How Twilio used Amazon SageMaker MLOps Pipelines with PrestoDB to enable frequent model re-training and optimized batch transform

*Amit Arora*, *Madhur Prashant*, *Antara Raisa*, *Johnny Chivers*

***This post is co-written with customer\_names from Twilio.***

[PLACEHOLDER (Twilio to add information here): Twilio is an American cloud communications company based in San Francisco, California, which provides programmable communication tools for making and receiving phone calls, sending and receiving text messages, and performing other communication functions using its web service APIs.] Being one of the largest AWS customers, Twilio engages with Data and AI/ML services to run their daily workloads. This blog outlines the steps AWS and Twilio took to migrate Twilio’s existing Machine Learning Operations (MLOps), implementation of training models and running batch inferences to Amazon SageMaker.

Machine learning (ML) models do not operate in isolation. They must integrate into existing production systems and infrastructure to deliver value. This necessitates considering the entire ML lifecycle during design and development. With the right processes and tools, MLOps enables organizations to reliably and efficiently adopt ML across their teams for their specific use cases. [Amazon SageMaker MLOps](https://aws.amazon.com/sagemaker/mlops/?sagemaker-data-wrangler-whats-new.sort-by=item.additionalFields.postDateTime&sagemaker-data-wrangler-whats-new.sort-order=desc) is a suite of features that includes [Amazon SageMaker Pipelines](https://aws.amazon.com/sagemaker/pipelines/), that allows for straightforward creation and management of ML workflows, while also offering storage and reuse capabilities for workflow steps and [Amazon SageMaker Model Registry](https://docs.aws.amazon.com/sagemaker/latest/dg/model-registry.html) that centralizes model tracking, simplifying model deployment.

This blog post focuses on enabling AWS customers to have flexibility for using their data source of choice, and integrate it seamlessly with [Amazon SageMaker Processing Jobs](https://sagemaker-examples.readthedocs.io/en/latest/sagemaker_processing/scikit_learn_data_processing_and_model_evaluation/scikit_learn_data_processing_and_model_evaluation.html). Using SageMaker Processing Jobs, you can leverage a simplified, managed experience to run data pre- or post-processing and model evaluation workloads on the Amazon SageMaker platform.

[Twilio](https://pages.twilio.com/twilio-brand-sales-namer-1?utm_source=google&utm_medium=cpc&utm_term=twilio&utm_campaign=G_S_NAMER_Brand_Twilio_Tier1&cq_plac=&cq_net=g&cq_pos=&cq_med=&cq_plt=gp&gad_source=1&gclid=CjwKCAjwtqmwBhBVEiwAL-WAYd5PgxP-XSLDYBvu6y_j8KUydoj33QX3XWpUo4zEm2DLzgn_bfdogBoC9dIQAvD_BwE) needed to implement a MLOps pipeline that queried data from [PrestoDB](https://prestodb.io/). PrestoDB is an open-source SQL query engine that is designed for fast analytic queries against data of any size from multiple sources.

In this post, we show you a step-by-step implementation to achieve the following:

* Read data available in PrestoDB from a SageMaker Processing Job
* Train a binary classification model using [SageMaker Training Jobs](https://sagemaker.readthedocs.io/en/v1.44.4/amazon_sagemaker_operators_for_kubernetes_jobs.html) and tune the model using [SageMaker Automatic Model Tuning](https://docs.aws.amazon.com/sagemaker/latest/dg/automatic-model-tuning.html)
* Run a [Batch Transform pipeline](https://docs.aws.amazon.com/sagemaker/latest/dg/batch-transform.html) for batch inference on data fetched from PrestoDB
* Deploy the trained model as a [Real-Time SageMaker Endpoint](https://docs.aws.amazon.com/sagemaker/latest/dg/realtime-endpoints.html)

## Use case overview

Burner phones, or ***burners*** are phone numbers which are available online to everyone and are used to hide identities by creating fake accounts on customers’ apps/websites. Twilio trained a binary classification Machine Learning (ML)model using [scikit-learn’s](https://scikit-learn.org/stable/) [RandomForestClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html). This model is used as part of a batch process that runs periodically to detect burner numbers. The training data used for this pipeline is made available via PrestoDB and is read into Pandas through the [PrestoDB Python client](https://pypi.org/project/presto-python-client/).

The end goal was to convert the existing steps into two pipelines: a training pipeline and a batch transform pipeline that connected the data queried from PrestoDB to a SageMaker Processing Job, and deploy the trained model to a SageMaker Endpoint for real-time inference.

In this blog, we use an open-source dataset available via the [TPCH-Connector](https://prestodb.io/docs/current/connector/tpch.html) that is packaged with PrestoDB to illustrate the end-to-end workflow that Twilio used. Twilio was able to use this open-source solution to migrate their burner model and existing MLOps pipeline to Amazon SageMaker. All the code for this open-source solution is available in the [GitHub](https://github.com/aws-samples/mlops-pipeline-prestodb?tab=readme-ov-file) repo.

## Solution overview

This solution is divided into three main steps that implement a training pipeline, a batch transform pipeline, and deploy the trained model as a real-time SageMaker Endpoint for inference:

* [Model Training Pipeline](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/0_model_training_pipeline.ipynb): In this step, we create a model training pipeline. We connect a SageMaker Processing Job to fetch data from a PrestoDB instance, train and tune the ML model, evaluate and register it with the SageMaker Model Registry.
* [Batch Transform Pipeline](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/1_batch_transform_pipeline.ipynb): In this step, we create a batch transform pipeline. Here, we execute a preprocess data step that reads data from a PrestoDB instance and runs batch inference on the registered ML model (from the Model Registry) that we [Approve](https://docs.aws.amazon.com/sagemaker/latest/dg/model-registry-approve.html) as a part of this pipeline. This model is approved either programmatically or manually via the Model Registry.
* [Real-time Inference](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/2_realtime_inference.ipynb): In this step, we deploy the latest approved model as a SageMaker Endpoint for [Real-Time inference](https://docs.aws.amazon.com/sagemaker/latest/dg/realtime-endpoints.html).

## Solution design

The solution design includes setting up the data preparation and training pipeline, implementing the batch transform pipeline, and deploying the approved model as a real time SageMaker Endpoint for inference. All [pipeline parameters](https://docs.aws.amazon.com/sagemaker/latest/dg/build-and-manage-parameters.html) used in this solution exist in a single [config.yml](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/config.yml) file. This file includes: necessary AWS and PrestoDB credentials to connect to the PrestoDB instance, information on the training [hyperparameters](https://sagemaker.readthedocs.io/en/stable/api/utility/hyperparameters.html) and [SQL](https://aws.amazon.com/what-is/sql/#:~:text=Structured%20query%20language%20(SQL)%20is,relationships%20between%20the%20data%20values.) queries that are run at training and inference steps to read data from PrestoDB. This solution is highly customizable for industry specific use cases so that it can be used with minimal-no code changes via simple config file updates.

An example of how a query is configured within the config.yml file is shown below. This query is used at the [data processing step](https://docs.aws.amazon.com/sagemaker/latest/dg/build-and-manage-steps.html#step-type-processing) of the training pipeline to fetch data from the PrestoDB instance. Here, we predict whether an order is a high\_value\_order or a low\_value\_order based on the orderpriority as given from the TPCH-data. More information on the TPCH-data, Database Entities, Relationships, and Characteristics can be found [here](https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.17.1.pdf). Users can change the query for their use case within the config file and run the solution with no code changes.

SELECT  
 o.orderkey,  
 COUNT(l.linenumber) AS lineitem\_count,  
 SUM(l.quantity) AS total\_quantity,  
 AVG(l.discount) AS avg\_discount,  
 SUM(l.extendedprice) AS total\_extended\_price,  
 SUM(l.tax) AS total\_payable\_tax,  
 o.orderdate,  
 o.orderpriority,  
 CASE  
 WHEN (o.orderpriority = '2-HIGH') THEN 1   
 ELSE 0  
 END AS high\_value\_order  
 FROM  
 orders o  
 JOIN  
 lineitem l ON o.orderkey = l.orderkey  
 GROUP BY  
 o.orderkey,  
 o.orderdate,  
 o.orderpriority  
 ORDER BY   
 RANDOM()   
 LIMIT 5000

The main steps of this solution are as described in detail below:

### Part 1 - [Data Preparation and Training Pipeline Step](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/0_model_training_pipeline.ipynb):

1. The training data is read from a PrestoDB instance, and any feature engineering needed is done as part of the SQL queries run in PrestoDB at retrieval time. The queries that are used to fetch data at training and batch inference steps are configured in the [config file](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/config.yml).
2. We use the [FrameworkProcessor](https://docs.aws.amazon.com/sagemaker/latest/dg/processing-job-frameworks.html) with SageMaker Processing Jobs to read data from PrestoDB using the Python PrestoDB client.
3. For the training and tuning step, we use the [SKLearn estimator](https://sagemaker.readthedocs.io/en/stable/frameworks/sklearn/sagemaker.sklearn.html) from the SageMaker SDK and the RandomForestClassifier from scikit-learn to train the ML model. The [HyperparameterTuner](https://sagemaker.readthedocs.io/en/stable/api/training/tuner.html) class is used for running automatic model tuning that finds the best version of the model by running many training jobs on the dataset using the algorithm and the ranges of hyperparameters.
4. The [Model Evaluation](https://sagemaker-examples.readthedocs.io/en/latest/sagemaker-pipelines/tabular/abalone_build_train_deploy/sagemaker-pipelines-preprocess-train-evaluate-batch-transform.html) step is to check that the trained and tuned model has an accuracy level above a user-defined threshold and only then [register that model](https://docs.aws.amazon.com/sagemaker/latest/dg/model-registry.html) within the Model Registry. If the model accuracy does not meet the threshold then the pipeline fails and the model is not registered with the Model Registry.
5. The model training pipeline is then run with the [pipeline.start](https://docs.aws.amazon.com/sagemaker/latest/dg/run-pipeline.html) which triggers and instantiates all steps mentioned above.

### Part 2 - [Batch Transform Step](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/1_batch_transform_pipeline.ipynb):

1. The batch transform pipeline implements a data preparation step that retrieves data from a PrestoDB instance (using a [data preprocess script](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/code/presto_preprocess_for_batch_inference.py)) and stores the batch data in S3.
2. We approve the latest model registered in the Model Registry from the training pipeline.
3. We create a [Transformer](https://sagemaker.readthedocs.io/en/stable/api/inference/transformer.html) instance and use it to run a batch transform job to get inferences on the entire dataset stored in S3 from the data preparation step and store the output in S3.

### Part 3 - [Real Time SageMaker Endpoint Support](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/2_realtime_inference.ipynb):

1. The latest approved model is retrieved from the Model Registry using the [describe\_model\_package](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/sagemaker/client/describe_model_package.html) function from the SageMaker SDK.
2. The latest approved model is deployed as a real-time SageMaker Endpoint.
3. The model is deployed on a ml.c5.xlarge instance with a minimum instance count of 1 and maximum instance count of 3 (configurable by the user) with the [automatic scaling policy](https://docs.aws.amazon.com/sagemaker/latest/dg/endpoint-auto-scaling.html) ENABLED. This removes unnecessary instances so that you don’t pay for provisioned instances that you aren’t using.

## Prerequisites

To implement the solution provided in this post, you should have an [AWS account](https://signin.aws.amazon.com/signin?redirect_uri=https%3A%2F%2Fportal.aws.amazon.com%2Fbilling%2Fsignup%2Fresume&client_id=signup), set up an [Amazon SageMaker Domain](https://docs.aws.amazon.com/sagemaker/latest/dg/sm-domain.html) to access [SageMaker Studio](https://aws.amazon.com/sagemaker/studio/) and have familiarity with SageMaker, S3, and PrestoDB.

The following prerequisites need to be in place before running this code.

#### PrestoDB

* We use the built-in datasets available in PrestoDB via the TPCH-connector for this solution. Follow the instructions in the GitHub [README.md](https://github.com/aws-samples/mlops-pipeline-prestodb?tab=readme-ov-file#prestodb) to setup PrestoDB on an Amazon EC2 instance in your account. ***If you already have access to a PrestoDB instance then you can skip this section but keep its connection details handy (see the presto section in the*** [***config***](./config.yml) ***file)***. Once you have your PrestoDB credentials, fill out the presto section in the [config](./config.yml) as given below. Enter your host public IP, port, credentials, catalog and schema:

presto:  
 host: <0.0.0.0>  
 parameter: "0000"  
 presto\_credentials: <presto\_credentials>  
 catalog: <catalog>  
 schema: <schema>

#### [Amazon VPC](https://aws.amazon.com/vpc/) Network Configurations

* We also define the network configurations of the machine learning model and operations in the [config](./config.yml) file. In the aws section, specify the enable\_network\_isolation status, security\_group\_ids, and subnets based on your network isolation preferences. View more information on network configurations and preferences [here](https://docs.aws.amazon.com/sagemaker/latest/dg/mkt-algo-model-internet-free.html):

network\_config:  
 enable\_network\_isolation: false  
 security\_group\_ids:   
 - <security\_group\_id>  
 subnets:  
 - <subnet-1>  
 - <subnet-2>  
 - <subnet-3>

#### IAM Role

Set up an execution role in [AWS Identity and Access Management (IAM)](https://aws.amazon.com/iam/) with appropriate permissions to allow SageMaker to access [AWS Secrets Manager](https://docs.aws.amazon.com/secretsmanager/latest/userguide/intro.html), [Amazon S3](https://aws.amazon.com/pm/serv-s3/?gclid=Cj0KCQjw2a6wBhCVARIsABPeH1sVCmK3CK8Vsv31A4fjV79s5YkxGqKoyDuv2rPuoBDfDqwh7ZiYaTQaAkeOEALw_wcB&trk=fecf68c9-3874-4ae2-a7ed-72b6d19c8034&sc_channel=ps&ef_id=Cj0KCQjw2a6wBhCVARIsABPeH1sVCmK3CK8Vsv31A4fjV79s5YkxGqKoyDuv2rPuoBDfDqwh7ZiYaTQaAkeOEALw_wcB:G:s&s_kwcid=AL!4422!3!536452728638!e!!g!!amazon%20s3!11204620052!112938567994) and other services within your AWS account. ***Until a AWS CloudFormation template is provided which creates the role with the requisite IAM permissions, use a SageMaker execution role that AmazonSageMakerFullAccess AWS managed policy for your execution role.*** Follow the instructions [here](https://github.com/aws-samples/amazon-sagemaker-w-snowflake-as-datasource/tree/main/iam) to create permissions for your iam roles.

#### AWS Secrets Manager

Setup a secret in Secrets Manager for the PrestoDB username and password. Call the secret prestodb-credentials and add a username field to it and a password field to it. For instructions on creating and managing secrets via Secrets Manager, view [this](https://docs.aws.amazon.com/secretsmanager/latest/userguide/managing-secrets.html).

### Steps to run

1. Clone the [code repo](https://github.com/aws-samples/mlops-pipeline-prestodb.git) in SageMaker Studio. Follow this [link](https://docs.aws.amazon.com/sagemaker/latest/dg/studio-tasks-git.html) to view instructions on cloning a git repository in SageMaker Studio.
2. Edit the [config.yml](./config.yml) file as follows:
   * Edit the parameter values in the presto section. These parameters define the connectivity to PrestoDB.
   * Edit the parameter values in the aws section. These parameters define the newtork connectivity, IAM role, bucket name, region and other AWS cloud related parameters.
   * Edit the parameter values in the sections corresponding to the pipeline steps i.e. training\_step, tuning\_step, transform\_step etc. Review all the parameters in these sections carefully and edit them as appropriate for your use-case.
   * Review all the parameters in these sections carefully and edit them as appropriate for your use case.

## Testing the solution

### AWS Architecture

Once the prerequisites are complete and the config.yml file is set up correctly, we are ready to run the [mlops-pipeline-prestodb](https://github.com/aws-samples/mlops-pipeline-prestodb/tree/main) solution. View the architecture diagram for a visual representation of the steps that we implement. This diagram shows the following three steps: the training pipeline, batch transform pipeline and deploying the model as a SageMaker Real-Time Endpoint:

|  |
| --- |
| Figure 1: mlops-pipeline-prestodb AWS Architecture Diagram |

* In the first block on the left, we see the architectural representation of our training pipeline. This includes the data preprocessing step, training and tuning step, model evaluation, condition step and lastly the register model step. The train, test, validation datasets and the [evaluation report](https://sagemaker-examples.readthedocs.io/en/latest/sagemaker-pipelines/tabular/abalone_build_train_deploy/sagemaker-pipelines-preprocess-train-evaluate-batch-transform.html) that are generated in this pipeline are sent to an S3 bucket.
* In the second block on the right, we see the architectural representation of our batch transform pipeline. This includes the batch data preprocessing step, approving the latest model from the model registry, creating the model instance and performing batch transform on data that is stored and retrieved from an S3 bucket.
* The PrestoDB server is hosted on an Amazon EC2 instance, with credentials stored in [AWS Secrets Manager](https://aws.amazon.com/secrets-manager/).
* Finally, the latest approved model from the SageMaker Model Registry is deployed as a SageMaker Real-Time Endpoint for inference.

### Solution Walkthrough

1. On the left panel in [SageMaker Studio](https://aws.amazon.com/sagemaker/studio/), choose **0\_model\_training\_pipeline.inpynb** in the navigation pane. When the notebook is open, on the Run menu, choose **Run All Cells** to run the code in this notebook. This notebook demonstrates how SageMaker Pipelines can be used to string together a sequence of data processing, model training, tuning and evaluation step to train a binary classification machine learning model using scikit-learn. At the end of this run, navigate to [pipelines](https://docs.aws.amazon.com/sagemaker/latest/dg/pipelines-studio.html) on the Studio Navigation pane:

* **After executing the entire training pipeline, your pipeline structure on** [**Amazon SageMaker Pipelines**](https://docs.aws.amazon.com/sagemaker/latest/dg/pipelines-sdk.html) **should look like this:**

|  |
| --- |
| * Figure 2: Training Pipeline Structure |

* ***The training pipeline consists of the following steps that are implemented through the notebook run***:
  1. **Preprocess data step**: In this step of the pipeline, we create a processing job for data pre-pocessing. For more information on processing jobs, see [Process Data](https://docs.aws.amazon.com/sagemaker/latest/dg/processing-job.html). We use a [preprocess script](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/code/presto_preprocess_for_training.py) which is used to connect and query data from a PrestoDB instance using the user specified SQL query in the [config file](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/config.yml). This step splits and sends data retrieved from PrestoDB as tain, test, and validation files to an S3 bucket. Using the data in these files, we can train our ML model.
  + We use the [sklearn\_processor](https://docs.aws.amazon.com/sagemaker/latest/dg/use-scikit-learn-processing-container.html) in our [ProcessingStep](https://docs.aws.amazon.com/sagemaker/latest/dg/build-and-manage-steps.html#step-type-processing) to run the scikit-learn script that preprocesses data. We define this step as follows:
  + # declare the sk\_learn processer  
    step\_args = sklearn\_processor.run(  
     ## code refers to the data preprocessing script that is responsible for querying data from the PrestoDB instance  
     code=config['scripts']['preprocess\_data'],  
     source\_dir=config['scripts']['source\_dir'],   
     outputs=outputs\_preprocessor,  
     arguments=[  
     "--host", host\_parameter,  
     "--port", port\_parameter,  
     "--presto\_credentials\_key", presto\_parameter,  
     "--region", region\_parameter,  
     "--presto\_catalog", presto\_catalog\_parameter,  
     "--presto\_schema", presto\_schema\_parameter,  
     "--train\_split", train\_split.to\_string(),   
     "--test\_split", test\_split.to\_string(),  
     ],  
     )  
      
     step\_preprocess\_data = ProcessingStep(  
     name=config['data\_processing\_step']['step\_name'],  
     step\_args=step\_args,  
     )
  + Here, we use the config['scripts']['source\_dir'] which points to our data preprocessing script that connects to the PrestoDB instance. Parameters used as arguments in [step\_args](https://docs.aws.amazon.com/sagemaker/latest/dg/build-and-manage-steps.html#:~:text=%2C%0A%20%20%20%20sagemaker_session%3Dpipeline_session%2C%0A-,step_args,-%3D%20pyspark_processor.run(%0A%20%20%20%20inputs)) are configurable and fetched from the config file.
  1. **Train Model Step**: In this step of the pipeline, we create a training job to train a model. For more information on training jobs, see [Train a Model with Amazon SageMaker](https://docs.aws.amazon.com/sagemaker/latest/dg/how-it-works-training.html). Here, we use the [Scikit Learn Estimator](https://sagemaker.readthedocs.io/en/stable/frameworks/sklearn/sagemaker.sklearn.html) from the SageMaker SDK to handle end-to-end training and deployment of custom Scikit-learn code. We use the RandomForestClassifier to train the ML model for our binary classification use case. The [HyperparameterTuner](https://sagemaker.readthedocs.io/en/stable/api/training/tuner.html) class is used for running automatic model tuning to determine the set of hyperparameters that provide the best performance based on a user-defined metric threshold (for example, maximizing the AUC metric).
     + In the code below, we use the sklearn\_estimator object with parameters that are configured in the [config file](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/config.yml) and use a [training script](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/code/training.py) to train the ML model. This step accesses the train, test and validation files that are created as a part of the previous data preprocessing step:
     + # declare a tuning step to use the train and test data to tune the ML model using the `HyperparameterTuner` declared above  
       step\_tuning = TuningStep(  
        name=config['tuning\_step']['step\_name'],  
        tuner=rf\_tuner,  
        inputs={  
        "train": TrainingInput(  
        s3\_data=step\_preprocess\_data.properties.ProcessingOutputConfig.Outputs[  
        "train" ## refer to this  
        ].S3Output.S3Uri,  
        content\_type="text/csv",  
        ),  
        "test": TrainingInput(  
        s3\_data=step\_preprocess\_data.properties.ProcessingOutputConfig.Outputs["test"].S3Output.S3Uri,  
        content\_type="text/csv",  
        ),  
        },  
       )
  2. **Evaluate model step**: This step in the pipeline checks if the trained and tuned model has an accuracy level above a user-defined threshold and only then registers the model with the Model Registry. If the model accuracy does not meet the [user-defined threshold](https://sagemaker-examples.readthedocs.io/en/latest/sagemaker-pipelines/tabular/abalone_build_train_deploy/sagemaker-pipelines-preprocess-train-evaluate-batch-transform.html#Define-a-Model-Evaluation-Step-to-Evaluate-the-Trained-Model) then the pipeline fails and the model is not registered with the Model Registry. We use the [ScriptProcessor](https://docs.aws.amazon.com/sagemaker/latest/dg/processing-container-run-scripts.html) with an [evaluation script](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/code/evaluate.py) that a user creates to evaluate the trained model based on a metric of choice.
     + The evaluation step uses the [evaluation script](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/code/evaluate.py) as a code entry. This script prepares the features, target values and calculates the prediction probabilities using model.predict. At the end of the run, an evaluation report is sent to S3 that contains information on precision, recall, accuracy metrics.
  + step\_evaluate\_model = ProcessingStep(  
     name=config['evaluation\_step']['step\_name'],  
     processor=evaluate\_model\_processor,  
     inputs=[  
     ProcessingInput(  
     source=step\_tuning.get\_top\_model\_s3\_uri(top\_k=0, s3\_bucket=bucket),  
     destination="/opt/ml/processing/model",  
     input\_name="model.tar.gz"   
     ),  
     ProcessingInput(  
     source=step\_preprocess\_data.properties.ProcessingOutputConfig.Outputs["test"].S3Output.S3Uri,  
     destination="/opt/ml/processing/test",  
     input\_name="test.csv"   
     ),  
     ],  
     outputs=[  
     ProcessingOutput(  
     output\_name="evaluation",  
     source="/opt/ml/processing/evaluation",  
     destination=Join(  
     on="/",  
     values=[  
     "s3://{}".format(bucket),  
     prefix,  
     ExecutionVariables.PIPELINE\_EXECUTION\_ID,  
     "evaluation",  
     ]  
     )  
     )  
     ],  
     code = config['scripts']['evaluation'],  
     property\_files=[evaluation\_report],  
     job\_arguments=[  
     "--target", target\_parameter,  
     "--features", feature\_parameter,  
     ]  
    )
    - Once the evaluation step is complete, we can analyze the metrics (Accuracy, Precision and Recall) in the evaluation report that is sent to the S3 bucket. View the image of the report sent below:

|  |
| --- |
| * + Figure 3: Evaluation Report: Accuracy=73.8%, Precision=22.8%, Recall=17.2% |

* 1. **Condition model step**: Once the model is evaluated, we can add conditions to the pipeline with a [ConditionStep](https://sagemaker.readthedocs.io/en/stable/workflows/pipelines/sagemaker.workflow.pipelines.html). This step registers the model only if the given user-defined metric threshold is met. In our solution, we only want to register the new model version with the Model Registry only if the new model meets a specific accuracy condition of above 70%.
  + # Create a SageMaker Pipelines ConditionStep, using the condition above.  
    # Enter the steps to perform if the condition returns True / False.  
    step\_cond = ConditionStep(  
     name=config['condition\_step']['step\_name'],  
     conditions=[cond\_gte],  
     if\_steps=[step\_register\_model],  
     else\_steps=[step\_fail], ## if this fails  
    )
  + If the accuracy condition is not met, a step\_fail step is executed that sends an error message to the user and the pipeline fails. For instance, since the user-defined accuracy condition is set to 0.7 in the config file, and the Accuracy calculated during the evaluation step exceeds it (73.8% > 70%), the outcome of this step is set to True and the model moves to the last step of the training pipeline.
  1. **Register model step**: This RegisterModel step is to register a [sagemaker.model.Model](https://sagemaker.readthedocs.io/en/stable/api/inference/model.html) or a [sagemaker.pipeline.PipelineModel](https://sagemaker.readthedocs.io/en/stable/api/inference/pipeline.html#pipelinemodel) with the Amazon SageMaker model registry. Once the trained model meets the model performance requirements, a new version of the model is registered with the [Model Registry](https://docs.aws.amazon.com/sagemaker/latest/dg/model-registry.html).
  + ***The model is registered with the Model Registry with approval status set to PendingManualApproval. This means the model cannot be deployed on a SageMaker Endpoint unless its status in the registry is changed to Approved manually via the SageMaker console, programmatically or through a Lambda function.***
  + ***Now that the model is registered, you can get access to the registered model manually on the SageMaker studio Model Registry console, or programmatically in the next notebook, approve it and run the second portion of this solution: Batch Transform Step***

1. Next Choose [1\_batch\_transform\_pipeline.ipynb](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/1_batch_transform_pipeline.ipynb). When the notebook is open, on the Run menu, choose **Run All Cells** to run the code in this notebook. This notebook will run a batch transform pipeline using the model trained in the previous notebook.

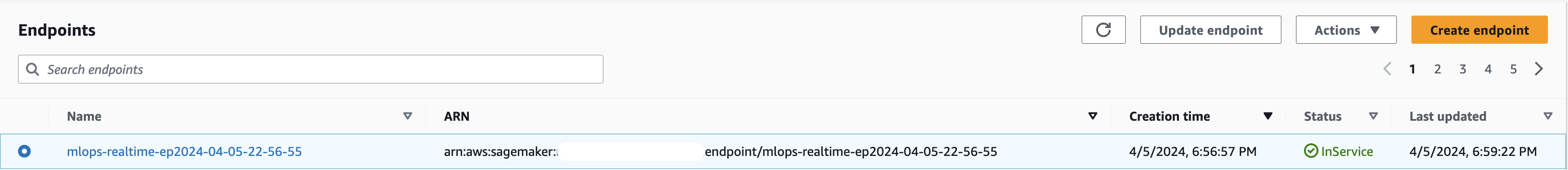
* **At the end of the batch transform pipeline, your pipeline structure on Amazon SageMaker Pipelines should look like this:**

|  |
| --- |
| * Figure 4: Batch Transform Pipeline Structure |

* ***The batch transform pipeline consists of the following steps that are implemented through the notebook run***
  1. **Extract the latest approved model from the SageMaker Model Registry**: In this step of the pipeline, we extract the latest model from the Model Registry, and set the ModelApprovalStatus to Approved:
  + ## updating the latest model package to approved status to use it for batch inference  
    model\_package\_update\_response = sm.update\_model\_package(  
     ModelPackageArn=latest\_model\_package\_arn,  
     ModelApprovalStatus="Approved",  
    )
  + Now we have extracted the latest model from the SageMaker Model Registry, and programmatically approved it. You can also approve the model manually on the [SageMaker Model Registry](https://docs.aws.amazon.com/sagemaker/latest/dg/model-registry.html) page in SageMaker Studio as given in the image below.

|  |
| --- |
| * + Figure 5: SageMaker Model Registry: Manual Model Approval via SageMaker Studio |

* 1. **Read raw data for inference from PrestoDB and store in an Amazon S3 bucket**: Once the latest model is approved, batch data is fetched from the PrestoDB instance and used for the batch transform step. In this step, we use a [batch preprocess script](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/code/presto_preprocess_for_batch_inference.py) that queries data from PrestoDB and saves it in a batch directory within an S3 bucket. The query that is used to fetch batch data is configured by the user within the config file in the transform\_step section.
  + # declare the batch step that is called later in pipeline execution  
    batch\_data\_prep = ProcessingStep(  
     name=config['data\_processing\_step']['step\_name'],  
     step\_args=step\_args,  
    )
  + Once the batch data is extracted into the S3 bucket, create a model instance and point to the [‘inference.py’](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/code/inference.py) script, which contains code that runs as part of getting inference from the trained model.
  + # create the model image based on the model data and refer to the inference script as an entry point for batch inference  
    model = Model(  
     image\_uri=image\_uri,  
     entry\_point=config['scripts']['batch\_inference'],  
     model\_data=model\_data\_url,  
     sagemaker\_session=pipeline\_session,  
     role=role,  
    )
  1. **Create a batch transform step to perform inference on the batch data stored in S3**: Now that a model instance is created, create a [Transformer](https://sagemaker.readthedocs.io/en/stable/api/inference/transformer.html) instance with the appropriate model type, compute instance type, and desired output S3 URI. Specifically, pass in the ModelName from the [CreateModelStep](https://docs.aws.amazon.com/sagemaker/latest/APIReference/API_CreateModel.html), step\_create\_model properties. The CreateModelStep properties attribute matches the object model of the DescribeModel response object. Use a transform step for batch transformation to run inference on an entire dataset. For more information about batch transform, see [Run Batch Transforms with Inference Pipelines](https://docs.aws.amazon.com/sagemaker/latest/dg/inference-pipeline-batch.html).
     + A transform step requires a transformer and the data on which to run batch inference.
     + transformer = Transformer(  
       model\_name=step\_create\_model.properties.ModelName,  
       instance\_type=config['transform\_step']['instance\_type'],  
       instance\_count=config['transform\_step']['instance\_count'],  
       strategy="MultiRecord",  
       accept="text/csv",  
       assemble\_with="Line",  
       output\_path=f"s3://{bucket}",  
       tags = config['transform\_step']['tags'],   
       env={  
        'START\_TIME\_UTC': st.strftime('%Y-%m-%d %H:%M:%S'),   
        'END\_TIME\_UTC': et.strftime('%Y-%m-%d %H:%M:%S'),  
       })
     + Now that the transformer object is created, pass the transformer input (that contains the batch data from the batch preprocess step) into the TransformStep declaration. Store the output of this pipeline in an S3 bucket.
  + step\_transform = TransformStep(  
     name=config['transform\_step']['step\_name'], transformer=transformer, inputs=transform\_input,   
    )

1. Lastly, Choose [2\_realtime\_inference.ipynb](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/2_realtime_inference.ipynb). When the notebook is open, on the Run menu, choose **Run All Cells** to run the code in this notebook. This notebook extracts the latest approved model from the Model Registry and deploys it as a SageMaker Endpoint for real-time inference. It does so by executing the following steps:
   1. **Extract the latest approved model from the SageMaker Model Registry**: To deploy a real-time SageMaker endpoint, first, fetch the image uri of your choice and extract the latest approved model from the Model Registry. Once the latest approved model is extracted, we use a container list with the specified inference.py as the script for the deployed model to use at inference. This model creation and endpoint deployment is specific to the [scikit-learn model](https://sagemaker.readthedocs.io/en/stable/frameworks/sklearn/sagemaker.sklearn.html) configuration.
   * In this code, we use the inference.py file specific to the scikit-learn model. We then create our endpoint configuration, setting our ManagedInstanceScaling to ENABLED with our desired MaxInstanceCount and MinInstanceCount for automatic scaling:
   * create\_endpoint\_config\_response = sm.create\_endpoint\_config(  
     EndpointConfigName = endpoint\_config\_name,  
     ProductionVariants=[{  
      'InstanceType': instance\_type,  
      # have max instance count configured here  
      'InitialInstanceCount': min\_instances,  
      'InitialVariantWeight': 1,  
      'ModelName': model\_name,  
      'VariantName': 'AllTraffic',   
      # change your managed instance configuration here  
      "ManagedInstanceScaling":{  
      "MaxInstanceCount": max\_instances,  
      "MinInstanceCount": min\_instances,  
      "Status": "ENABLED",}  
     }])
   1. **Run inferences on the deployed real time endpoint**: Once you have extracted the latest approved model, created the model from the desired image uri, and configured the Endpoint configuration, you can then deploy it as a real-time SageMaker endpoint.
   * create\_endpoint\_response = sm.create\_endpoint(  
     EndpointName=endpoint\_name,  
     EndpointConfigName=endpoint\_config\_name)  
       
     # wait for endpoint to reach a terminal state (InService) using describe endpoint  
     describe\_endpoint\_response = sm.describe\_endpoint(EndpointName=endpoint\_name)  
       
     while describe\_endpoint\_response["EndpointStatus"] == "Creating":  
      describe\_endpoint\_response = sm.describe\_endpoint(EndpointName=endpoint\_name)
   * Upon deployment, you can view the endpoint in service on the SageMaker Endpoints under the Inference option on the left panel: 
   1. **Now, run inference against the data extracted from PrestoDB**:
   * body\_str = "total\_extended\_price,avg\_discount,total\_quantity\n1,2,3\n66.77,12,2"  
       
     response = smr.invoke\_endpoint(  
      EndpointName=endpoint\_name,  
      Body=body\_str.encode('utf-8') ,  
      ContentType='text/csv',  
     )  
       
     response\_str = response["Body"].read().decode()  
     response\_str

## Results

Here is a compilation of some queries and responses generated by our implementation from the real time endpoint deployment stage: [ to add results here, querying data, fetching it, making predictions etc]

mlops-pipeline-prestodb results

| Query | Answer |
| --- | --- |
| total\_extended\_price,avg\_discount,total\_quantity,2,3,12,2 | – response – |

## Conclusion

We have demonstrated an end-to-end MLOps solution on SageMaker. The process involved fetching data by connecting a SageMaker Processing Job to a PrestoDB instance, followed by training, evaluating, and registering the model. We approved the latest registered model from the training pipeline and ran batch inference against it using batch data queried from PrestoDB and stored in S3. Furthermore, we deployed the latest approved model as a real-time SageMaker endpoint to run inferences.

The rise of generative AI increases the demand for training, deploying, and running machine learning models, and consequently, the use of data. By integrating SageMaker Processing Jobs with PrestoDB, you can seamlessly migrate your workloads to SageMaker pipelines without additional data preparation, storage, or accessibility burdens. You can build, train, evaluate, run batch inferences, and deploy models as real-time endpoints while leveraging your existing data engineering pipelines with minimal or no code changes.

Explore SageMaker Pipelines, open-source data querying engines like PrestoDB, and build a solution using the sample implementation provided.

Portions of this code are released under the Apache 2.0 License as referenced here: https://aws.amazon.com/apache-2-0/

## Author bio

Amit Arora is an AI and ML Specialist Architect at Amazon Web Services, helping enterprise customers use cloud-based machine learning services to rapidly scale their innovations. He is also an adjunct lecturer in the MS data science and analytics program at Georgetown University in Washington D.C.

Madhur Prashant

Antara Raisa is an AI and ML Solutions Architect at Amazon Web Services supporting Strategic Customers based out of Dallas, Texas. She also has previous experience working with large enterprise partners at AWS, where she worked as a Partner Success Solutions Architect for digital native customers.