How Twilio used Amazon SageMaker MLOps Pipelines with PrestoDB to enable frequent model re-training, optimized batch processing, and detect burner phone numbers

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

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

Machine learning (ML) models do not operate in isolation. To deliver value, they must integrate into existing production systems and infrastructure, which necessitates considering the entire ML lifecycle during design and development. ML operations, known as MLOps, focus on streamlining, automating, and monitoring ML models throughout their lifecycle. Building a robust MLOps pipeline demands cross-functional collaboration. Data scientists, ML engineers, IT staff, and DevOps teams must work together to operationalize models from research to deployment and maintenance. 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 Projects](https://docs.aws.amazon.com/sagemaker/latest/dg/sagemaker-projects.html) (CI/CD), [Amazon SageMaker Pipelines](https://aws.amazon.com/sagemaker/pipelines/) and [Amazon SageMaker Model Registry](https://docs.aws.amazon.com/sagemaker/latest/dg/model-registry.html). In this blog post, we will discuss SageMaker Pipelines and Model Registry.

**SageMaker Pipelines** allows for straightforward creation and management of ML workflows, while also offering storage and reuse capabilities for workflow steps. The **SageMaker Model Registry** 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), where 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 an MLOPs pipeline and query data as a part of this process 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:

* How you can read raw data available in PrestoDB via SageMaker Processing Jobs
* Train a binary classification model using SageMaker Training Jobs and tune the model using SageMaker Automatic Model Tuning
* Run a batch transform for inference on your raw data fetched from prestoDB and deploy the model as a real time SageMaker endpoint for inference

## Use case overview

Twilio is an american cloud communications company, based in San Francisco, California and 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 servives to run their daily workloads. This blog resolves around the steps AWS and Twilio took to migrate Twilio’s existing MLOps, implementation of training models and running batch inferences (that are able to detect burner accounts based on unusual user activity) to Amazon SageMaker.

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 built a data and machine learning operations pipeline to detect these anomaly phone numbers with the help of a binary classification model using the scikit-learn RandomForestClassifier. The training data they used for this pipeline is made available via PrestoDB tables and is read into Pandas through the [PrestoDB Python client](https://pypi.org/project/presto-python-client/). This data is then read into an Apache Spark dataframe for further analysis and machine learning operations.

The end goal was to convert all the existing steps into a three fold solution utilizing SageMaker Pipelines to enable more frequent model re-training, optimized batch processing, while customers take advatage of flexibility of data access via an open-source SQL query engine: 1/Implement a training pipeline and 2/batch inference pipeline (by connecting a sagemaker processing job with data queried from Presto). 3/Finally, we also demonstrate deploying the trained model on a SageMaker Endpoint for real-time inference.

For the proof of concept, we used the [TPCH-Connector](https://prestodb.io/docs/current/connector/tpch.html) as our choice of open source data (this allows users to test Presto’s capabilities and query syntax without needing to configure access to an external data source). Using this solution, Twilio successfully migrated to SageMaker pipelines with the open source solution that can be viewed here published on aws-samples github: [mlops-pipeline-prestodb](https://github.com/aws-samples/mlops-pipeline-prestodb?tab=readme-ov-file).

## Solution overview

The solution presented provides an implementation for training a machine learning model and running batch inference on Amazon SageMaker using data fetched from a PrestoDB table. This solution provides a design pattern built on AWS best practices that can be replicated for other ML workloads with minimal overhead. This is divided into three main steps: training pipeline, batch inference pipeline, and an implementation of real time inference support for the choice of maching learning model.

This solution is now open source and can be run through simple config file updates. For more information on the ***config.yml*** file walkthrough, view [this link](./config.yml) (add a link here poiting to the config file).

This solution includes the following steps:

* [Model Training Pipeline](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/0_model_training_pipeline.ipynb): In this step, we connect a sagemaker processing job to data fetched from a Presto server that runs on an Amazon EC2 instance, train and tune the ML model and register it with the [SageMaker model registry](https://docs.aws.amazon.com/sagemaker/latest/dg/model-registry.html). All the steps in this notebook are executed as part of a training pipeline.
* [Batch Transform Pipeline](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/1_batch_transform_pipeline.ipynb): This notebook is used to perform an automatic model approval step that changes the state of the model registered with the model registry from PendingForApproval to Approved state. This step can be removed for production ready accounts where a human in the loop or some criteria based approval would be required. After this, we launch the batch inference pipeline that reads data from PrestoDB and runs batch inference on it using the most recent Approved ML model. This model is approved either programmatically or manually via the SageMaker model registry.
* [Realtime Inference](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/2_realtime_inference.ipynb): This notebook is used to 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 consists of the following parts - Setting up the data preparation and training pipeline, preparing for the batch transform step, and deploying the approved model of choice as a real time SageMaker endpoint for inference. All of these parts utilize information from a single [config.yml file](./config.yml), which includes the necessary AWS and Presto credential information to connect to a presto server on an EC2 instance, Individial step [pipeline parameters](https://docs.aws.amazon.com/sagemaker/latest/dg/build-and-manage-parameters.html) for the data preprocessing, training, tuning, model evaluation, model registeration and real time endpoint steps of this solution. This configuration file is highly customizable for the user to use and run the solution end to end with minimal-no code changes.

The main components 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 server started on an EC2 instance, and any feature engineering needed is done as part of the SQL queries run in PrestoDB at retrieval time. The queries used to fetch data at the training and batch inference step can be configured in the [config file here](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/config.yml).
2. We use a [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 SageMaker SDK and the RandomForestClassifier from scikit-learn to train the ML model. The [HyperparameterTunerclass](https://sagemaker.readthedocs.io/en/stable/api/training/tuner.html) is used for running automatic model tuning to determine the set of hyperparameters that provide the best performance for a given use case (for example, maximize the [AUC metric](https://en.wikipedia.org/wiki/Receiver_operating_characteristic)).
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 configurable threshold and only then register the model with the model registry (from where it can be subsequently approved and deployed). If the model accuracy does not meet a configured 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 above.

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

1. The batch inference pipeline consists of two steps: a data preparation step that retrieves the data from PrestoDB (using a [batch data preprocess script](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/code/presto_preprocess_for_batch_inference.py) that connects and fetches data from the presto server deployed on EC2) and stores it in S3 (same implementation as in the training pipeline mentioned above). After this, a batch transform step runs inference on this data stored in S3 and stores the output data in S3.
2. In this step, we utilize the transformer instance and the TransformInput with the batch\_data pipeline parameter defined.

### 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 from the model registry is deployed as a realtime endpoint.
2. The latest approved model is retrieved from the 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.
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) and [automatic scaling policy](https://docs.aws.amazon.com/sagemaker/latest/dg/endpoint-auto-scaling.html) configured.

## 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) and familarity with SageMaker, S3, and PrestoDB.

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

#### PrestoDB

We will use the built-in datasets available in PrestoDB for this repo. Follow the instructions below 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)***.

1. Create a security group to limit access to Presto. Create a security group called **MyPrestoSG** with two inbound rules to only allow access to Presto.
   * Create the first rule to allow inbound traffic on port 8080 to Anywhere-IPv4
   * Create the second rule rule to allow allow inbound traffic on port 22 to your IP only.
   * You should allow all outbound traffic
2. Spin-up an EC2 instance with the following settings. This instance is used to run PrestoDB.
   * AMI: Amazon Linux 2 AMI (HVM)
   * SSD Volume Type – ami-0b1e534a4ff9019e0 (64-bit x86) / ami-0a5c7dec456e07a8d (64-bit Arm)
   * Instance type: t3a.medium
   * Subnet: Pick a public one and assign a public IP
   * IAM role: None
   * EBS: 8 GB gp2
   * Security group: MyPrestoSG
3. Install the JVM and Presto binaries.
   * Once the instance state changes to “running” and status checks are passed. Try to ssh into your EC2 instance with:
   * ssh ec2-user@{public-ip} -i {location}
   * If everything goes well, you will see the shell of your EC2 instance.
   * Install Presto 330 on the EC2 instance. Presto 330 requires the long-term support version Java 11. So let’s install it. First elevate yourself to root
   * sudo su
   * Then update yum and install Amazon Corretto 11.
   * yum update -y  
     yum install java-11-amazon-corretto.x86\_64  
     java --version
   * Now download the PrestoDB release binaries into the EC2 instance. You can download the Presto release binaries from the Maven Central Repository with wget. Then extract the archive to a directory named presto-server-330.
   * wget https://repo.maven.apache.org/maven2/io/prestosql/presto-server/330/presto-server-330.tar.gz   
       
     tar xvzf presto-server-330.tar.gz   
       
     ls -ltr presto-server-330
4. Configure Presto and add a data source. Before we start the Presto daemon, we must first provide a set of configuration files in presto-server-330/etc and add a data source. Go into presto-server-330 and create the etc directory

* cd presto-server-330  
  mkdir etc
  + Then create the three files using vim or your favorite text editor.
    - Presto logging configuration file etc/config.properties
    - coordinator=true  
      node-scheduler.include-coordinator=true  
      http-server.http.port=8080  
      query.max-memory=5GB  
      query.max-memory-per-node=1GB  
      query.max-total-memory-per-node=2GB  
      discovery-server.enabled=true  
      discovery.uri=http://localhost:8080
    - Presto node configuration: etc/node.properties
    - node.environment=demo
    - JVM configuration: etc/jvm.config
    - -server  
      -Xmx4G  
      -XX:+UseG1GC  
      -XX:G1HeapRegionSize=32M  
      -XX:+UseGCOverheadLimit  
      -XX:+ExplicitGCInvokesConcurrent  
      -XX:+HeapDumpOnOutOfMemoryError  
      -XX:+ExitOnOutOfMemoryError  
      -Djdk.nio.maxCachedBufferSize=2000000  
      -Djdk.attach.allowAttachSelf=true
    - Catalog properties file for the TPC-H connector: etc/catalog/tpch.properties
    - connector.name=tpch

1. Run the PrestoDB daemon. Use the bin/launcher script to start Presto as a foreground process. The script is in the folder **presto-server-330**

* bin/launcher run
* If you’ve set everything up right, Presto will begin printing logs to stdout and stderr. After awhile you should see this line
* INFO main io.prestosql.server.PrestoServer ======== SERVER STARTED

1. You have a running instance of PrestoDB! Since you launched PrestoDB on a public subnet and enabled 8080 inbound traffic. You can even access the UI at http://{ec2-public-ip}:8080.

#### IAM Role

The SageMaker execution role used to run this solution should have permissions to launch, list and describes various SageMaker services and artifacts. \*\*\*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.

#### 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 to it and a password field to it.

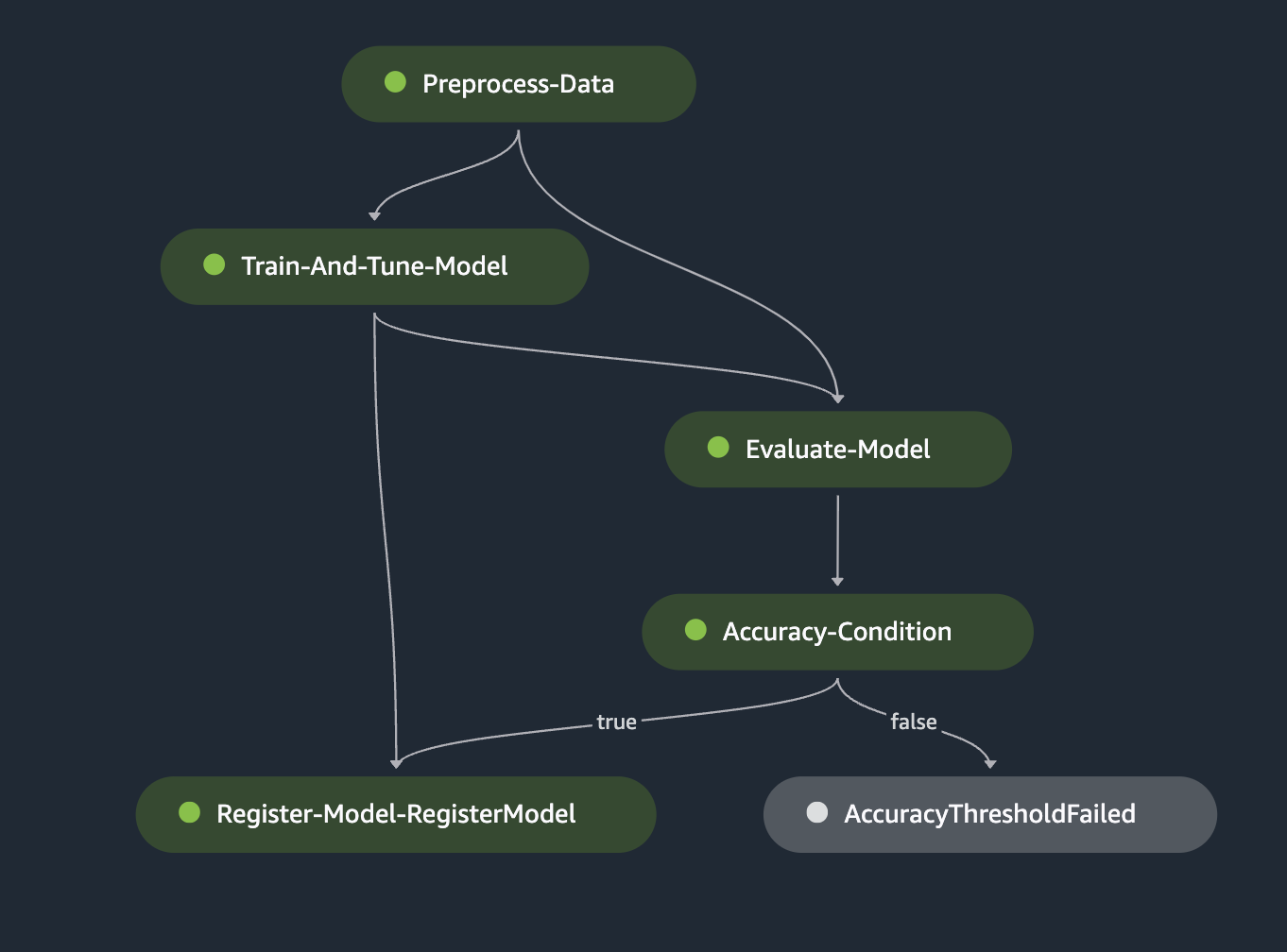
### Steps to run

1. Clone the [code repo](https://github.com/aws-samples/mlops-pipeline-prestodb.git) on SageMaker Studio.
2. Edit the [config](./config.yml) as per PrestoDB connection, IAM role and other pipeline details such as instance types for various pipeline steps etc.
   * 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 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 the parameters in the rest of the sections of the [config](./config.yml)and edit them if needed.

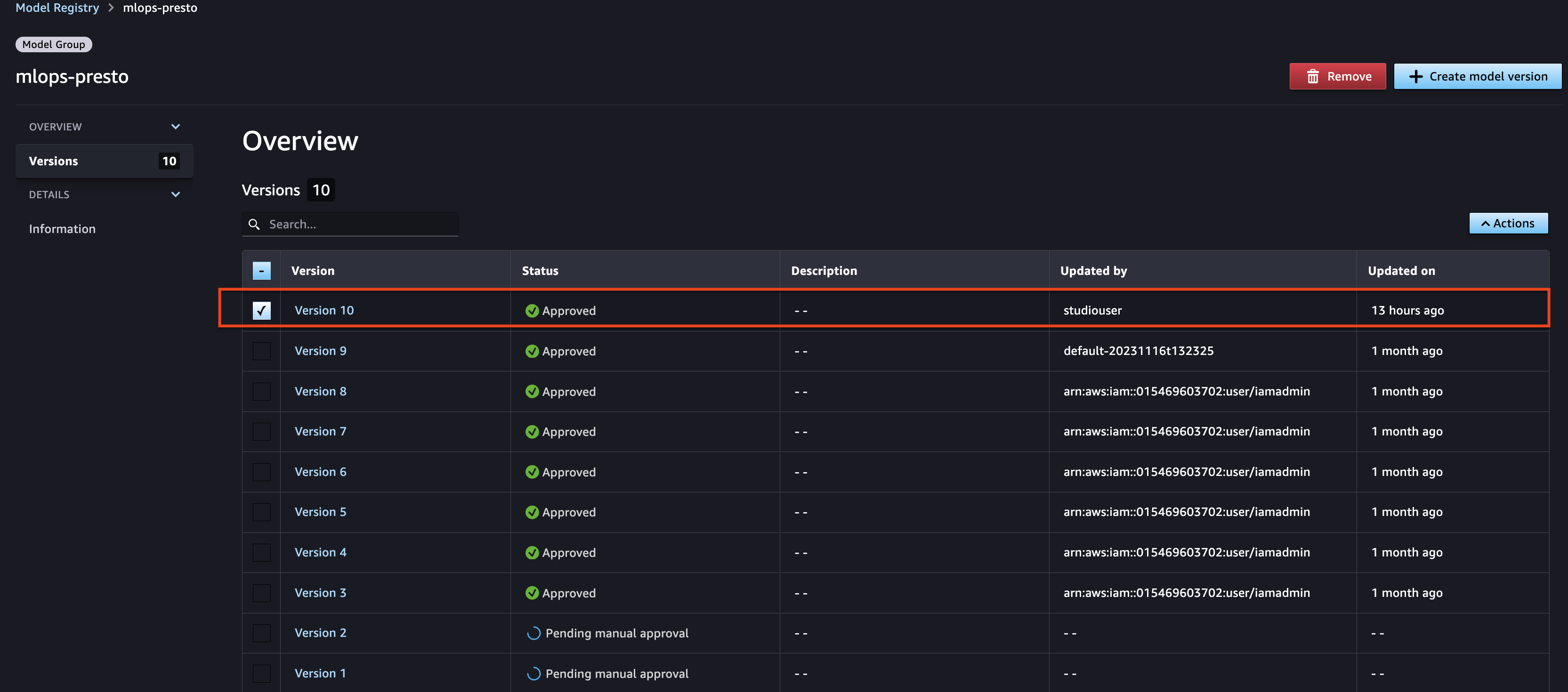
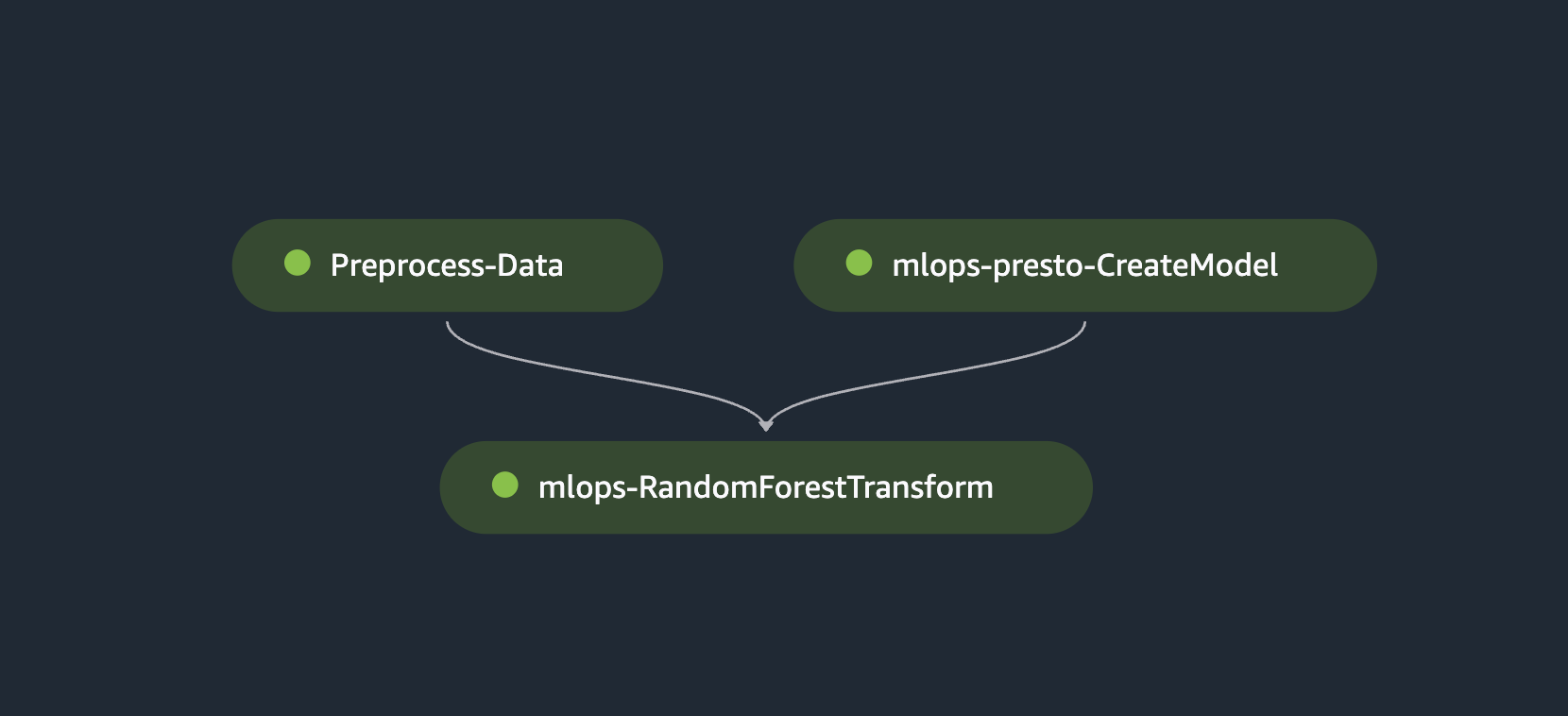
## Testing the solution

Once the prerequisites and set up is complete and the config.yml file is set up correctly, we are now ready to run the mlops-pipeline-prestodb implementation. Follow the steps below or access the [github repository](https://github.com/aws-samples/mlops-pipeline-prestodb/tree/main) to walk through the solution:

1. On the SageMaker console, or your IDE of choice, 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. The trained model can then be used for batch inference, or hosted on a SageMaker endpoint for realtime inference.
   * **Preprocess data step**: In this step of the notebook, we set our pipeline input parameters when triggering our pipeline execution. We use a [preprocess script](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/code/presto_preprocess_for_training.py) which is read to connect to the presto server on our EC2 instance, and query data (using the query specified and configurable in the [config file](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/config.yml)), that is then sent to an S3 bucket split into train, test and validation datasets. Using the data in these files, we can train our machine learning model.
     + We use the [sklearn\_processor](https://docs.aws.amazon.com/sagemaker/latest/dg/use-scikit-learn-processing-container.html) in a SageMaker Pipelines ProcessingStep and define it as given below:
   * # 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 presto server  
      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,  
      )
     + We are using the config['scripts']['source\_dir'] which refers to our data preprocessing script that connects to the EC2 instance where the presto server runs. This script is responsible for extracting data from the query that you define. You can query the data you want by modifying the query parameter in the ***config.yml*** file query parameter. We are using the sample query as an example to extract open source TPCH data on orders, discounts and order priorities.
   * 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,  
      o.orderdate,  
      o.orderpriority,  
      CASE  
      WHEN SUM(l.extendedprice) > 20000 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
   * **Train Model Step**: In this step of the notebook, we use the [SKLearn estimator](https://sagemaker.readthedocs.io/en/stable/frameworks/sklearn/sagemaker.sklearn.html) from SageMaker SDK and the RandomForestClassifier from scikit-learn to train the ML model. The [HyperparameterTunerclass](https://sagemaker.readthedocs.io/en/stable/api/training/tuner.html) is used for running automatic model tuning to determine the set of hyperparameters that provide the best performance based on a given metric threshold (maximize the AUC metric).
   * In the code below, the sklearn\_estimator object is created with parameters that are configured in the [config file](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/config.yml) and uses this [training script](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/code/training.py) to train the ML model. The hyperparameters are also configurable by the user via the config file. This step accesses the train, test and validation files that are created as a part of the previous data preprocessing step.
   * sklearn\_estimator = SKLearn(  
      # we configure the training script that accesses the train, test and validation files from the data preprocessing step  
      entry\_point=config['scripts']['training\_script'],  
      role=role,  
      instance\_count=config['training\_step']['instance\_count'],  
      instance\_type=config['training\_step']['instance\_type'],  
      framework\_version=config['training\_step']['sklearn\_framework\_version'],  
      base\_job\_name=config['training\_step']['base\_job\_name'],  
      hyperparameters={  
      # Hyperparameters are fetched and are configured in the config.yml file  
      "n\_estimators": config['training\_step']['n\_estimators'],  
      "max\_depth": config['training\_step']['max\_depth'],   
      "features": config['training\_step']['training\_features'],  
      "target": config['training\_step']['training\_target'],  
      },  
      tags=config['training\_step']['tags']  
     )  
     # Create Hyperparameter tuner object. Ranges from https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost-tuning.html  
     rf\_tuner = HyperparameterTuner(  
      estimator=sklearn\_estimator,  
      objective\_metric\_name=config['tuning\_step']['objective\_metric\_name'],  
      hyperparameter\_ranges={  
      "n\_estimators": IntegerParameter(config['tuning\_step']['hyperparam\_ranges']['n\_estimators'][0], config['tuning\_step']['hyperparam\_ranges']['n\_estimators'][1]),  
      "max\_depth": IntegerParameter(config['tuning\_step']['hyperparam\_ranges']['max\_depth'][0], config['tuning\_step']['hyperparam\_ranges']['max\_depth'][1]),  
      "min\_samples\_split": IntegerParameter(config['tuning\_step']['hyperparam\_ranges']['min\_samples\_split'][0], config['tuning\_step']['hyperparam\_ranges']['min\_samples\_split'][1]),  
      "max\_features": CategoricalParameter(config['tuning\_step']['hyperparam\_ranges']['max\_features'])  
      },  
      max\_jobs=config['tuning\_step']['maximum\_training\_jobs'], ## reducing this for testing purposes  
      metric\_definitions=config['tuning\_step']['metric\_definitions'],  
      max\_parallel\_jobs=config['tuning\_step']['maximum\_parallel\_training\_jobs'], ## reducing this for testing purposes  
     )  
       
     # 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",  
      ),  
      },  
     )
   * **Evaluate model step**: The purpose of this step is to check if the trained and tuned model has an accuracy level above a configurable threshold and only then register the model with the model registry (from where it can be subsequently [approved and deployed](https://docs.aws.amazon.com/sagemaker/latest/dg/model-registry-approve.html)). If the model accuracy does not meet a configured threshold 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.
     + Once this step is run, an Evaluation Report is generated that is sent to the S3 bucket for analysis:
   * evaluation\_report = PropertyFile(  
      name="EvaluationReport", output\_name="evaluation", path=config['evaluation\_step']['evaluation\_filename']  
     )
     + The evaluation step uses the [evaluation script](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/code/evaluate.py) as a code entry in the step below. This script prepares the features and target values and calculates teh prediciton probabilities using model.predict. The evaluation report sent to S3 contains information on metrics like precision, recall, accuracy.
   * 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,  
      ]  
     )
   * **Register model step**: 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) for further analysis and model creation.

* # Crete a RegisterModel step, which registers the model with SageMaker Model Registry.  
  step\_register\_model = RegisterModel(  
   name=config['register\_model\_step']['step\_name'],  
   estimator=sklearn\_estimator,  
   model\_data=step\_tuning.get\_top\_model\_s3\_uri(top\_k=0, s3\_bucket=bucket),  
   content\_types=["text/csv"],  
   response\_types=["text/csv"],  
   inference\_instances=config['register\_model\_step']['inference\_instance\_types'],  
   transform\_instances=config['register\_model\_step']['transform\_instance\_types'],  
   model\_package\_group\_name=model\_group,  
   approval\_status=model\_approval\_status,  
   model\_metrics=model\_metrics,  
   tags=config['register\_model\_step']['tags']  
   )
* ***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.***
* Adding conditions to the pipeline is done with a [ConditionStep](https://sagemaker.readthedocs.io/en/stable/workflows/pipelines/sagemaker.workflow.pipelines.html). In this case, we only want to register the new model version with the model registry if the new model meets a specific accuracy condition:
* step\_fail = FailStep(  
   name=config['fail\_step']['step\_name'],  
   error\_message=Join(on=" ", values=["Execution failed due to Accuracy <", accuracy\_condition\_threshold]),  
  )  
    
  # Create accuracy condition to ensure the model meets performance requirements.  
  # Models with a test accuracy lower than the condition will not be registered with the model registry.  
  cond\_gte = ConditionGreaterThanOrEqualTo(  
   left=JsonGet(  
   step\_name=step\_evaluate\_model.name,  
   property\_file=evaluation\_report,  
   json\_path="binary\_classification\_metrics.accuracy.value",  
   ),  
   right=accuracy\_condition\_threshold,  
  )  
    
  # 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.
  + **Orchestrate all steps and start the pipeline**: Once you have created the pipeline steps as above, you can instantiate and start it with custom parameters making the pipeline agnostic to who is triggering it, but also to the scripts and data used. The pipeline can be started using the CLI, the SageMaker Studio UI or the SDK.
  + # Start pipeline with credit data and preprocessing script  
    execution = pipeline.start(  
     execution\_display\_name=pipeline.name,  
     parameters=dict(  
     AccuracyConditionThreshold=config['evaluation\_step']['accuracy\_condition\_threshold'],  
     MaximumParallelTrainingJobs=config['tuning\_step']['maximum\_parallel\_training\_jobs'],  
     MaximumTrainingJobs=config['tuning\_step']['maximum\_training\_jobs'],  
     ModelGroup=config['register\_model\_step']['model\_group'],  
     ),  
     )  
      
    # View further information on the state of the execution  
    execution.describe()  
      
    # wait for the execution process to complete  
    execution.wait()  
      
    # print the summary of the pipeline run once it is completed  
    print\_pipeline\_execution\_summary(execution.list\_steps(), pipeline.name)
* **At the end of the 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:** 
* ***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 step using the model trained in the previous notebook. It does so by running the following steps:
   * **Extract the latest approved model from the SageMaker model registry**: In this step, we first define pipeline input parameters that are used for the ***EC2 instance types to use for processing and training steps***. These parameters can be configured on the [config.yml](./config.yml) file.

* # What instance type to use for processing.  
  processing\_instance\_type = ParameterString(  
   name="ProcessingInstanceType", default\_value=config['data\_processing\_step']['processing\_instance\_type']  
  )  
    
    
  # Create SKlearn processor object,  
  # The object contains information about what instance type to use, the IAM role to use etc.  
  # A managed processor comes with a preconfigured container, so only specifying version is required.  
    
  est\_cls = sagemaker.sklearn.estimator.SKLearn  
    
  sklearn\_processor = FrameworkProcessor(  
   estimator\_cls=est\_cls,  
   framework\_version=config['training\_step']['sklearn\_framework\_version'],  
   role=role,  
   instance\_type=processing\_instance\_type,  
   instance\_count=config['data\_processing\_step']['instance\_count'],  
   tags=config['data\_processing\_step']['tags'],   
   sagemaker\_session=pipeline\_session,  
   base\_job\_name=config['pipeline']['base\_job\_name'], )
* Now, we use an image URI that we create to extract the latest model that was approved from the model registry, and set the ModelApprovalStatus to Approved:
* # list all model packages and select the latest one  
  model\_packages = []  
    
  for p in sm.get\_paginator('list\_model\_packages').paginate(  
   ModelPackageGroupName=config['register\_model\_step']['model\_group'],  
   SortBy="CreationTime",  
   SortOrder="Descending",  
   ):  
   model\_packages.extend(p["ModelPackageSummaryList"])  
    
  if len(model\_packages) == 0:  
   raise Exception(f"No model package is found for {config['register\_model\_step']['model\_group']} model package group")  
    
  ## print the latest model, approve it  
  latest\_model\_package\_arn = model\_packages[0]["ModelPackageArn"]  
    
  ## 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 on SageMaker Studio as given in the screenshot below. 
  + **Read raw data for inference from PrestoDB and store in an Amazon S3 bucket**: Once the latest model is approved, we get the latest batch data from presto and use that for our batch transform step. In this step, we use another [batch preprocess script](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/code/presto_preprocess_for_batch_inference.py) that is dedicated to reading and fetching data from presto and saving in a batch directory within our S3 bucket.
* ## represents the output processing for the batch pre processing step  
  batch\_output=[  
   ProcessingOutput(  
   output\_name="batch",  
   source="/opt/ml/processing/batch",  
   destination=Join(  
   on="/",  
   values=[  
   "s3://{}".format(bucket),  
   prefix,  
   ExecutionVariables.PIPELINE\_EXECUTION\_ID,  
   "batch",  
   ],   
   ),  
   ),  
   ]  
    
    
  # Use the sklearn\_processor's run method and configure the batch preprocessing step  
  step\_args = sklearn\_processor.run(  
   # here, we add in a `code` or an entry point that uses the data preprocess script for collecting data in a batch and storing it in S3  
   code=config['scripts']['batch\_transform\_get\_data'],  
   source\_dir=config['scripts']['source\_dir'],   
   outputs=batch\_output,  
   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,  
   ],  
  )  
    
  # 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 preparation step is complete, we declare a model with the image uri and refer to the [‘inference.py’](https://github.com/aws-samples/mlops-pipeline-prestodb/blob/main/code/inference.py) script that grabs information on features to use while making predictions. Using this, we will create the model which automatically trigger the training and the preprocess data step Run the transformer step on the created 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,  
  )
  + **Create a batch transform step to provide inference on the data. The inference results are also stored in S3**: Now that a model instance is defined, create a Transformer instance with the appropriate model type, compute instance type, and desired output S3 URI. Specifically, pass in the ModelName from the CreateModelStep, step\_create\_model properties. The CreateModelStep properties attribute matches the object model of the DescribeModel response object.
* 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 our transformer object is created, we pass the transformer input (that contains the batch data from our batch pre process step) into the step declaration:
* step\_transform = TransformStep(  
   name=config['transform\_step']['step\_name'], transformer=transformer, inputs=transform\_input,   
  )  
    
  batch\_transform\_pipeline = Pipeline(  
  name=pipeline\_name,  
  parameters=[processing\_instance\_type,  
  host\_parameter,  
  presto\_parameter,  
  region\_parameter,  
  port\_parameter,  
  target\_parameter,   
  feature\_parameter,  
  presto\_catalog\_parameter,  
  presto\_schema\_parameter,],  
  steps=[  
   batch\_data\_prep,  
   step\_create\_model,   
   step\_transform,  
  ],  
  )  
    
  # start the pipeline execution:  
  response = sagemaker\_client.start\_pipeline\_execution(  
  PipelineName=batch\_transform\_pipeline.name  
  )  
    
  while True:  
   resp = client.describe\_pipeline\_execution(  
   PipelineExecutionArn=response['PipelineExecutionArn']  
   )  
   status = resp['PipelineExecutionStatus']
* **At the end of the batch transform pipeline, your pipeline structure on Amazon SageMaker Pipelines should look like this:** 

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 running the following steps:
   * **Extract the latest approved model from the SageMaker model registry**: To deploy a real time SageMaker endpoint, we first will fetch the image uri to use and extract the latest approved model the same way we did in the prior batch transfrom notebook. Once you have extracted the latest approved model, use a container list with the specific inference.py file to create the model and run inferences against. 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 and will change based on your use case.

* container\_list = [{  
   'Image': image\_uri,  
   'ModelDataUrl': model\_data\_url,  
   'Environment': {  
   'SAGEMAKER\_PROGRAM': 'inference.py',   
   'SAGEMAKER\_SUBMIT\_DIRECTORY': compressed\_inference\_script\_uri,   
   }  
  }]  
    
  ## create the model object and call deploy on it  
  create\_model\_response = sm.create\_model(  
   ModelName = model\_name,  
   ExecutionRoleArn = role,  
   Containers=container\_list  
  )
* 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",}  
  }])
  + **Runs inferences for testing the real time deployed 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 below:
* create\_endpoint\_response = sm.create\_endpoint(  
  EndpointName=endpoint\_name,  
  EndpointConfigName=endpoint\_config\_name)  
  logger.info(f"Going to deploy the real time endpoint -> {create\_endpoint\_response['EndpointArn']}")  
    
  # 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)  
   print(describe\_endpoint\_response["EndpointStatus"])  
   time.sleep(15)
* 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
* We have now seen an end to end process of our solution, from fetching data by connecting to a Presto Server on an EC2 instance, followed by training, evaluating, registering the model. We then approved the latest registered model from our training pipeline solution and ran batch inference against batch data stored in S3. We finally deployed the latest approved model as a real time SageMaker endpoint to run inferences against. Take a look at the results below.

## Results

Here is a compilation of some queries and responses generated by our implementation from the real time endpoint deployment stage:

mlops-pipeline-prestodb results

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

## Cleanup

–> need to perform and add a clean up section

## Conclusion

With the rise of generative AI, the use of training, deploying and running machine learning models exponentially increases, and so does the use of data. With an integration of SageMaker Processing Jobs with PrestoDB, customers can easily and seamlessly migrate their workloads to SageMaker pipelines without any burden of additional data preparation and storage. Customers can now build, train, evaluate, run batch inferences and deploy their models as real time endpoints while taking advantage of their existing data engineering pipelines with minimal-no code changes.

We encourage you to learn more by exploring SageMaker Pipeline, Open source data querying engines like PrestoDB and building a solution using the sample implementation provided in this post.

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.