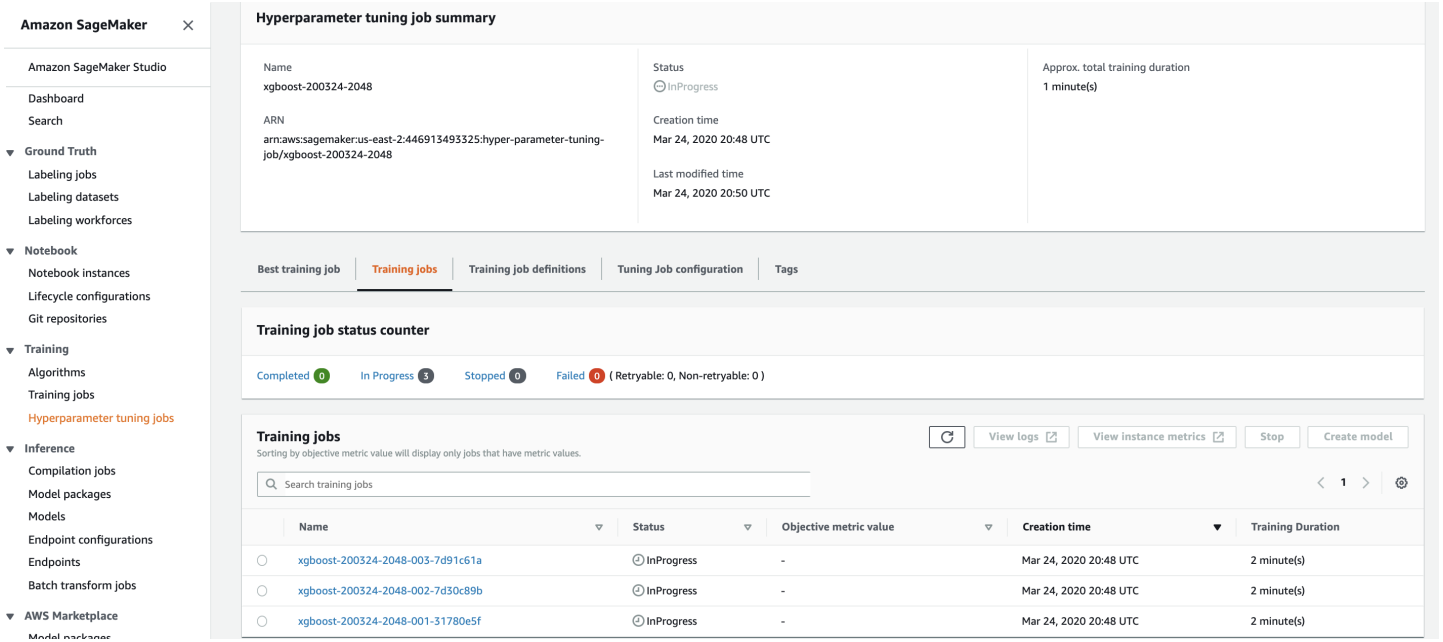
```
# cell 22
boto3.client('sagemaker').describe_hyper_parameter_tuning_job(
HyperParameterTuningJobName=tuner.latest_tuning_job.job_name)['HyperParameterTuningJobStatus']
```

```
'InProgress'
```

6. we can go to SageMaker console to track the progress of the hyperparameter tuning job until it is completed. It will take around **30 mins** to gets completed.



7. Select the Tuning job to see the detail view:



8. (Optional) Once the tuning job is completed, you can pick the training job with the best performance, deploy, predict and evaluate the model developed by the job as done before.

```
# cell 23
# return the best training job name
tuner.best_training_job()
```

```
# cell 24
#  Deploy the best trained or user specified model to an Amazon SageMaker endpoint
tuner_predictor = tuner.deploy(initial_instance_count=1,
                               instance_type='ml.m4.xlarge')
```

```
# cell 25
# Create a serializer
tuner_predictor.serializer = sagemaker.serializers.CSVSerializer()
```

```
# cell 26
# Predict
predictions = predict(test_data.drop(['y_no', 'y_yes'], axis=1).to_numpy(),tuner_predictor)
```

```
# cell 27
# Collect predictions and convert from the CSV output our model provides into a NumPy array
pd.crosstab(index=test_data['y_yes'], columns=np.round(predictions), rownames=['actuals'], colnames=['predictions'])
```

Clean Up

Once you done with this notebook, you can run the below cell to remove the hosted endpoint to avoid any charges

Clean-up

If you are done with this notebook, please run the cell below. This will remove the hosted endpoint you created and avoid any charges from a stray instance being left on.

```
# cell 29
tuner_predictor.delete_endpoint(delete_endpoint_config=True)
```

Conclusion

In this lab you have walked through the process of build, train, tune and deploy XGBoost model using Sage maker built in algorithm. In this lab you have used SageMaker Python SDK to train, tune and deploy the model. You have also looked the console view while you do the training, automatic model tuning and hosting the model.