Option 2: Batch Inference Pipeline (SageMaker Pipelines)

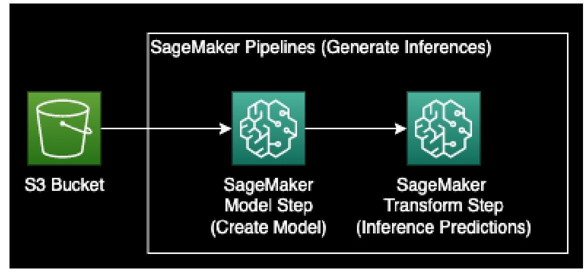
⚠ Make sure you have performed the steps described in the **Prerequisites** section before beginning this lab.

Overview

Amazon SageMaker Pipelines, is a capability of Amazon SageMaker that makes it easy for data scientists and engineers to build, automate, and scale end to end machine learning pipelines. SageMaker Pipelines is a native workflow orchestration tool for building ML pipelines that take advantage of direct Amazon SageMaker integration.

This lab focuses on integration of offline hosting or batch transformation capability of with SageMaker Pipelines to generate predictions using a customer churn classification example. This pipeline includes steps to create model in SageMaker and generate inferences using SageMaker Batch Transform.

The following diagram illustrates the high-level architecture of the ML workflow with the different steps to generate inferences using the trained model artifacts.



Inference Pipeline consists of the following steps:

1. Create a model in SageMaker using the latest approved model from SageMaker Model Registry.
2. Generate Inferences using the trained model artifacts.

Prerequisites

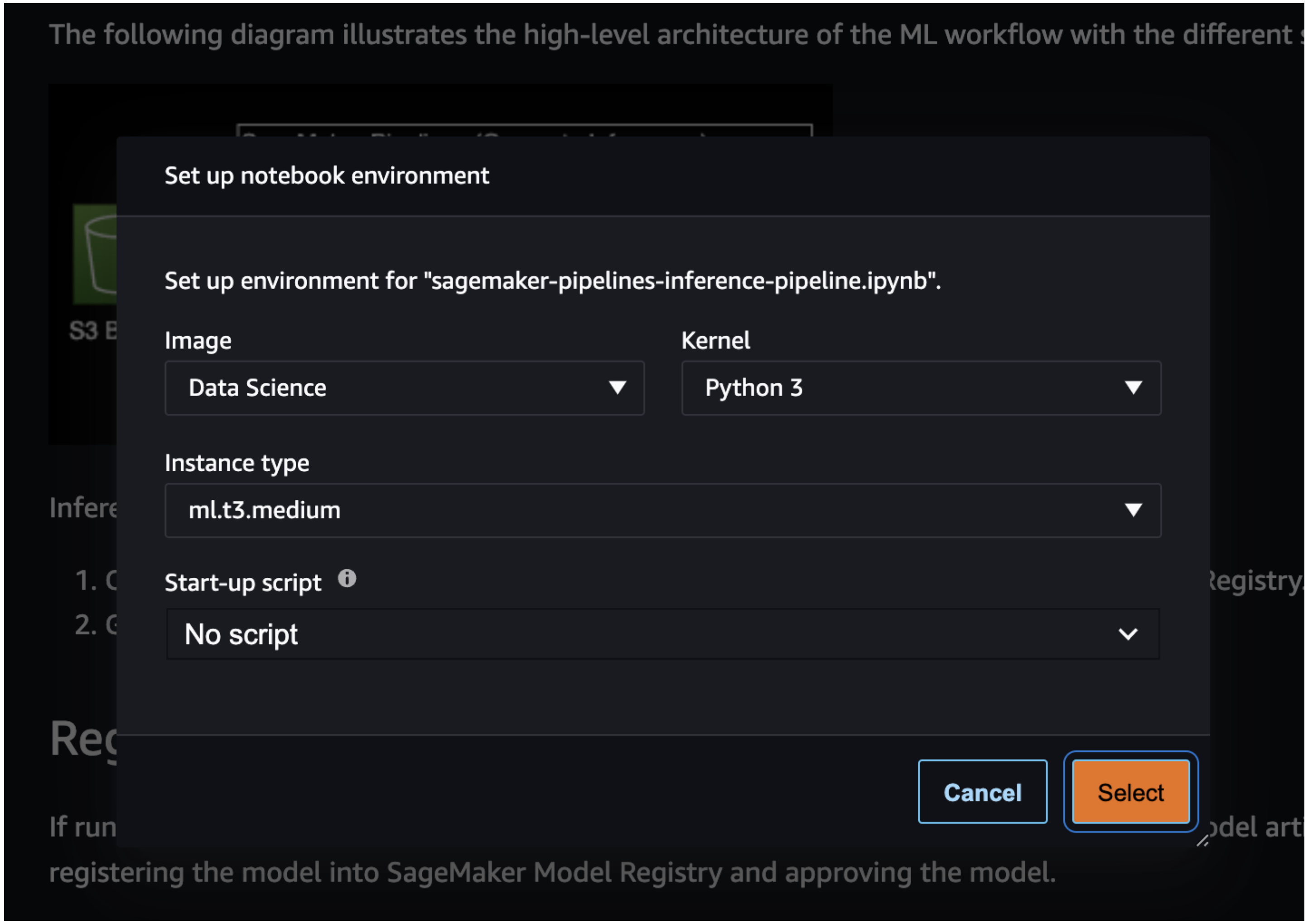
- Follow the instructions to launch Amazon SageMaker Studio
- Please ensure that you have git cloned the repository in your SageMaker Studio environment.

Register model into SageMaker Model Registry

If running through this lab independently, go through the optional step of uploading the model artifact customer-retention-model.tar.gz into S3 Bucket, registering the model into SageMaker Model Registry and approving the model.

Click on the “amazon-sagemaker-immersion-day” folder and then double click on the **sagemaker-pipelines-inference-pipeline.ipynb** notebook within the project home directory.

If you are prompted to choose a Kernel, choose the “Python 3 (Data Science)” kernel and click “Select”.



Upload Model Artifact to S3 Bucket

Run the first cell within the notebook to set up SageMaker and S3 client objects and set up the S3 bucket location using the default bucket that comes with a SageMaker session:

Upload Model Artifact to S3 Bucket

```
[2]: import boto3
import sagemaker

sagemaker_session = sagemaker.session.Session()
default_bucket = sagemaker_session.default_bucket()
s3_client = boto3.resource('s3')
s3_client.Bucket(default_bucket).upload_file("model/customer-retention-model.tar.gz","churn/model_artifacts/customer-retention-model.tar.gz")
```

Create Model Group

Run the next cell which within SageMaker Model Registry, creates a model group that will track a group of versioned models.

Create Model Group

```
[3]: import time
import os
```



```
region = sagemaker_session.boto_region_name
sm_client = boto3.client('sagemaker', region_name=region)
model_package_group_input_dict = {
 "ModelPackageGroupName" : model_package_group_name,
}

create_model_package_group_response = sm_client.create_model_package_group(**model_package_group_input_dict)
print('ModelPackageGroup Arn : {}'.format(create_model_package_group_response['ModelPackageGroupArn']))
```

Register Model in Model Registry

In the next cell, you can register a model into SageMaker Model Registry, by specifying the containers, model artifact and the associated environment variables.

Register Model in Model Registry

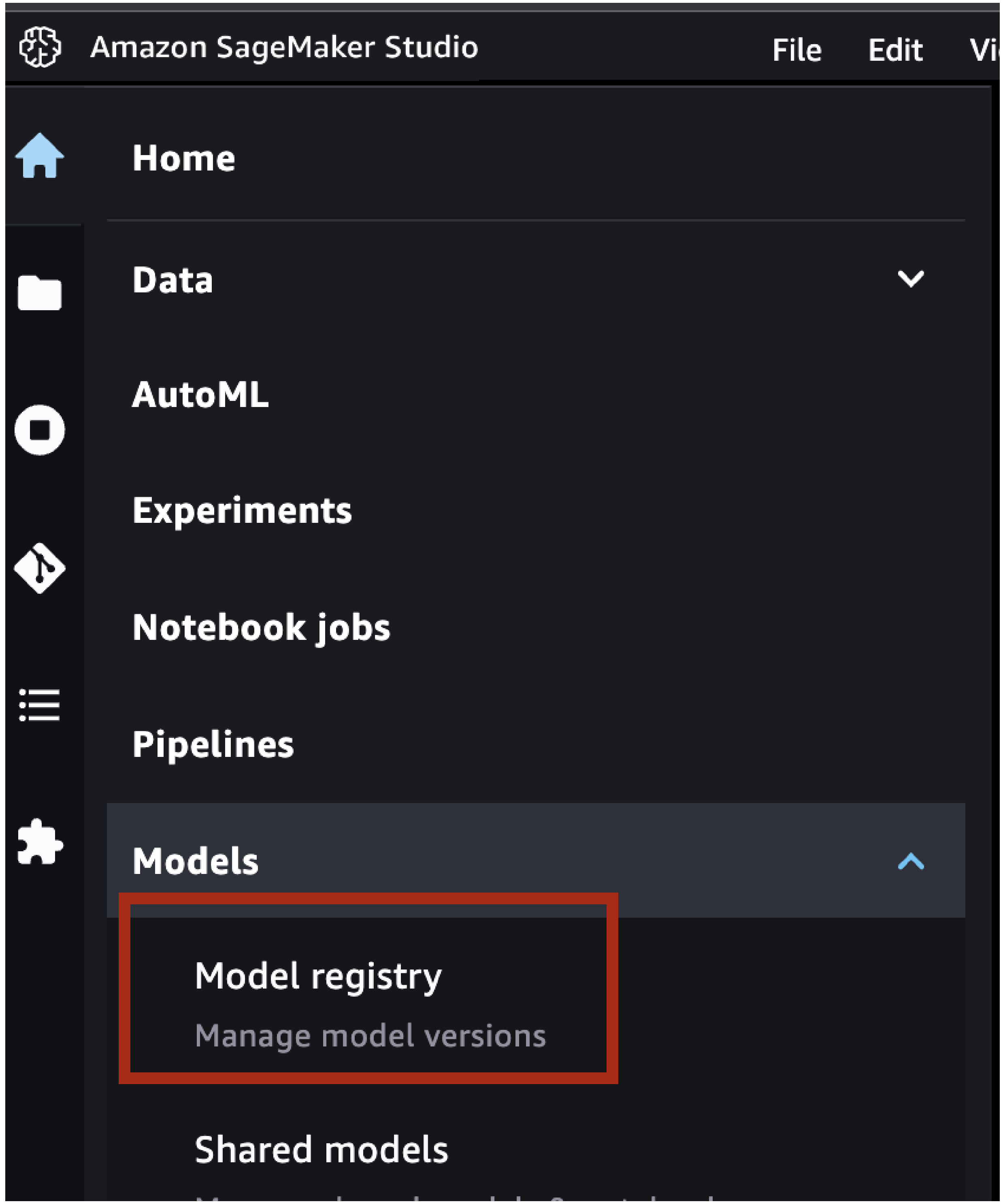
```python
[4]: # retrieve the image uri used to train model
image_uri = sagemaker.image_uris.retrieve(
    framework="xgboost",
    region=region,
    version="1.0-1",
    py_version="py3"
)

# Specify the model source
model_url = f"s3://{default_bucket}/churn/model_artifacts/customer-retention-model.tar.gz"

modelpackage_inference_specification =  {
    "InferenceSpecification": {
      "Containers": [
         {
             "Image": image_uri,
             "ModelDataUrl": model_url
```



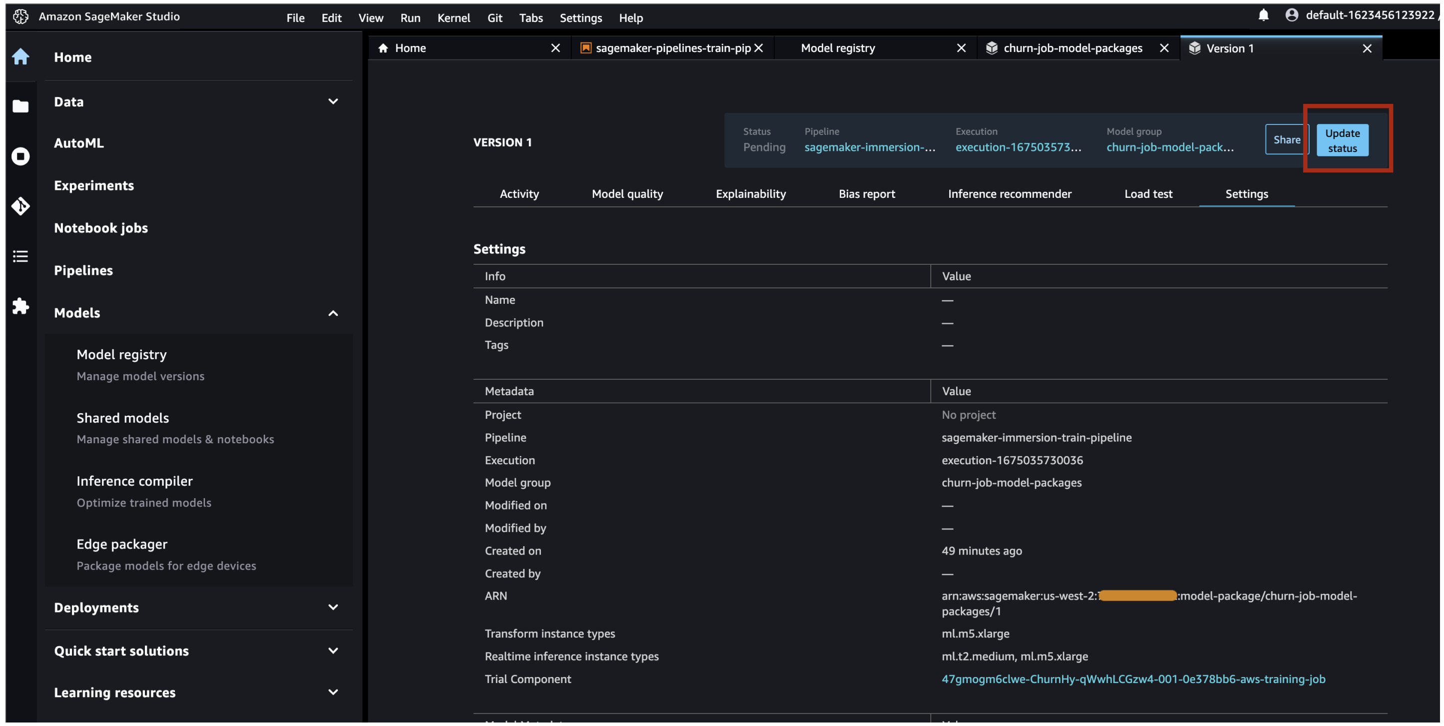
```python
      "SupportedContentTypes": [ "text/csv" ],
      "SupportedResponseMIMETypes": [ "text/csv" ],
   }
 }

# Alternatively, you can specify the model source like this:
# modelpackage_inference_specification["InferenceSpecification"]["Containers"][0]["ModelDataUrl"]=model_url

create_model_package_input_dict = {
    "ModelPackageGroupName" : model_package_group_name,
    "ModelPackageDescription" : "Model to detect 3 different types of irises (Setosa, Versicolour, and Virginica)",
    "ModelApprovalStatus" : "PendingManualApproval"
}
create_model_package_input_dict.update(modelpackage_inference_specification)

create_model_package_response = sm_client.create_model_package(**create_model_package_input_dict)
model_package_arn = create_model_package_response["ModelPackageArn"]
print('ModelPackage Version ARN : {}'.format(model_package_arn))
```

Approve Model in Model Registry

The model registered within model registry can be checked by going to the home screen and choosing the *Models → Model Registry.*


Amazon SageMaker Studio
File
Edit
Home
Data
AutoML
Experiments
Notebook jobs
Pipelines
Models
Model registry
Manage model versions
Shared models

you can click on the Update Status tab and manually approve the model.



Build the pipeline components

Import statements and declare parameters and constants

Run the next cell to set up SageMaker and S3 client objects, create PipelineSession, and set up the S3 bucket location using the default bucket that comes with a SageMaker session:

Step 1: Import statements and declare parameters and constants

```
[5]: import boto3
import pandas as pd
import sagemaker
from sagemaker.workflow.pipeline_context import PipelineSession

s3_client = boto3.resource('s3')
pipeline_name = f"sagemaker-immersion-inference-pipeline"
sagemaker_session = sagemaker.session.Session()
region = sagemaker_session.boto_region_name
role = sagemaker.get_execution_role()
pipeline_session = PipelineSession()
default_bucket = sagemaker_session.default_bucket()
model_package_group_name = f"ChurnModelPackageGroup"
```

Pipelines supports parameterization, which allows you to specify input parameters at runtime without changing your pipeline code. You can use the modules available under the sagemaker.workflow.parameters module, such as ParameterInteger, ParameterFloat, and ParameterString, to specify pipeline parameters of various data types. Run the following code to set up multiple input parameters:

```
[6]: from sagemaker.workflow.parameters import (
 ParameterInteger,
 ParameterString,
 ParameterFloat)

base_job_prefix = "churn-example"
processing_instance_count = ParameterInteger(name="ProcessingInstanceCount", default_value=1)
processing_instance_type = ParameterString( name="ProcessingInstanceType", default_value="ml.m5.xlarge")
transform_instance_type = ParameterString(name="TransformInstanceType", default_value="ml.m5.xlarge")
transform_instance_count = ParameterInteger(name="TransformInstanceCount", default_value=1)
batch_data_path = "s3://{}/data/batch/batch.csv".format(default_bucket)
model_approval_status = ParameterString( name="ModelApprovalStatus", default_value="PendingManualApproval")
```

Generate Data for Inferences

Run the next cell to generate the batch dataset, which you use later in the batch transform step. For this, download and save the sample dataset if not already done into the project folder. Sample on the original dataset to generate the data although in the real world scenario this would be a new dataset which the model has not been trained on before.

```
[9]: def preprocess_batch_data(file_path):
         df = pd.read_csv(file_path)
         ## Convert to datetime columns
         df["firstorder"]=pd.to_datetime(df["firstorder"],errors='coerce')
         df["lastorder"] = pd.to_datetime(df["lastorder"],errors='coerce')
         ## Drop Rows with null values
         df = df.dropna()
         ## Create Column which gives the days between the last order and the first order
         df["first_last_days_diff"] = (df['lastorder']-df['firstorder']).dt.days
         ## Create Column which gives the days between when the customer record was created and the first order
         df['created'] = pd.to_datetime(df['created'])
         df['created_first_days_diff']=(df['created']-df['firstorder']).dt.days
         ## Drop Columns
         df.drop(['custid','created','firstorder','lastorder'],axis=1,inplace=True)
         ## Apply one hot encoding on favday and city columns
         df = pd.get_dummies(df,prefix=['favday','city'],columns=['favday','city'])
         return df

     # convert the store_data file into csv format
     store_data = pd.read_excel("storedata_total.xlsx")
     store_data.to_csv("storedata_total.csv")
```

```
[10]: # preprocess batch data and save into the data folder
      batch_data = preprocess_batch_data("storedata_total.csv")
      batch_data.pop("retained")
      batch_sample = batch_data.sample(frac=0.2)
      pd.DataFrame(batch_sample).to_csv("batch.csv",header=False,index=False)
```

Upload Inferences Data to S3 Bucket

Upload the dataset to Amazon S3.

Step 3: Upload Inferences Data to S3 Bucket

```
[11]: s3_client.Bucket(default_bucket).upload_file("batch.csv","data/batch/batch.csv")
```

Info about the Trained Model

Retrieve the information of the model approved in SageMaker Model Registry using which the inferences will be generated.

```
[13]: sm_client = boto3.client("sagemaker")

# get a list of approved model packages from the model package group you specified earlier
approved_model_packages = sm_client.list_model_packages(
      ModelApprovalStatus='Approved',
      ModelPackageGroupName=model_package_group_name,
      SortBy='CreationTime',
      SortOrder='Descending'
  )

# find the latest approved model package
try:
    latest_approved_model_package_arn = approved_model_packages['ModelPackageSummaryList'][0]['ModelPackageArn']
except Exception as e:
    print("Failed to retrieve an approved model package:", e)

print(latest_approved_model_package_arn)

 # retrieve required information about the model
latest_approved_model_package_descr =  sm_client.describe_model_package(ModelPackageName = latest_approved_model_package_arn)

# model artifact uri (tar.gz file)
model_artifact_uri = latest_approved_model_package_descr['InferenceSpecification']['Containers'][0]['ModelDataUrl']
# sagemaker image in ecr
image_uri = latest_approved_model_package_descr['InferenceSpecification']['Containers'][0]['Image']
```

Define create model step

Run the next cell to create a SageMaker model using the Pipelines model step. This step utilizes the approved model from SageMaker Model Registry.

```
[15]: from sagemaker import Model
from sagemaker.inputs import CreateModelInput
from sagemaker.workflow.model_step import ModelStep

model = Model(
image_uri=image_uri,
model_data=model_artifact_uri,
sagemaker_session=pipeline_session,
role=role
)

step_create_model = ModelStep(
name="ChurnCreateModel",
step_args=model.create(instance_type="ml.m5.large", accelerator_type="ml.eia1.medium"),
)
```

Define Transform Step to Perform Batch Transformation

Run the next cell to run batch transformation using the trained model with the batch input created in the first step:

```
[16]: from sagemaker.transformer import Transformer
from sagemaker.inputs import TransformInput
from sagemaker.workflow.steps import TransformStep

transformer = Transformer(
    model_name=step_create_model.properties.ModelName,
    instance_type="ml.m5.xlarge",
    instance_count=1,
    output_path=f"s3://{default_bucket}/ChurnTransform",
    sagemaker_session=pipeline_session
)

step_transform = TransformStep(
    name="ChurnTransform",
    step_args=transformer.transform(
                    data=batch_data_path,
                    content_type="text/csv"
                 )
)
```

Build and Trigger the pipeline run

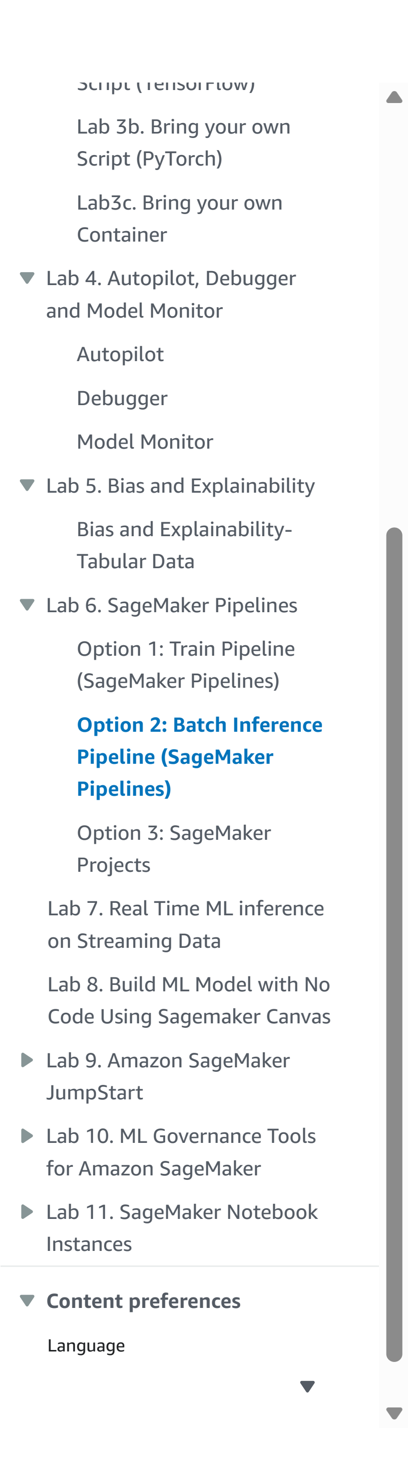
After defining all of the component steps, you can assemble them into a Pipelines object. You don’t need to specify the order of pipeline because Pipelines automatically infers the order sequence based on the dependencies between the steps.

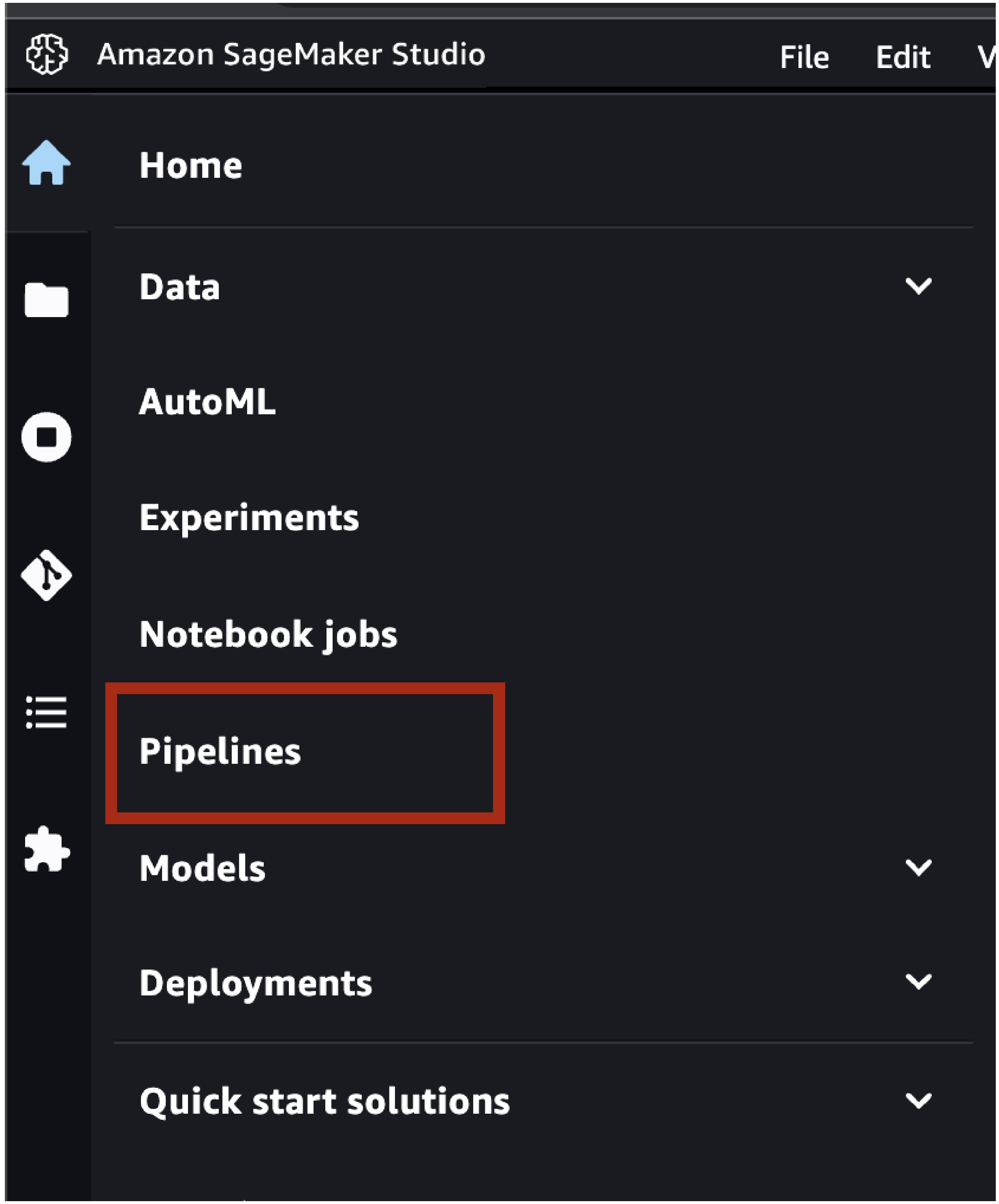
```
[17]: from sagemaker.workflow.pipeline import Pipeline

pipeline = Pipeline(
    name=pipeline_name,
    parameters=[
        processing_instance_type,
        processing_instance_count,
        transform_instance_type,
        transform_instance_count,
        batch_data,
    ],
    steps=[step_create_model,step_transform],
)
```

Run the following code in the next cell in your notebook. If the pipeline already exists, the code updates the pipeline. If the pipeline doesn't exist, it creates a new one.

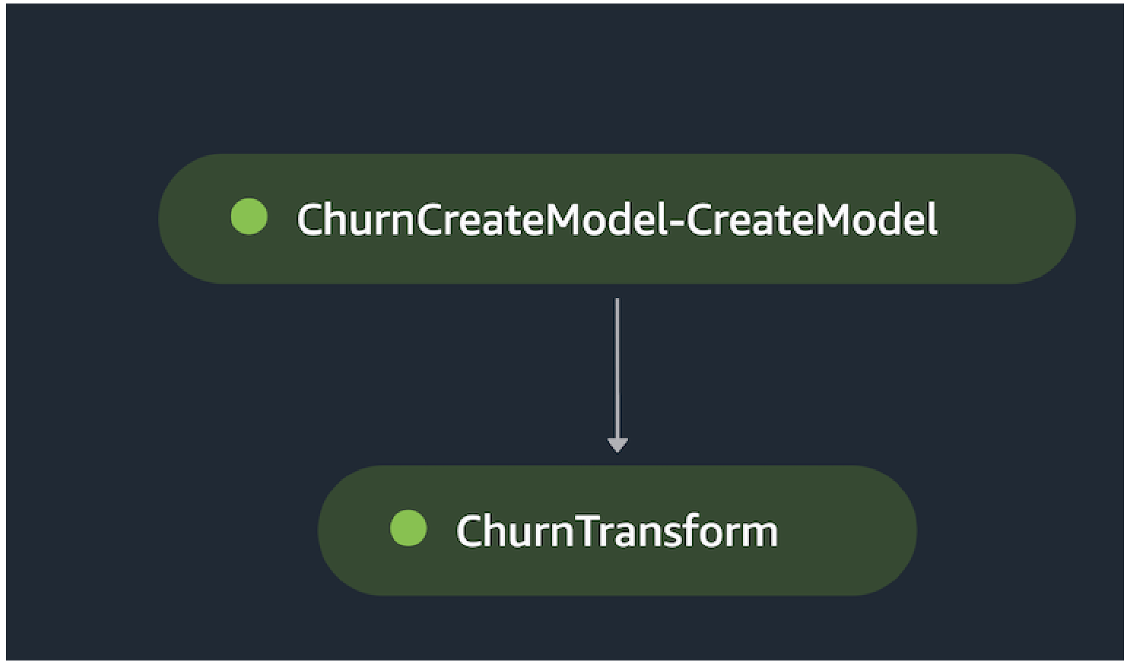
```
[18]: # Create a new or update existing Pipeline
pipeline.upsert(role_arn=role)
# start Pipeline execution
pipeline.start()
```

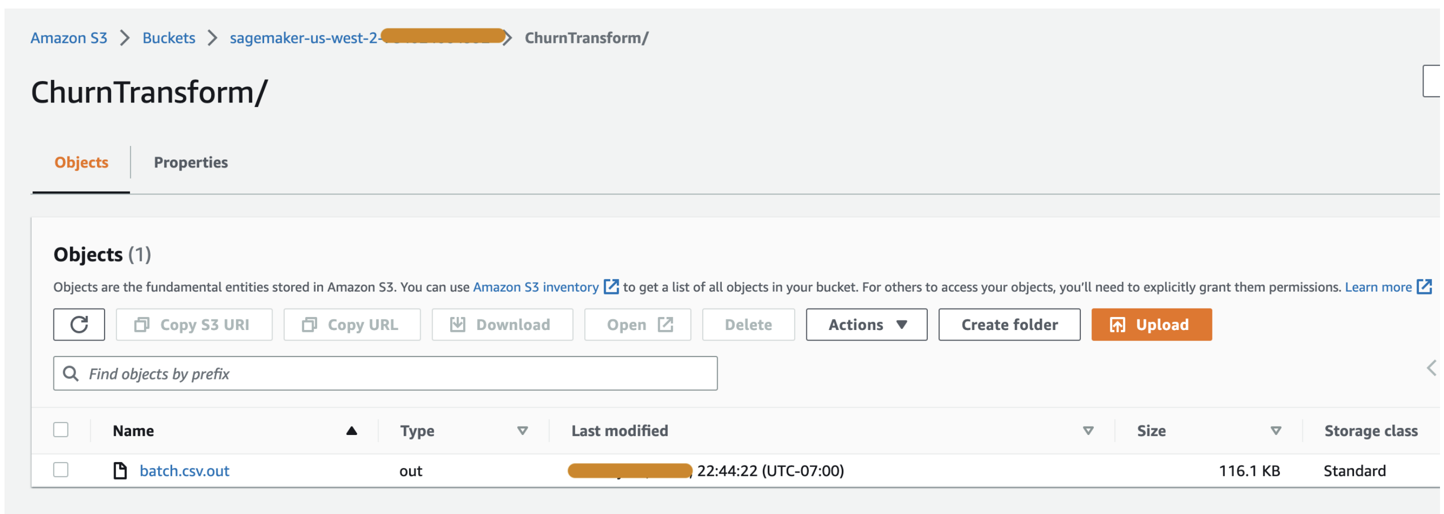
Go to home within the studio domain and click on pipelines.


Script (TensorFlow)
Lab 3b. Bring your own Script (PyTorch)
Lab3c. Bring your own Container
Lab 4. Autopilot, Debugger and Model Monitor
Autopilot
Debugger
Model Monitor
Lab 5. Bias and Explainability
Bias and Explainability- Tabular Data
Lab 6. SageMaker Pipelines
Option 1: Train Pipeline (SageMaker Pipelines)
Option 2: Batch Inference Pipeline (SageMaker Pipelines)
Option 3: SageMaker Projects
Lab 7. Real Time ML inference on Streaming Data
Lab 8. Build ML Model with No Code Using Sagemaker Canvas
Lab 9. Amazon SageMaker JumpStart
Lab 10. ML Governance Tools for Amazon SageMaker
Lab 11. SageMaker Notebook Instances
Content preferences
Language


Amazon SageMaker Studio
File
Edit
Home
Data
AutoML
Experiments
Notebook jobs
Pipelines
Models
Deployments
Quick start solutions

Once the pipeline execution is complete, the inferences generated should be available within the S3 Bucket.





Conclusion

In this lab we have walked through how to build an ML workflow with the different steps to generate inferences using batch transform with trained model artifact along with other built-in SageMaker features for churn prediction. The solution can be extended with other steps as needed to implement your own ML workflow. To learn more about SageMaker Pipelines, check out the website and the documentation.

Previous Next