

Automation of end to end Machine learning pipeline using Amazon Web Services(AWS)

Vehicle Insurance Claim Fraud Detection using Machine Learning

Naveen Raju Sreerama Raju Govinda Raju
naveenraju100@gmail.com | +1 224 7067718 | +91 9886157101 |
<https://www.linkedin.com/in/naveen-raju-s-g-bb1486124>
<https://naveenrajusg.github.io/Portfolio/>

Table of Contents

| |
|--|
| 1) Project Description |
| 2) Dataset..... |
| 3) Exploratory data Analysis..... |
| 4) Automating Machine Learning Workflows..... |
| A) Set up your Amazon SageMaker Studio domain..... |
| B) Set up a SageMaker Studio notebook and parameterize the pipeline..... |
| C) Build the pipeline components..... |
| D) Build and run the pipeline..... |
| E) Test the pipeline by invoking the endpoint..... |
| F) Clean up the resources..... |

Please zoom in to view the screenshots clearly

1. Project Description

The goal of this project is to build a model that can detect auto insurance fraud. And also automate different stages of ML life cycle like Data processing, Model training, Model evaluation, model creation, model bias check, model explain-ability check, model registration and model deployment in Amazon web services(AWS).

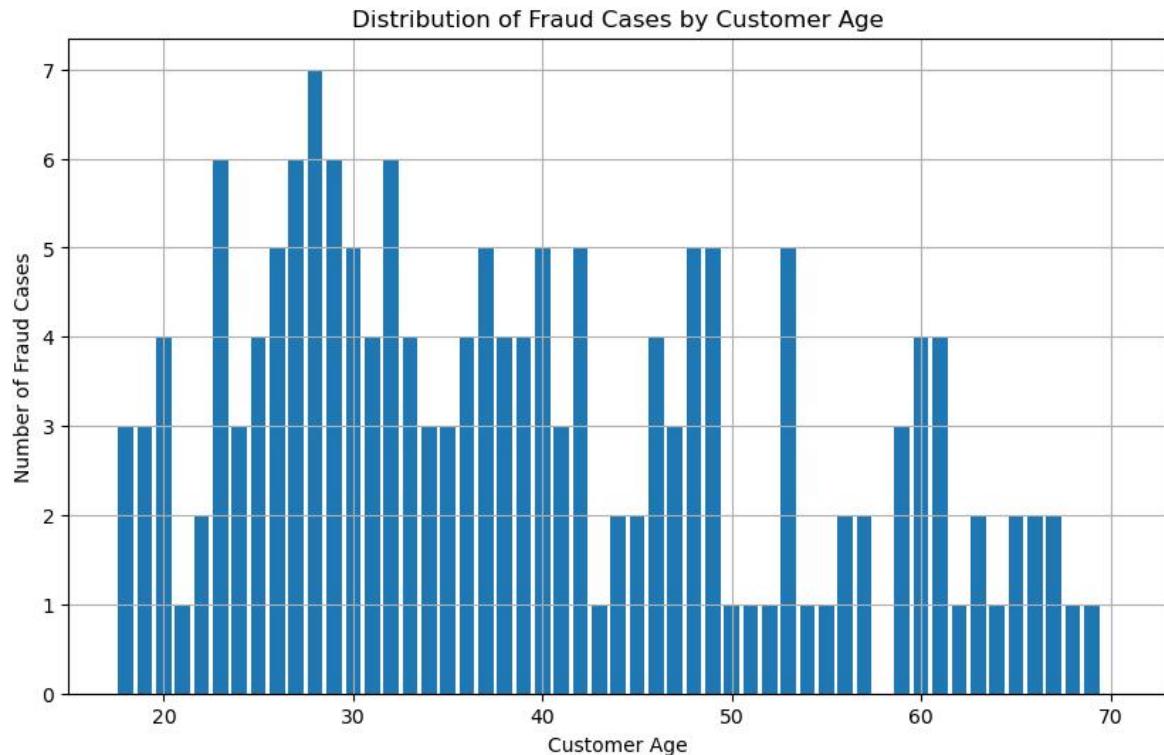
2. Dataset

| Attribute | Type |
|---------------------------|----------|
| fraud | int64 |
| driver_relationship | category |
| incident_type | category |
| collision_type | category |
| incident_severity | category |
| authorities_contacted | category |
| num_vehicles_involved | int64 |
| num_injuries | int64 |
| num_witnesses | int64 |
| police_report_available | float64 |
| injury_claim | int64 |
| vehicle_claim | float64 |
| total_claim_amount | float64 |
| incident_month | int64 |
| incident_day | int64 |
| incident_dow | int64 |
| incident_hour | int64 |
| customer_age | int64 |
| months_as_customer | int64 |
| num_claims_past_year | int64 |
| num_insurers_past_5_years | int64 |
| policy_state | category |
| policy_deductable | int64 |
| policy_annual_premium | int64 |
| policy_liability | float64 |
| customer_gender | category |
| customer_education | category |
| auto_year | int64 |

3. Exploratory data Analysis

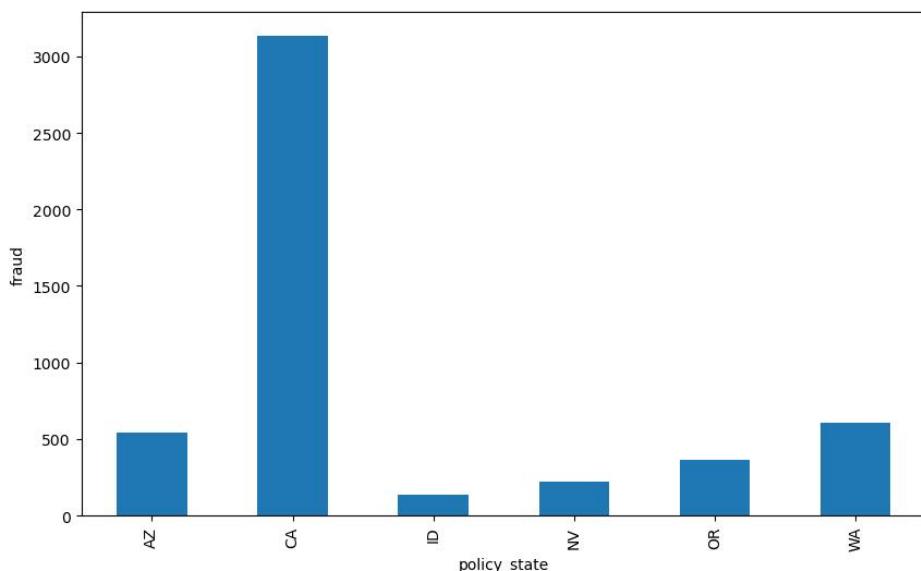
A) Distribution of fraud cases by customer age

We can see customers with age between 20 - 30 have more fraud cases.



B) Distribution of policy_state which is fraud claim.

California has highest fraud claims whereas Idaho state has least fraud claims.



Percentages of fraud claims state wise :-

Idaho state :

Total size of observations of Idaho state= 135

Number of frauds by Idaho state= 4

Percentage of fraud in claims made= 2.96 %

Arizona state :

Total size of observations of arizona state= 539

Number of frauds by Arizona state= 17

Percentage of fraud in claims made= 3.15 %

Nevada state:

Total size of observations of Nevada state= 222

Number of frauds by Nevada state= 6

Percentage of fraud in claims made= 2.7 %

Oregon state:

Total size of observations of Oregon state= 363

Number of frauds by Oregon state= 12

Percentage of fraud in claims made= 3.31 %

Washington state:

Total size of observations of Washington state= 607

Number of frauds by Washington state= 19

Percentage of fraud in claims made= 3.13 %

California state:

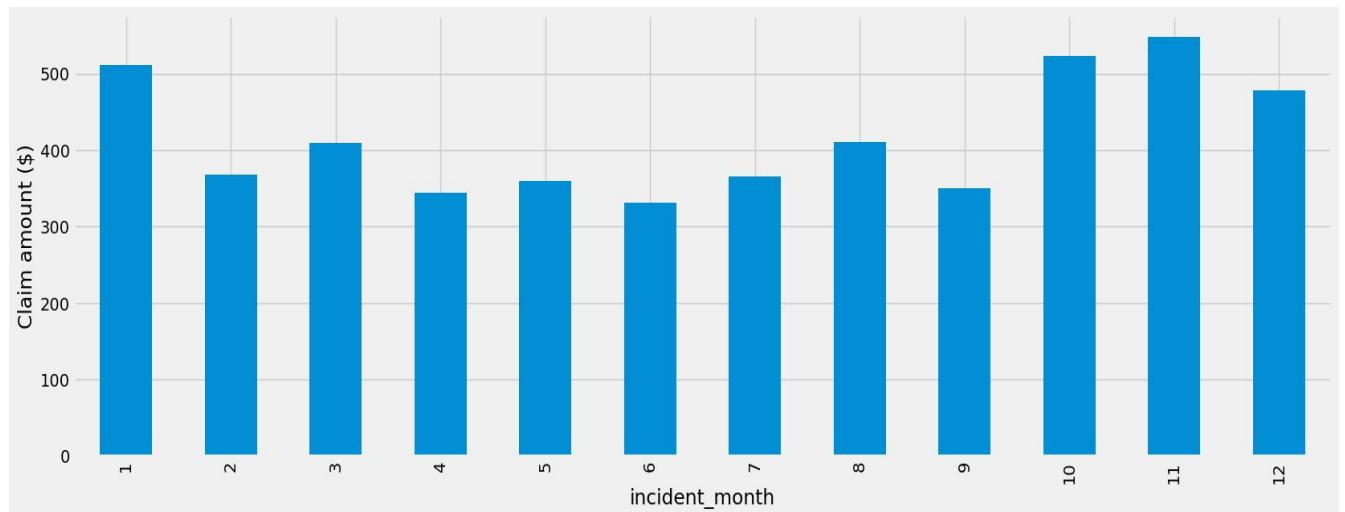
Total size of observations of California state= 3134

Number of frauds by California state= 106

Percentage of fraud in claims made= 3.38 %

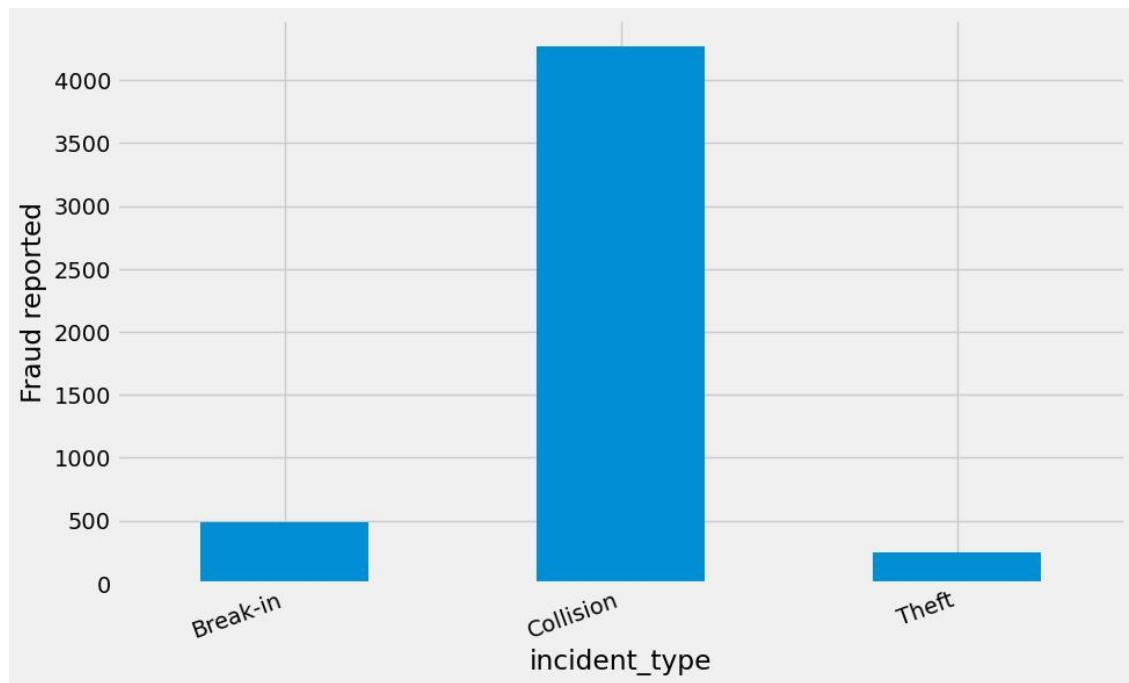
C) Total claim amounts in each month of the year

January, October and November has highest claims compared to other months



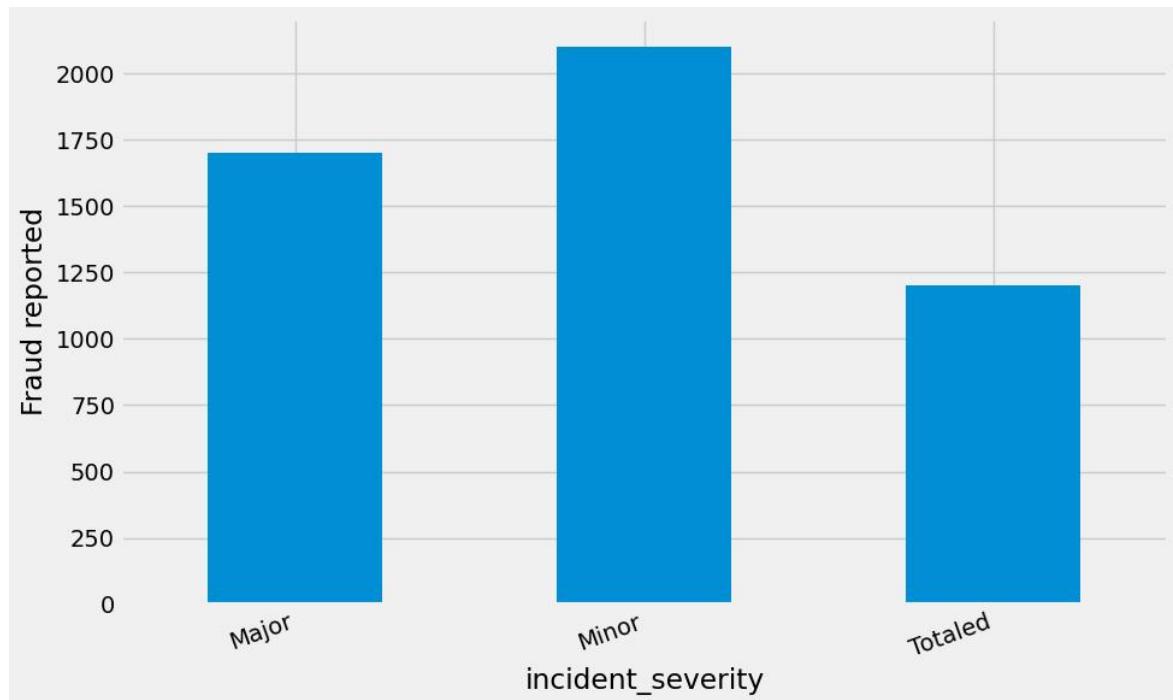
D) Most common type of Incident type of fraud cases

Vehicles claimed to be met with collision is the most fraud cases and least being the theft cases.



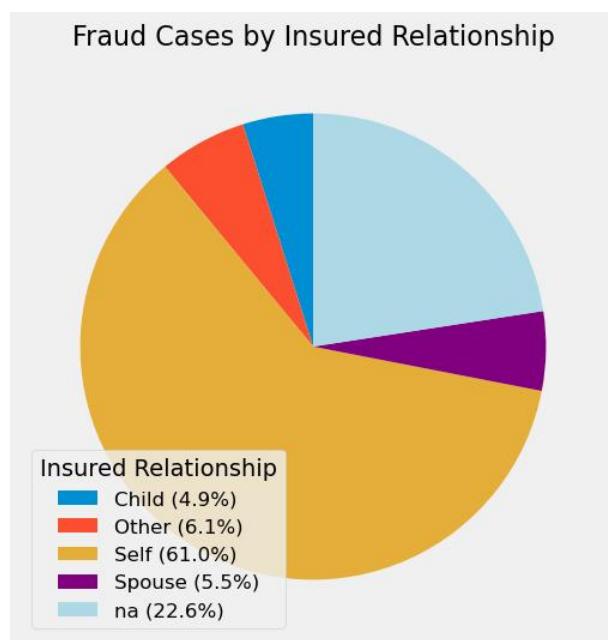
E) Distribution of incident category which is fraud claim.

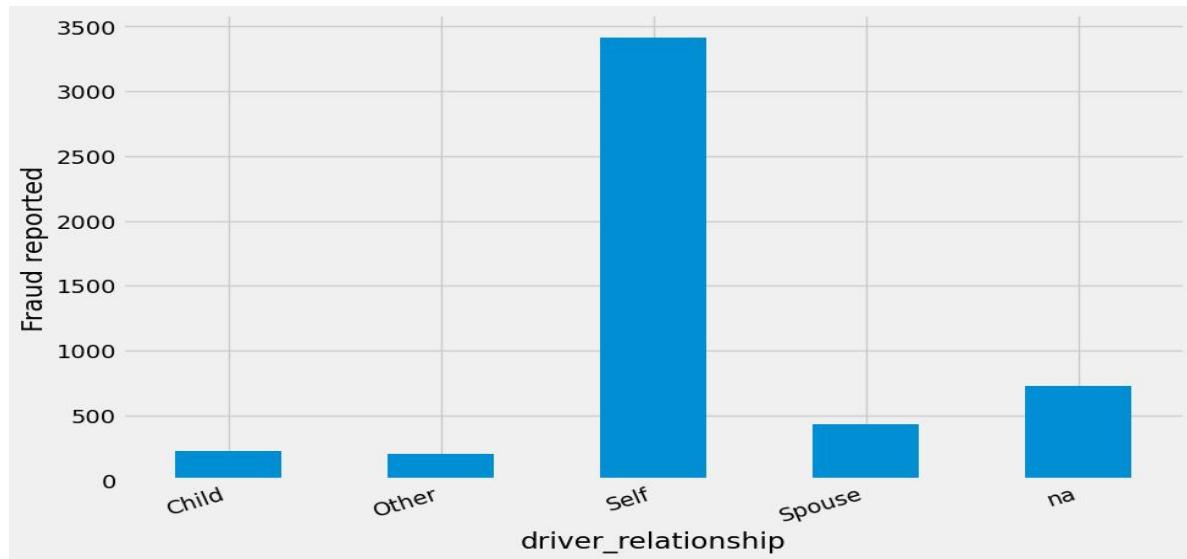
Incident severity category belonging to Minority had most fraud cases, second being Major and least being Totaled.



F) Distribution of driver relationship category which is fraud claim.

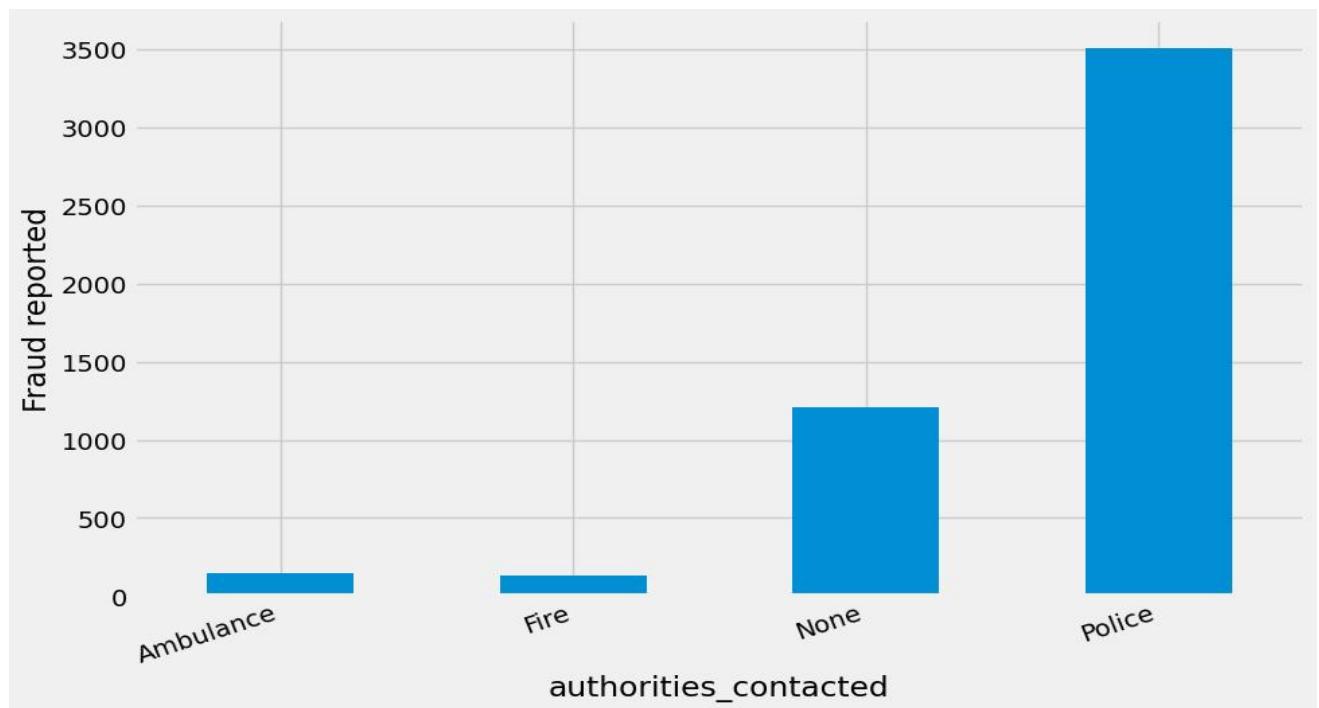
Insurance claims with driver relationship as self has the highest fraudulent cases.





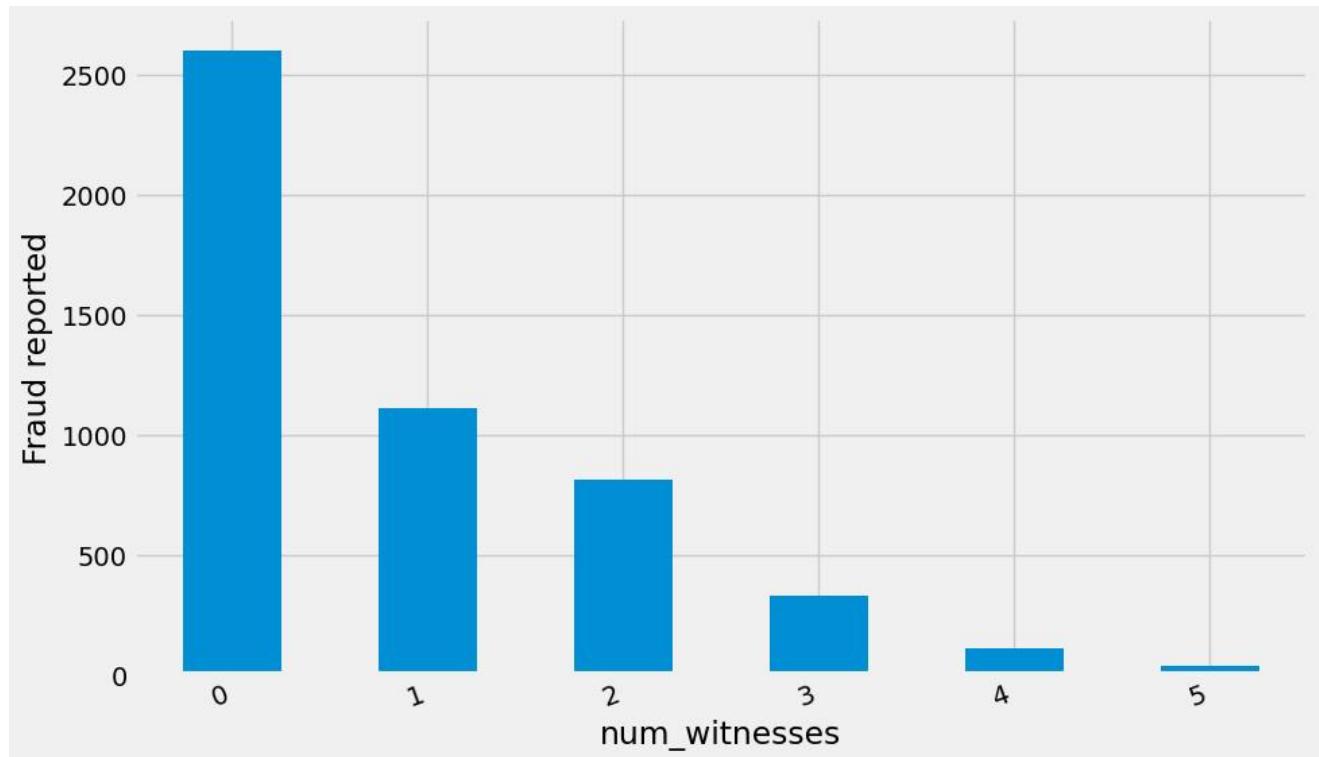
G) Distribution of driver authority category which is fraud claim.

Most fraud claims made were when authorities contacted were Police, which is quite surprising. As this should not be the fraud case.



H) Distribution of number of witness category which is fraud claim.

With 0 being the number of witness when fraud cases were filled is maximum which is as expected.



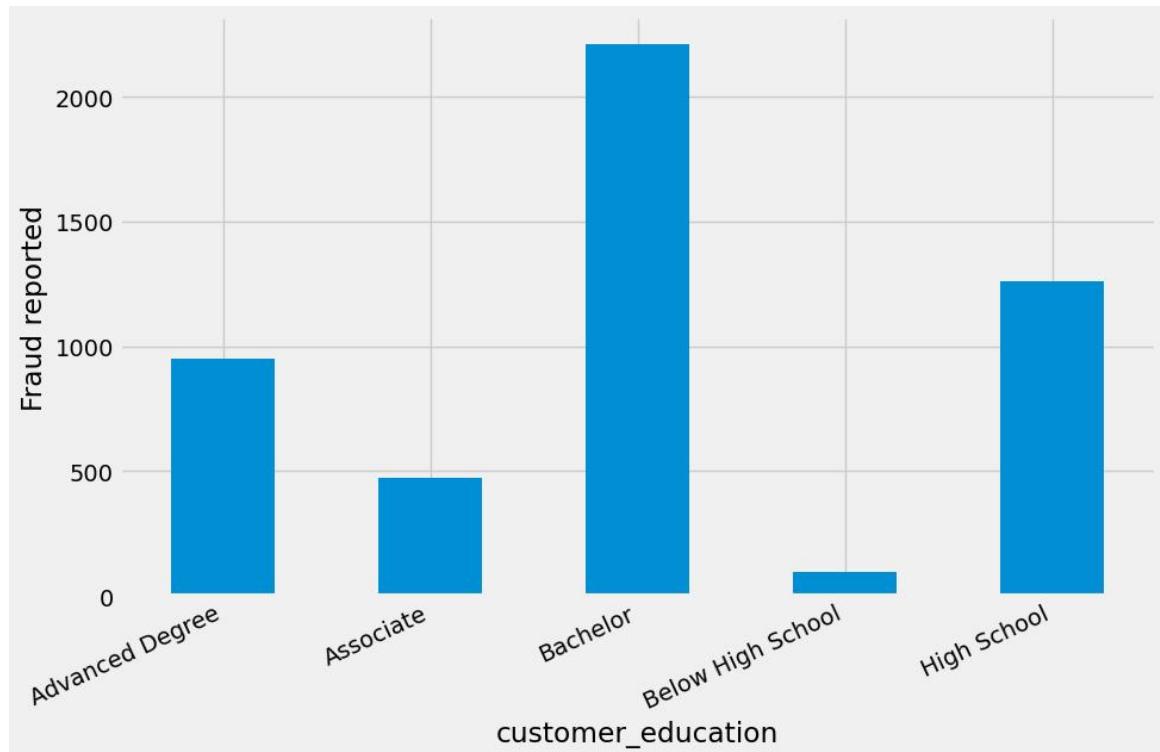
I) Distribution of incident hour category which is fraud claim.

Number of fraudulent cases are considerably high between 8am to 4pm. Least at 9pm-3am.



J) Distribution of education category of persons who did fraud claim.

People with Bachelors degree do a maximum fraudulent claims and people with education below high school do less fraud claims.



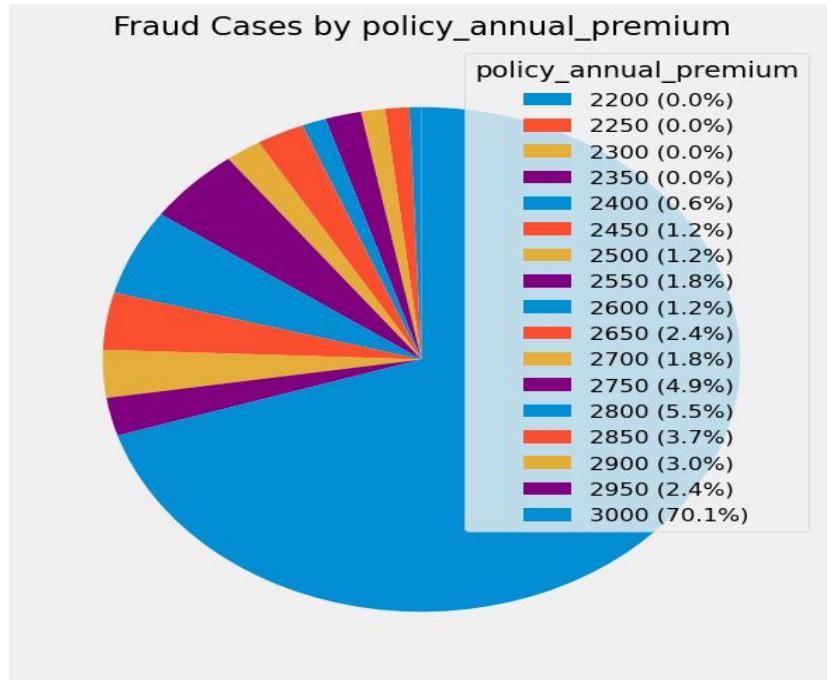
K) Distribution of gender category who did fraud claim.

Males do majority fraud claims compared to females.



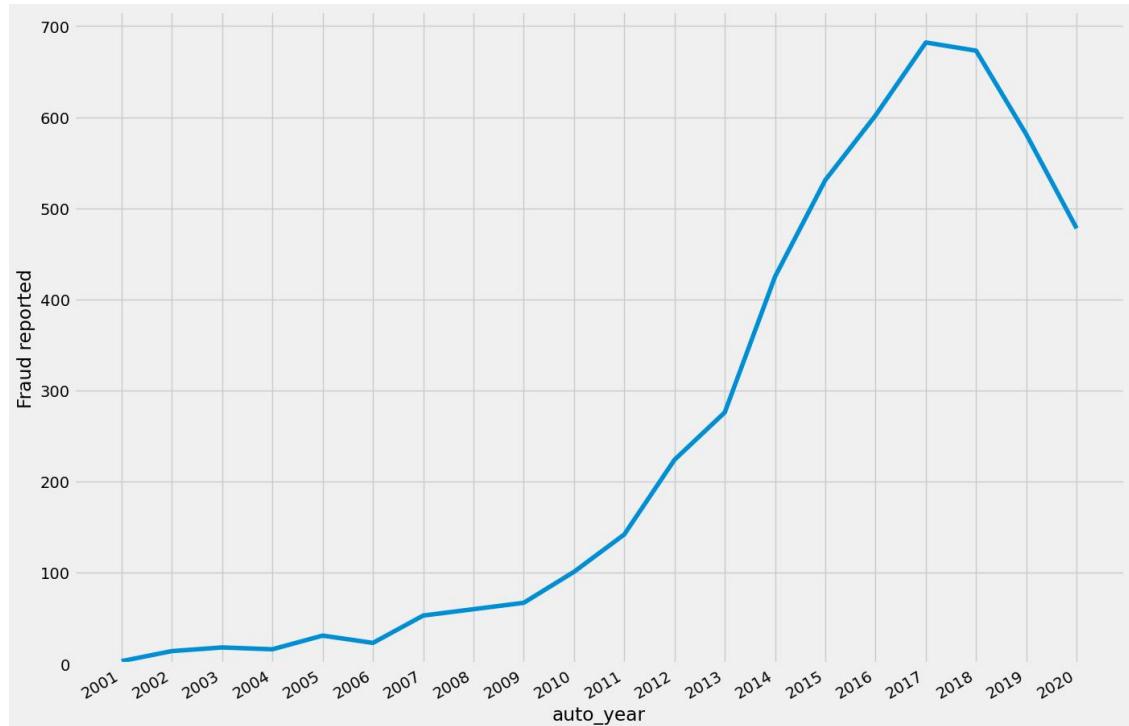
L) Distribution of Annual premium category which has fraud claims.

This describes that Annual Policy with highest Premium gets more fraud claims.



M) Distribution of vehicles make year vs fraud claims.

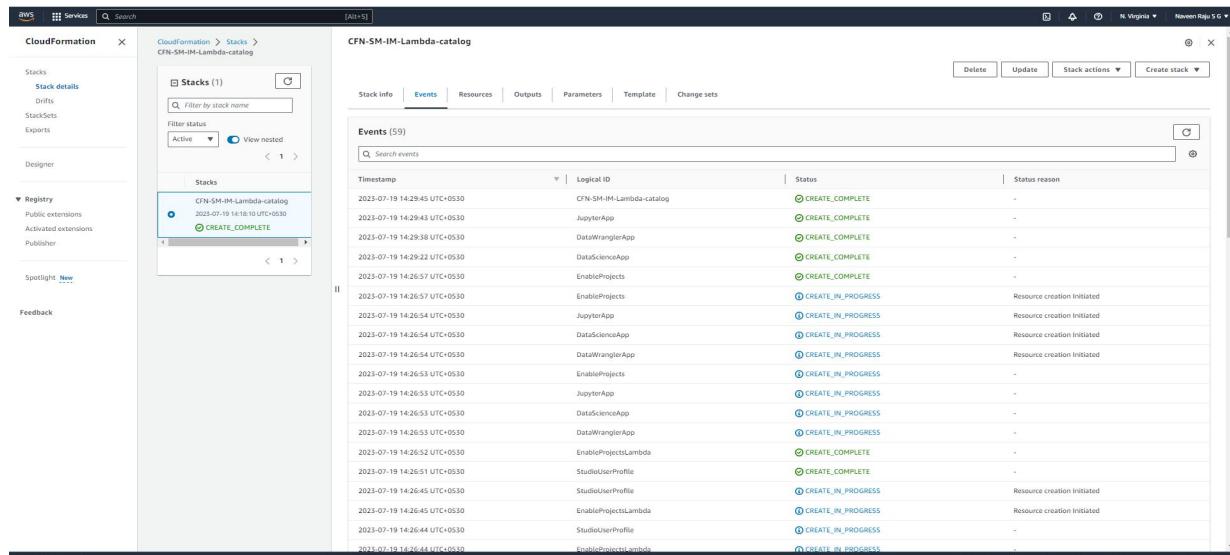
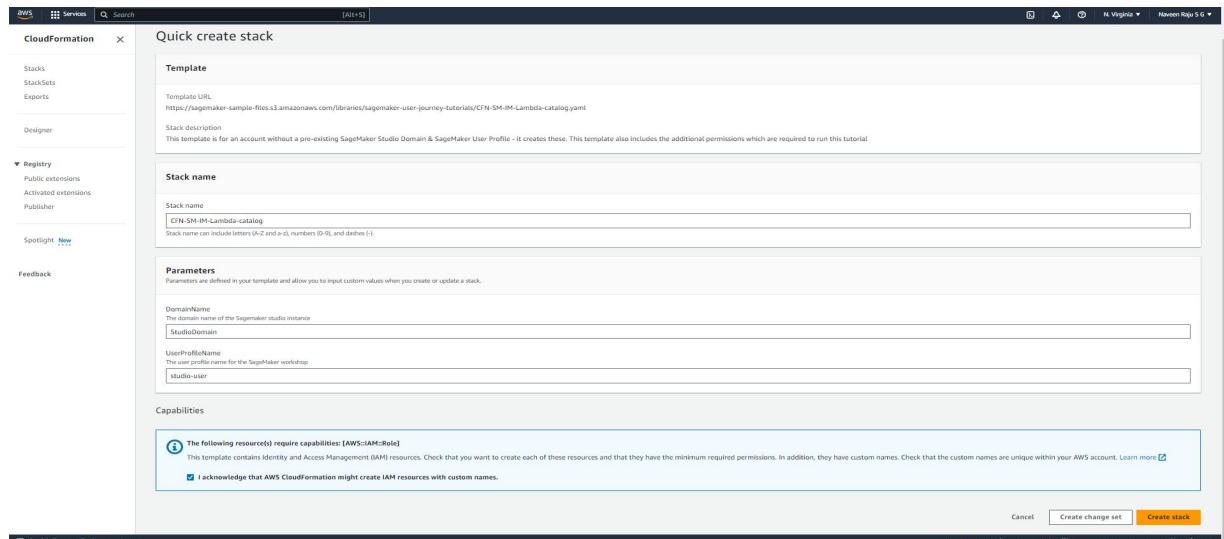
Vehicles with make year starting from 2014 has increasing trend of false claims, with vehicles made in 2017-2018 has highest fraud claims.



5) Automating Machine Learning Workflows

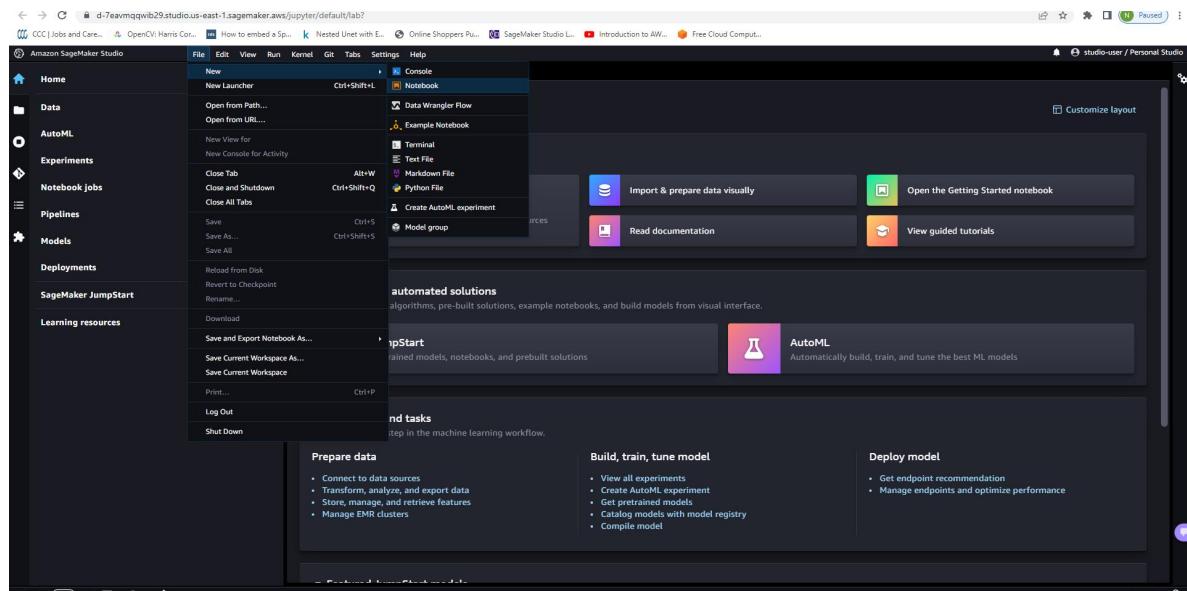
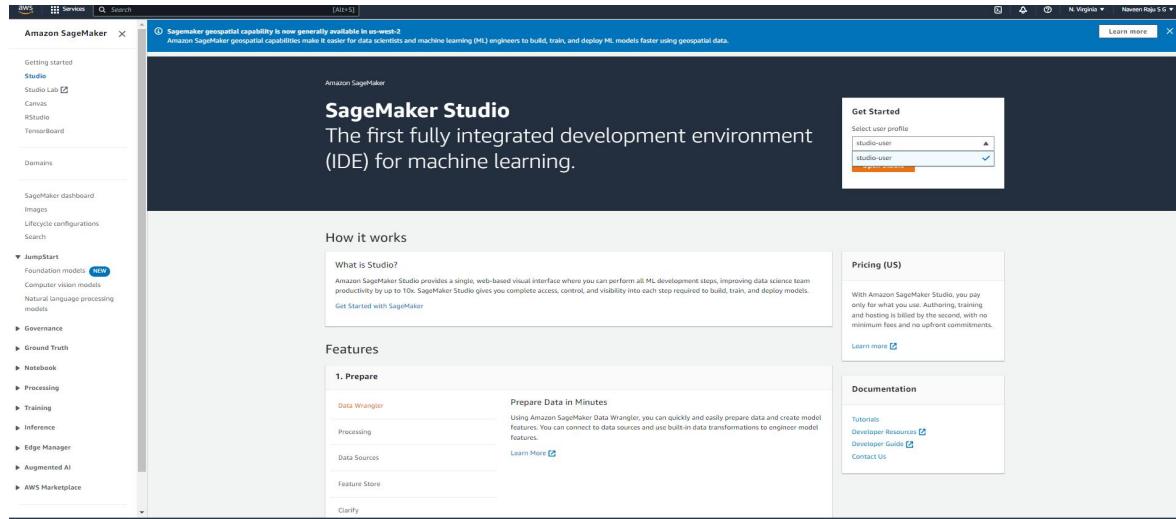
Step 1 : Setting up SageMaker studio domain

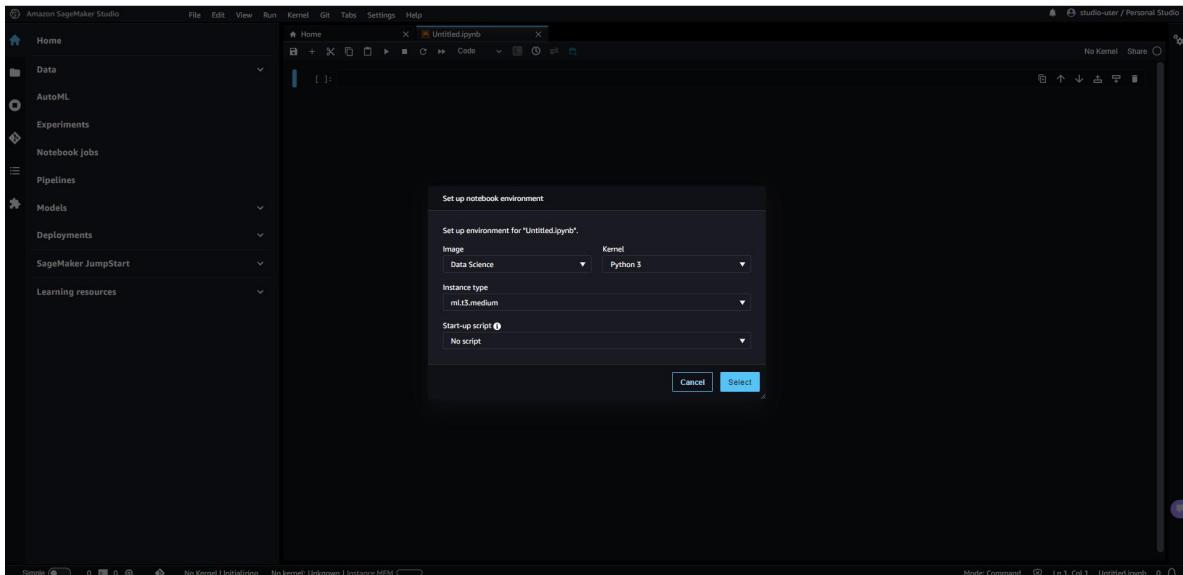
Open AWS Cloud Formation console, enter stack name ‘CFN-SM-IM-Lambda-catalog’ and click create stack, this creates a domain named Studio Domain with user named studio-user. Basically the stack will have several permissions like ‘AmazonSageMakerFullAccess’ , ‘AWSCloudFormationFullAccess ’ , ‘AmazonSageMakerPipelinesIntegrations ’, AWS Lambda related permissions, S3 buckets full access, EC2 instances access.



Step 2 : Setting up a SageMaker Studio notebook and parameterize the pipeline

Open SageMaker Studio through AWS console. Select Studio Domain and user studio-user profile and open jupyter notebook with Image - Data Science and Kernel - Python 3.



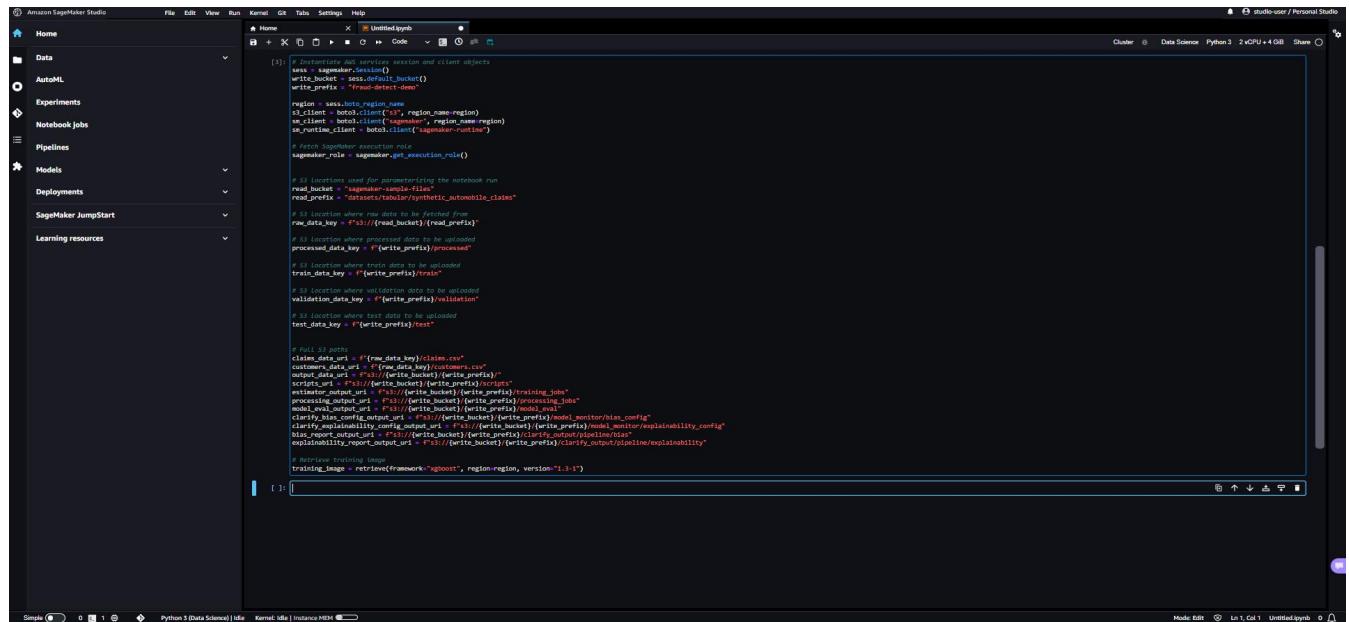


Main.py

Code all the import statements required:

A screenshot of the Amazon SageMaker Studio interface, similar to the one above, showing a code editor with a single cell containing a large amount of Python code. The code consists primarily of import statements from the 'sagemaker' library and its sub-modules, such as 'pandas', 'pd', 'json', 's3', 's3fs', 'pathlib', 'io', 'sagemaker', 'sagemaker.deserializers', 'sagemaker.serializers', 'sagemaker.videos', 'sagemaker.estimator', 'sagemaker.sklearn_processing', 'sagemaker.workflow', 'ProcessingInput', 'ProcessingOutput', 'ScriptProcessor', 'sagemaker.inputs', 'TrainingInput', 'sagemaker.workflow.pipeline', 'sagemaker.workflow.steps', 'sagemaker.workflow.condition_step', 'sagemaker.workflow.check_job_config', 'sagemaker.workflow.parameters', 'sagemaker.workflow.model_clarity', 'sagemaker.workflow.step_collections', 'sagemaker.workflow.step_collections.RegisterModelStep', 'sagemaker.workflow.step_collections.ConditionStepIfNotEqualTo', 'sagemaker.workflow.properties', 'sagemaker.workflow.condition_step', 'sagemaker.workflow.functions', 'sagemaker.workflow.lambda_step', 'sagemaker.lambda_helper', 'sagemaker.model_metrics', and 'sagemaker drift'. The code is intended to demonstrate the required imports for working with Amazon SageMaker.

The below code sets up SageMaker and S3 client objects using SageMaker and AWS SDK's. Basically these objects are required to make SageMaker perform actions like deploying, invoking endpoints, and also interact with AWS S3 and AWS Lambda. Also code sets up S3 bucket locations as where original dataset, pre-processed data set and model artifacts need to be present and stored.



```

# Import boto3 and session
sess = boto3.Session()
write_bucket = sess.default_bucket()
write_prefix = "fraud-detect-0000"

region_name = sess.region_name
region = sess.region
s3_client = boto3.client("s3", region_name=region)
sm_client = boto3.client("sagemaker", region_name=region)
sm_runtime_client = boto3.client("sagemaker-runtime")

# Fetch Sagemaker execution role
sagemaker_role = sm_client.get_execution_role()

# S3 locations used for parameterizing the notebook run
read_bucket = "sagemaker-sample-files"
read_prefix = "datasets/titanic/synthetic/automobile_claims"
raw_data_key = f"s3://{read_bucket}/{read_prefix}/"

# S3 Location where processed data to be uploaded
processed_data_key = f"{write_prefix}/processed"

# S3 Location where train data to be uploaded
train_data_key = f"{write_prefix}/train"

# S3 Location where validation data to be uploaded
validation_data_key = f"{write_prefix}/validation"

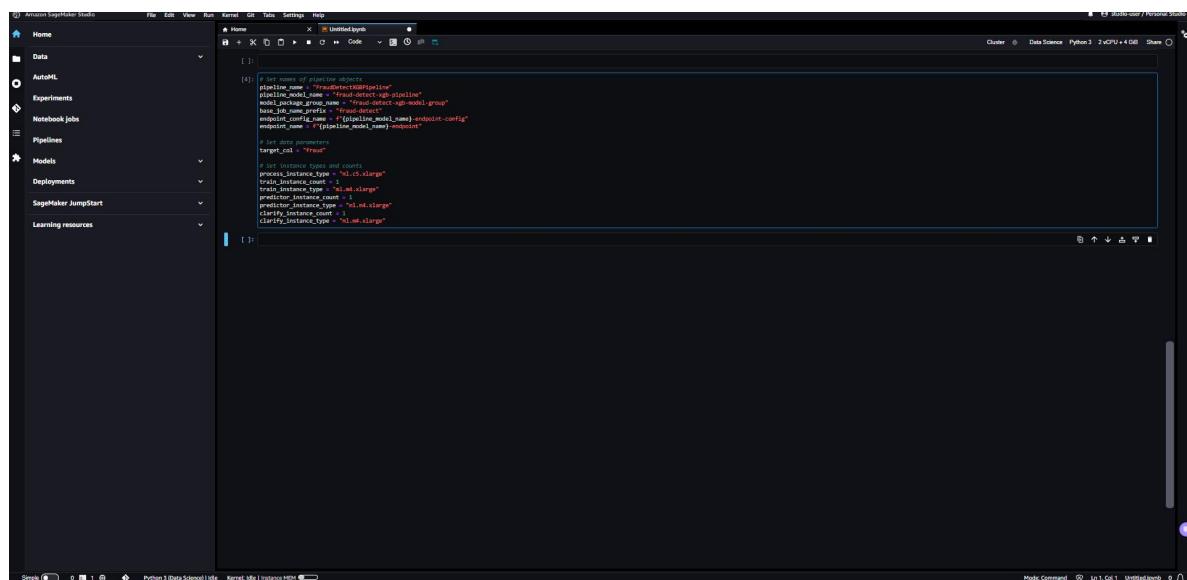
# S3 Location where test data to be uploaded
test_data_key = f"{write_prefix}/test"

# Full S3 paths
claims_data_uri = f"(tmp/data/key)/claims.csv"
customers_data_uri = f"(tmp/data/key)/customers.csv"
output_data_uri = f"s3://(write_bucket)/(write_prefix)/"
script_output_uri = f"s3://(write_bucket)/(write_prefix)/script"
estimator_output_uri = f"s3://(write_bucket)/(write_prefix)/training_jobs"
processing_output_uri = f"s3://(write_bucket)/(write_prefix)/processing_jobs"
model_output_uri = f"s3://(write_bucket)/(write_prefix)/models"
clarify_bias_config_output_uri = f"s3://(write_bucket)/(write_prefix)/model_monitor/bias_config"
clarify_explainability_report_output_uri = f"s3://(write_bucket)/(write_prefix)/clarify_output/pipeline/explainability_config"
explainability_report_output_uri = f"s3://(write_bucket)/(write_prefix)/clarify_output/pipeline/explainability"
explainability_report_output_key = f"(write_prefix)/clarify_output/pipeline/explainability"

# Retrieve training image
training_image = retrieve(framework="xgboost", region=region, version="1.3-1")

```

The below code contains names of several SageMaker pipeline components, training and inference instance types and counts.



```

# Set names of pipeline objects
pipeline_name = "FraudDetectorSagePipeline"
pipeline_model_name = "fraud-detect-sgh-pipeline"
base_idb_name_prefix = "fraud-detect"
endpoint_config_name = f"(pipeline_model_name)-endpoint-config"
role_name = f"(pipeline_model_name)-role"
model_name = f"(pipeline_model_name)-model"

# Set data parameters
target_col = "Fraud"
# Set instance types and counts
train_instance_type = "ml.c5.large"
train_instance_type_1 = "ml.m5.xlarge"
train_instance_type_2 = "ml.m5.2xlarge"
predictor_instance_type = "ml.m5.xlarge"
clarify_instance_type = "ml.m5.2xlarge"

```

The below code contains parameterization I.e setting up pipeline input parameters like processing instance types, training instance type and count, deployment instance type and count, Clarify check instance and count, model bias check parameter, registering new baseline model bias parameter, model explainability, model registration parameter, model approval parameter check parameter this allows to specify input parameters at run time without changing pipeline code.

This is achieved by using modules available in `sagemaker.workflow.parameters` module, those modules are `ParameterInteger`, `ParameterFloat`, `ParameterString`, and `ParameterBoolean`.

```

# Set up pipeline input parameters
# Set processing instance type
process_instance_type_param = ParameterString(
    name="processinginstancetype",
    default_value=process_instance_type,
)

# Set training instance type
train_instance_type_param = ParameterString(
    name="traininstance-type",
    default_value=train_instance_type,
)

# Set training instance count
train_instance_count_param = ParameterInteger(
    name="traininstancescount",
    default_value=train_instance_count
)

# Set deployment instance type
deploy_instance_type_param = ParameterString(
    name="deployinstance-type",
    default_value=predictor.instance_type,
)

# Set deployment instance count
deploy_instance_count_param = ParameterInteger(
    name="deployinstancescount",
    default_value=predictor.instance_count
)

# Set clarify instance type
clarify_instance_type_param = ParameterString(
    name="clarifyinstance-type",
    default_value=clarify_instance_type,
)

# Get model bias check params
skip_check_model_bias_param = ParameterBoolean(
    name="skipcheckmodelbias",
    default_value=False
)

register_new_baseline_model_bias_param = ParameterBoolean(
    name="registernewbaselinebias",
    default_value=False
)

supplied_baseline_constraints_model_bias_param = ParameterString(
    name="suppliedbaselineconstraintsmodelbias",
    default_value=""
)

# Get model explainability check params
skip_explainability_check_param = ParameterBoolean(
    name="skipmodelexplainabilitycheck",
    default_value=False
)

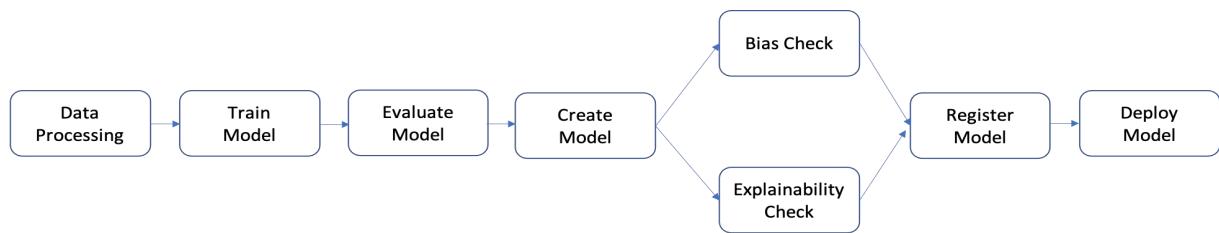
register_new_baselining_model_explainability_param = ParameterBoolean(
    name="registernewmodelexplainabilitybaseline",
    default_value=False
)

supplied_baselining_constraints_explainability_param = ParameterString(
    name="suppliedbaseliningsconstraintsexplainability",
    default_value=""
)

# Get model approval params
model_approval_status_param = ParameterString(
    name="ModelApprovalStatus", default_value="Approved"
)

```

Step 3 : Building the pipeline components



SageMaker encodes the pipeline in a Directed Acyclic Graph (DAG), where each node represents a step and the connections between the nodes represent dependencies.

Read original data sets, Preprocess data and do train,test,val data split.

Read dataset : Read claims.csv and customers.csv dataset

Pre-process dataset:

- make column names to lowercase and replace “ “ in column names with”_”

- Do left inner join of 2 datasets using column “policy_id”
- Drop customer_zip column
- Full missing values with ‘na’
- One-hot vector encoding of only categorical columns
- Map ordinal columns with user defined ordinal numerical values
- drop policy_id and customer_gender_unkown columns
- Do train test and validation split based on 80:10:10 ratio and store respective files in respective s3 buckets.

```

# Preprocessing script for policy dataset
# This script performs several steps to prepare the data for machine learning
# 1. Handles command-line arguments for input and output paths
# 2. Loads data from CSV files
# 3. Merges the two datasets
# 4. Drops unnecessary columns
# 5. Encodes categorical columns using one-hot encoding
# 6. Splits the data into training, validation, and test sets
# 7. Writes the processed datasets to local paths
# 8. Prints logs for each step

import argparse
import os
import sys
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

logger = logging.getLogger()
logger.setLevel(logging.INFO)
logger.addHandler(logging.StreamHandler())

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--train-data-path", type=str, default=".")
    parser.add_argument("--test-data-path", type=str, default=".")
    parser.add_argument("--val-data-path", type=str, default=".")
    parser.add_argument("--output-path", type=str, default=".")
    args = parser.parse_args()
    logger.info(f"Received arguments: {args}")

    # Set local path prefix in the processing container
    local_prefix = "/opt/ml/processing"

    # Input data path
    input_data_path_claims = os.path.join(local_prefix, "claims.csv")
    input_data_path_customers = os.path.join(local_prefix, "customers.csv")

    # Output data path
    output_data_path_claims = os.path.join(local_prefix, "claims.csv")
    output_data_path_customers = os.path.join(local_prefix, "customers.csv")

    # Load data
    df_claims = pd.read_csv(input_data_path_claims)
    df_customers = pd.read_csv(input_data_path_customers)

    logger.info("Merging customers data from {}...".format(input_data_path_customers))
    df_customers = df_customers.set_index("customer_id")

    logger.info("Renaming column names...")
    df_claims = df_claims.rename({(c,): c.replace(' ', '_') for c in df_claims.columns}, axis=1)
    df_customers = df_customers.rename({(c,): c.replace(' ', '_') for c in df_customers.columns}, axis=1)

    logger.debug("Merging datasets...")
    df_data = df_claims.merge(df_customers, on="policy_id", how="left")

    # Drop selected columns not required for model building
    df_data = df_data.drop(["customer_id"], axis=1)

    # One-hot encoding
    ordinal_cols = ["policy_report_available", "policy_liability", "customer_education"]
    df_data[ordinal_cols] = df_data[ordinal_cols].apply(LabelEncoder().fit_transform)

    # Select categorical columns and fillna with 'na'
    cat_cols_all = list(df_data.select_dtypes(object).columns)
    cat_cols_all.remove("customer_id")
    df_data[cat_cols_all] = df_data[cat_cols_all].fillna("na")

    logger.debug("One-hot encoding categorical columns...")
    df_data = pd.get_dummies(df_data, columns=df_data[cat_cols_all])

    logger.info("Encoding ordinal columns...")
    # Ordinal encoding
    mapping = {
        "1": "1",
        "2": "2",
        "3": "3",
        "4": "4",
        "5": "5"
    }

    df_data["policy_liability"] = df_data["policy_liability"].map(mapping)
    df_data["policy_liability"] = df_data["policy_liability"].astype(int)

    mapping = {
        "Below High School": "0",
        "High School": "1",
        "Some College": "2",
        "Bachelor": "3",
        "Advanced Degree": "4"
    }

    df_data["customer_education"] = df_data["customer_education"].map(mapping)
    df_data["customer_education"] = df_data["customer_education"].astype(int)

    df_processed = df_data.copy()
    df_processed = df_processed.drop(["policy_id", "customer_gender_unkown"], axis=1)

    # Split into train, validation, and test sets
    train_ratio = args.train_ratio
    val_ratio = args.validation_ratio
    test_ratio = args.test_ratio
    test_ratio = args.validation_ratio

    logger.debug("Splitting data into train, validation, and test sets")
    y = df_processed["label"]
    X_train_val, X_test, y_train_val, y_test = train_test_split(y, test_size=test_ratio, random_state=42)
    X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=val_ratio, random_state=42)

    train_df = pd.concat([X_train, y_train], axis=1)
    val_df = pd.concat([X_val, y_val], axis=1)
    test_df = pd.concat([X_test, y_test], axis=1)
    dataset_df = pd.concat([y, X_val], axis=1)

    logger.info("Train data shape after preprocessing: ({}, {})".format(len(train_df), len(train_df.columns)))
    logger.info("Validation data shape after preprocessing: ({}, {})".format(len(val_df), len(val_df.columns)))
    logger.info("Test data shape after preprocessing: ({}, {})".format(len(test_df), len(test_df.columns)))

    # Save processed datasets to the local paths in the processing container
    logger.debug("Writing processed datasets to container local path")
    train_output_path = os.path.join(local_prefix, "train.csv")
    validation_output_path = os.path.join(local_prefix, "val.csv")
    test_output_path = os.path.join(local_prefix, "test.csv")
    dataset_output_path = os.path.join(local_prefix, "dataset.csv")

    logger.info("Saving train data to ({})".format(train_output_path))
    train_df.to_csv(train_output_path, index=False)

    logger.info("Saving validation data to ({})".format(validation_output_path))
    val_df.to_csv(validation_output_path, index=False)

    logger.info("Saving test data to ({})".format(test_output_path))
    test_df.to_csv(test_output_path, index=False)

    logger.info("Saving full processed data to ({})".format(full_processed_output_path))
    dataset_df.to_csv(dataset_output_path, index=False)

```

```

logger.info("Saving full processed data to ({})".format(full_processed_output_path))
dataset_df.to_csv(full_processed_output_path, index=False)

```

The following code will instantiate processor and SageMaker pipeline step to execute pre-processing script. Because pre-processing script is written in Pandas we use SKLearnProcessor. From this code SageMaker PipelinesStep will take arguments like processor, s3 locations of dataset, output s3 locations to save pre-processed dataset, train,test and validation split ratio.

```

from sagemaker.workflow.pipeline_context import PipelineSession
# Create a processing script to ss
# s3://write_prefix/scripts/preprocessing.py, Bucket:write_bucket, Key:#(write_prefix)/scripts/preprocessing.py

# Create a SKLearnProcessor
sklearn_processor = SKLearnProcessor(
    framework_version='1.3-23-1',
    role=role,
    instance_count=1,
    instance_type='ml.m5.xlarge',
    base_job_name="#(base_job_name_prefix)-processing",
)

# Create a ProcessingStep
process_step = ProcessingStep(
    name="data-processing",
    processor=sklearn_processor,
    inputs=[ProcessingInput(source="s3://write_prefix/customer_data_uris", destination="s3://write_prefix/output/pkl/processing/customer")],
    outputs=[
        ProcessingOutput(destination="processing_output_ur1", output_name="train_data", source="s3://write_prefix/output/pkl/processing/train"),
        ProcessingOutput(destination="processing_output_ur1", output_name="validation_data", source="s3://write_prefix/output/pkl/processing/val"),
        ProcessingOutput(destination="processing_output_ur1", output_name="test_data", source="s3://write_prefix/output/pkl/processing/test"),
        ProcessingOutput(destination="processing_output_ur2", output_name="processed_data", source="s3://write_prefix/output/pkl/processing/full")
    ],
    job_arguments=[
        '--train-ratio', '0.8',
        '--val-ratio', '0.1',
        '--test-ratio', '0.1'
    ]
)
code="#!/usr/bin/python
#(write_prefix)/scripts/preprocessing.py"

```

Training step

The following code contains training code of XGBoost binary classifier and hyperparameters, roc_auc score calculation for train and validation data, saving metrics calculated to specified s3 bucket, saving trained model to specified s3 bucket.

```

#!/usr/bin/python
#(write_prefix)/scripts/xgboost_train.py

import argparse
import os
import sys
import json
import pandas as pd
from sagemaker import session
from sagemaker_xgboost_train import XGBoost
from sagemaker import metric_definitions as metrics

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    # Hyperparameters and algorithm parameters are described here
    parser.add_argument("--model_dir", type=str, default="")
    parser.add_argument("--max_depth", type=int, default=6)
    parser.add_argument("--min_child_weight", type=int, default=1)
    parser.add_argument("--lambda", type=float, default=0.5)
    parser.add_argument("--eta", type=float, default=0.1)
    parser.add_argument("--gamma", type=float, default=0.4)
    parser.add_argument("--eval_metric", type=str, default="auc")
    parser.add_argument("--nthread", type=int, default=4)
    parser.add_argument("--early_stopping_rounds", type=int, default=10)

    # Algorithm specific arguments. Defaults are set in environment variables
    # under SAGEMAKER_XGBOOST_TRAINING
    parser.add_argument("--train_data_dir", type=str, default=os.environ.get("S3_CHANNEL_TRAIN"))
    parser.add_argument("--validation_data_dir", type=str, default=os.environ.get("S3_CHANNEL_VALIDATION"))
    parser.add_argument("--model_dir", type=str, default=os.environ.get("S3_MODEL_DIR"))
    parser.add_argument("--output_dir", type=str, default=os.environ.get("S3_OUTPUT_DIR"))
    parser.add_argument("--output_data_dir", type=str, default=os.environ.get("S3_OUTPUT_DATA_DIR"))

    args = parser.parse_args()

    data_train = pd.read_csv(args.train_data_dir + "train.csv")
    data_train = data_train.drop(['id'], axis=1)
    data_train['label'] = data_train['label'].map({0: -1, 1: 1})
    data_train['age'] = data_train['age'].map(lambda x: float(x))

    data_validation = pd.read_csv(args.validation_data_dir + "validation.csv")
    data_validation = data_validation.drop(['id'], axis=1)
    data_validation['label'] = data_validation['label'].map({0: -1, 1: 1})
    data_validation['age'] = data_validation['age'].map(lambda x: float(x))

    # Cross-validation
    params = {
        "max_depth": args.max_depth,
        "eta": args.eta,
        "gamma": args.gamma,
        "min_child_weight": args.min_child_weight,
        "lambda": args.lambda,
        "nthread": args.nthread,
        "eval_metric": args.eval_metric
    }

    num_boost_round = args.early_stopping_rounds
    every_iter_evaluating_rounds = args.early_stopping_rounds

    # Cross-validation train XGBoost model
    cv_results = xgb.cv(
        params,
        data_train,
        data_val,
        num_boost_round,
        metrics=['auc'],
        evals=[(data_train, 'train'), (data_val, 'val')]
    )

```

Below code contains setting up the model training using SageMaker XGBoost estimator and SageMaker pipelines training step function.

The screenshot shows the Amazon SageMaker Studio interface with a Jupyter notebook titled "Untitled.ipynb". The code in the notebook is as follows:

```
# Set environment variables
hyperparameters = {
    "model_type": "xgb",
    "objective": "binary:logistic",
    "max_depth": 5,
    "eta": 0.3,
    "subsample": 0.75,
    "colsample_bytree": 0.75,
    "lambda": 10
}

# Create estimator
xgb_estimator = XGBClassifier(
    entry_point='sgboost-train.py',
    role=SGBOOST_TRAINING_ROLE,
    instance_type='ml.m5.2xlarge',
    hyperparameters=hyperparameters,
    instance_count=1,
    instance_type='ml.m5.2xlarge',
    framework_version='1.3-1'
)

# Specify the location where the preceding processing step saved train and validation datasets
x_input_train = TrainingInput(
    input_config=xgb_processing_output_config['train_data'],
    content_type='SGFRecords',
    data_type='S3Prefix'
)
x_input_validation = TrainingInput(
    input_config=xgb_processing_output_config['validation_data'],
    content_type='SGFRecords',
    data_type='S3Prefix'
)

# Set pipeline training step
train_step = TrainStep(
    name='SGBoostTrainStep',
    estimator=xgb_estimator,
    inputs=[x_input_train, x_input_validation],
    outputs=[x_train=xgb_input_train, x_validation=xgb_input_validation]
)
```

The below code will create SageMaker model using SageMaker Pipelines CreateModelStep function. This will use output of training step to package model for deployment.

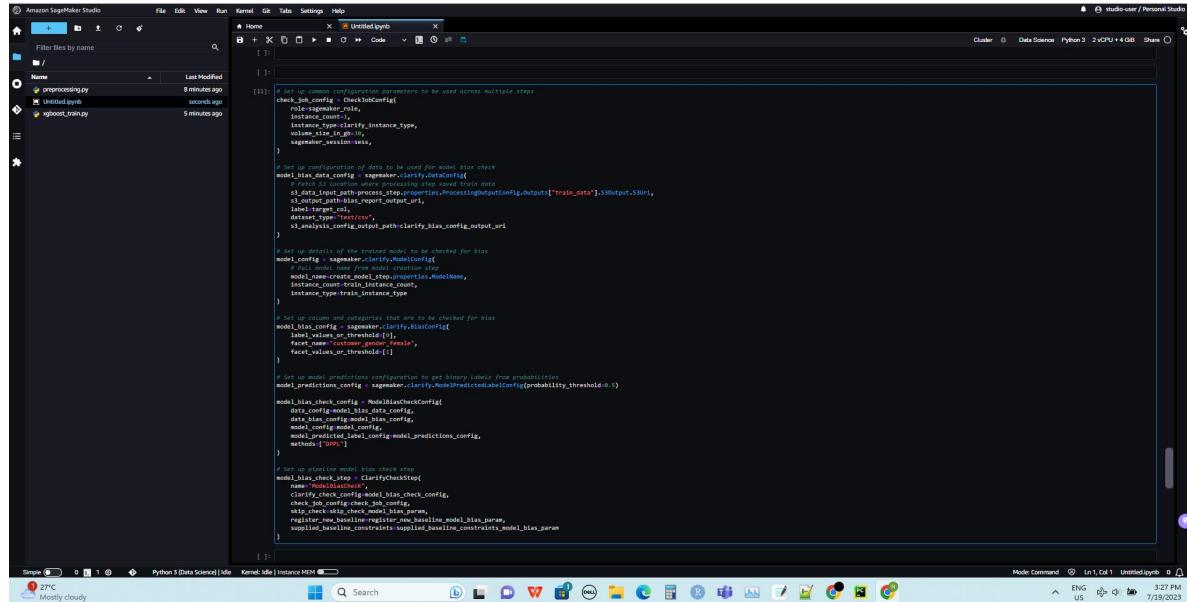
From this we can see the steps of training step to package model for deployment.

```
[1]: # Create a SageMaker model
model = sagemaker.estimator.Estimator(
    image_uris['train'],
    role_data_train.type,
    properties.ModelArtifacts.s3ModelArtifacts,
    role_sagemaker_role
)

# Specify model deployment instance type
inputs = sagemaker.inputs.CreateModelInput(instance_type=deploy_instance_type_params)

create_model_step = CreateModelStep(name='CreateModel', model=model, inputs=inputs)
```

The code below contains data bias check. SageMaker uses SageMaker Clarify for model bias check. It uses training data and model created in previous steps. Model bias check calculates bias metric named (Difference in Positive Proportions in Predicted Labels)DPPL.



```

# Set up common configuration parameters to be used across multiple steps
check_job_config = CheckJobConfig(
    role=sagemaker_role,
    instance_type='ml.m5.xlarge',
    instance_type_clarify=instance_type,
    volume_size_in_gb=10,
    sagemaker_session=sess,
)

# Set up configuration of data to be used for model bias check
model_data_config = sagemaker_clarify.ModelDataConfig(
    s3_data_input_path=processing_step.properties.ProcessingOutputConfig["train_data"].S3Output.S3Uri,
    s3_data_output_report_output_uri=processing_step.properties.ProcessingOutputConfig["clarify_bias_report"].S3Output.S3Uri,
    label_target_col="label",
    dataset_type="text/csv",
    s3_analytics_config_output_path=clarify_bias_config.output_uri
)

# Set up details of the treated model, to be checked for bias
model_config = sagemaker_clarify.ModelConfig(
    model_name=create_model_step.properties.ModelName,
    instance_type='ml.m5.xlarge',
    instance_type_train=instance_type
)

# Set up column and categories that are to be checked for bias
model_bias_config = sagemaker_clarify.BiasConfig(
    label_values_or_threshold=[],
    fairness_categories=[],
    facet_values_or_threshold=[]
)

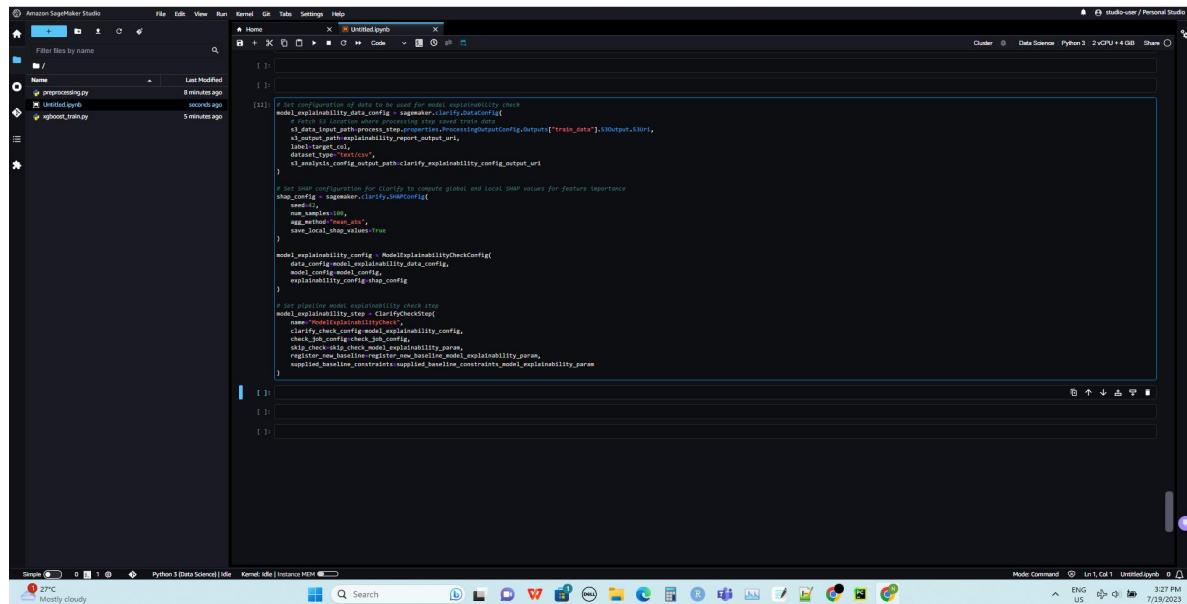
# Set up model predictions configuration to get binary labels from probabilities
model_predictions_config = sagemaker_clarify.ModelPredictionsConfig(probability_threshold=0.5)

model_bias_check_config = sagemaker_clarify.BiasCheckConfig(
    check_step_name="ClarifyBiasStep",
    check_step=s3_data_input_path,
    data_bias_config=model_data_config,
    model_config=model_config,
    model_predictions_config=model_predictions_config,
    methods=["DPPL"]
)

# Set up clarify check step
model_bias_check_step = ClarifyCheckStep(
    name="ModelBiasCheckStep",
    check_step=s3_data_input_path,
    check_job_config=check_job_config,
    check_step_name="ClarifyBiasStep",
    s3_output_uris=[clarify_bias_config.output_uri],
    register_new_baseline=True,
    supplied_baseline_constraints=supplied_baseline_constraints.supplied_baseline_constraints.model_bias_params
)

```

The below code is for model explainability check which provides insights such as feature importance.



```

# Set configuration of data to be used for model explainability check
model_explainability_data_config = sagemaker_clarify.ModelDataConfig(
    s3_data_input_path=processing_step.properties.ProcessingOutputConfig["train_data"].S3Output.S3Uri,
    s3_data_output_report_output_uri=processing_step.properties.ProcessingOutputConfig["clarify_explainability_report"].S3Output.S3Uri,
    label_target_col="label",
    dataset_type="text/csv",
    s3_analytics_config_output_path=clarify_explainability_config.output_uri
)

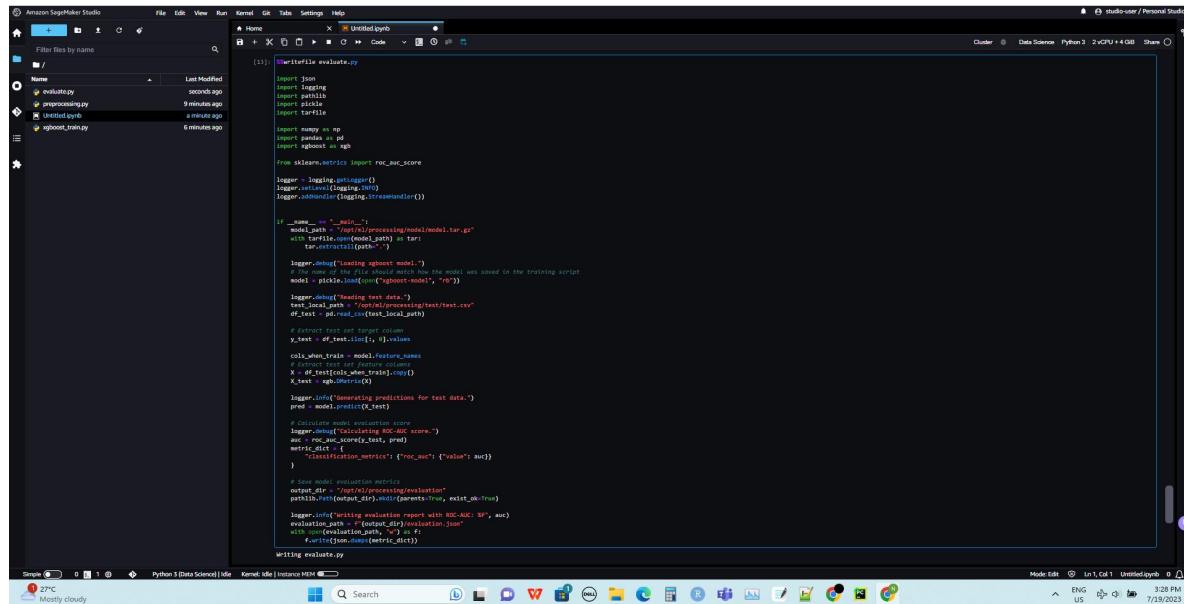
# Set SHAP configuration for Clarify to compute global and local SHAP values for feature importance
shap_config = sagemaker_clarify.ShapConfig(
    num_samples=1000,
    save_local_shap_values=False,
    save_global_shap_values=True
)

model_explainability_config = ModelExplainabilityCheckConfig(
    data_config=model_explainability_data_config,
    model_config=model_config,
    explainability_config=shap_config
)

# Set pipeline model explainability check step
model_explainability_step = ClarifyCheckStep(
    name="ModelExplainabilityStep",
    check_step=s3_data_input_path,
    clarify_check_config=model_explainability_config,
    check_job_config=check_job_config,
    check_step_name="ClarifyExplainabilityStep",
    s3_output_uris=[model_explainability_config.output_uri],
    register_new_baseline=True,
    supplied_baseline_constraints=supplied_baseline_constraints.supplied_baseline_constraints.model_explainability_params
)

```

The below code is used for the following: before model is to be registered and deployed it should be verified if the model is performing better hence here we calculate AUC-ROC value on test data set only after it meets a set threshold value, model is accepted.



```

# This script is used to evaluate the trained model
# It takes the trained model and test data as input
# and outputs the evaluation metrics

import json
import logging
import os
import pickle
import tarfile
import time
import warnings
import numpy as np
import pandas as pd
import s3fs
import sys
from sklearn.metrics import roc_auc_score
import logging, argparse
logger=logging.getLogger()
logger.setLevel(logging.INFO)
logger.addHandler(logging.StreamHandler())
model_name = "model.tar.gz"
model_path = f"/opt/ml/processing/model/{model_name}"
with tarfile.open(model_path) as tar:
    tar.extractall(path=f"/opt/ml/processing/test")

logger.debug(f"Loading {model_name} model")
# A check of the file should match how the model was saved in the training script
model = pickle.load(open(f"/opt/ml/processing/model/{model_name}", "rb"))

logger.debug("Reading test data")
test_csv_file = f"/opt/ml/processing/test/test.csv"
df_test = pd.read_csv(test_csv_file)
X_test = df_test.drop(['target'], axis=1)
y_test = df_test['target'].values

cols_to_drop = model.feature_names
X_test = X_test[X_test.columns.difference(cols_to_drop)]
X_test = X_test.dropna()
X_test = np.array(X_test)

logger.info("Generating predictions for test data")
pred = model.predict(X_test)

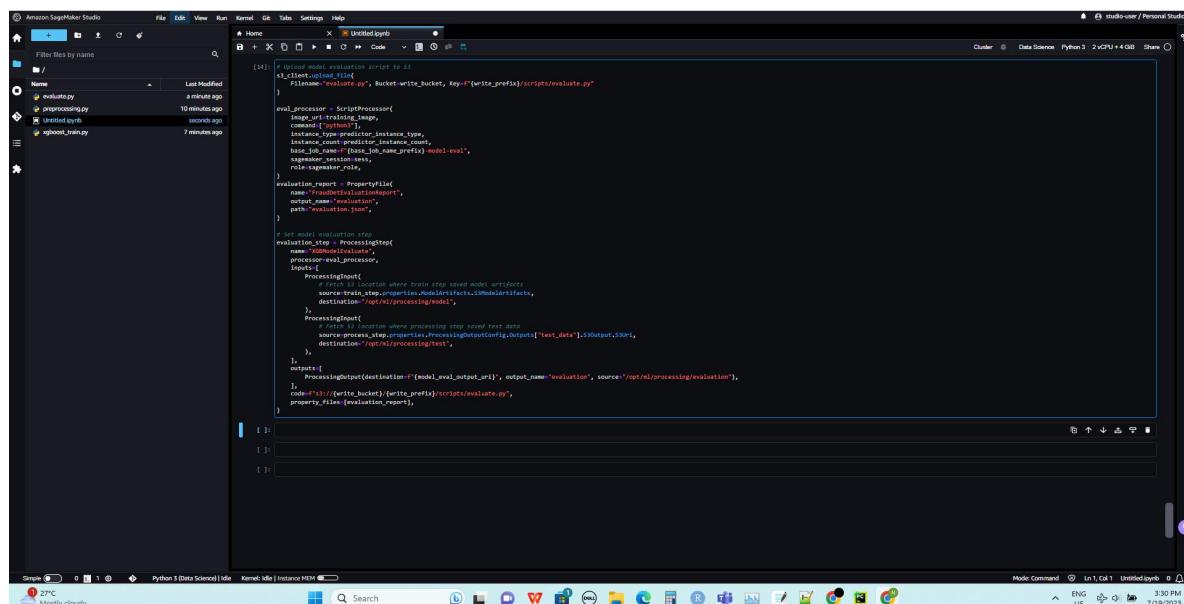
# Calculate model evaluation score
logger.info("Calculating ROC-AUC score")
auc = roc_auc_score(y_test, pred)
metric_dict = {
    "Classification_Accuracy": {"value": auc}
}

# Save model evaluation metrics
output_dir = f"/opt/ml/processing/evaluation"
path_to_model_evaluation_report = os.path.join(output_dir, "model_evaluation_report")
logger.info(f"Creating evaluation report with ROC-AUC: {auc}")
evaluation_path = f"{output_dir}/evaluation.json"
with open(evaluation_path, "w") as f:
    f.write(json.dumps(metric_dict))

Writing evaluate.py

```

The below code instantiate processor and SageMaker pipeline step to run evaluation script. Since we have write custom evaluation script we use ScriptProcessor. The Processing step takes following arguments: processor, s3 bucket location of test data set, model artifacts, output location to store evaluation results. Additionally we use property_files argument to store information from output of processing step as json file which later will be used in conditional step.



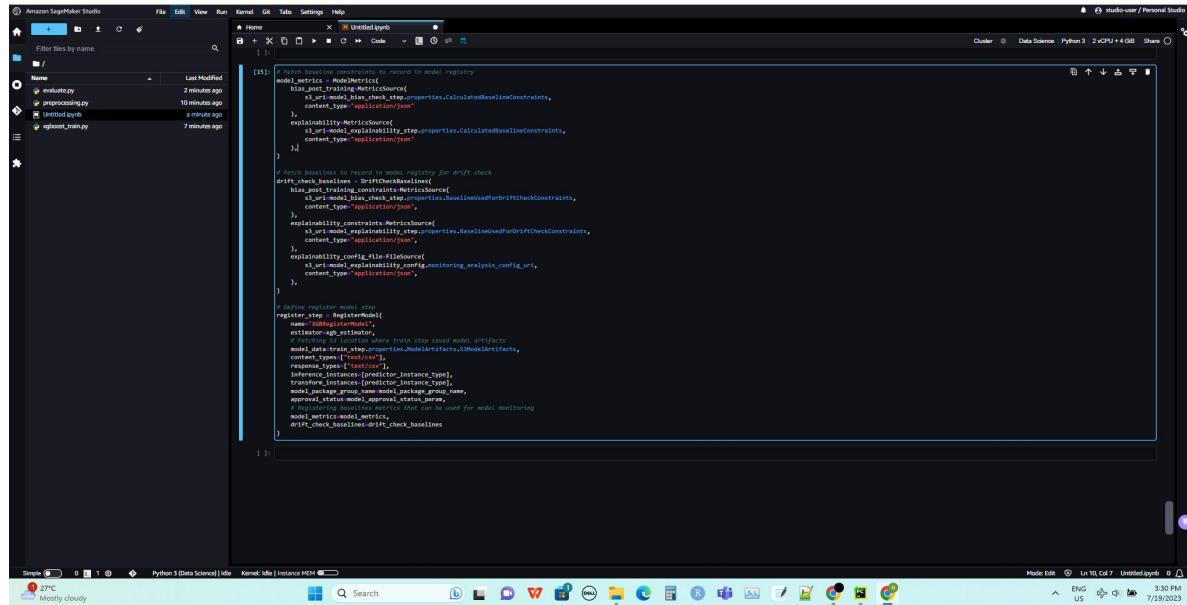
```

# Upload model_evaluation script to S3
s3_client.upload_file(
    Filename="evaluate.py",
    Bucket=write_bucket,
    Key=f"write_prefix/scripts/evaluate.py"
)

eval_processor = ScriptProcessor(
    image_uri="training_image",
    role=role,
    instance_type="ml.m5.xlarge",
    instance_count=1,
    base_job_name="baseline_ml_m5xlarge",
    sagemaker_session=sagemaker.Session(),
    role=sagemaker.get_execution_role(),
    volume_size=50
)
evaluator_report = PropertyFile(
    name="ModelEvaluationReport",
    output_name="evaluation",
    path="evaluation.json",
)
processor_eval_processor = ProcessorStep(
    name="processor_eval_processor",
    inputs=[],
    ProcessingInput(
        # Data location where final step need model artifacts
        # and state properties: modelArtifacts, jobName, artifacts,
        destination=f"/opt/ml/processing/model"
    ),
    ProcessingInput(
        # Data location where processing step need test data
        # and process step properties: inputData, outputConfig.outputs["text_data"], destination,
        destination=f"/opt/ml/processing/test"
    ),
    outputs=[],
    outputArtifact(destination=f"model_eval_output_url", output_name="evaluation", source=f"/opt/ml/processing/evaluation"),
    code=f"#!/usr/bin/python3 /opt/ml/processing/scripts/evaluate.py",
    property_file(evaluation_report)
)

```

The following codes sets up model registry step. Model registry is used to create model catalogs, manage model versions which can be used to selectively deploy specific model. It uses model_metrics and drift_check_baselines calculated in previous steps using which it configures the baselines associated with the model so they can be used in drift checks and model monitoring jobs.

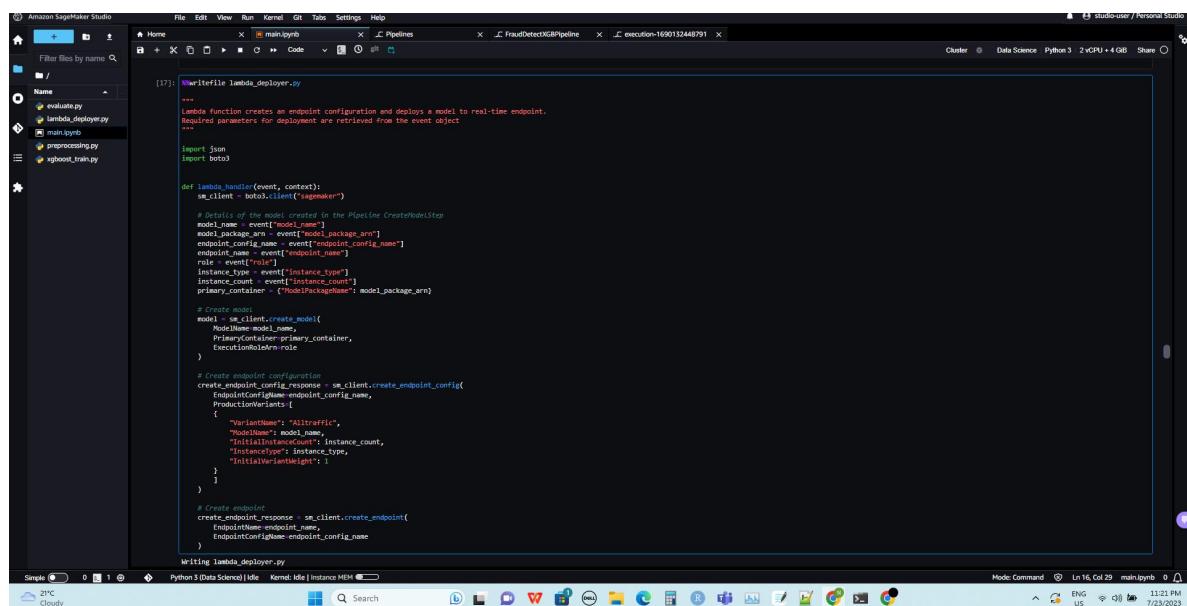


```
[1]: # Fetch metrics source to record in model registry
model_metrics = ModelMetrics(
    bias_post_training_step.properties.calculatedBaselineConstraints,
    content_type='application/json'
)
explainability_config_file_source(
    s3_uris=ModelExplainabilityStep.properties.calculatedBaselineConstraints,
    content_type='application/json'
)

# Fetch baselines to record in model_registry for drift check
drift_check_constraints = DriftCheckConstraints(
    bias_post_training_constraints.MetricSource(
        s3_uris=ModelBiasStep.properties.baselineUsedForDriftCheckConstraints,
        content_type='application/json'
    ),
    explainability_constraints.MetricSource(
        s3_uris=ModelExplainabilityStep.properties.baselineUsedForDriftCheckConstraints,
        content_type='application/json'
    ),
    explainability_config_file_source(
        s3_uris=ModelExplainabilityConfigMonitoringAnalysisConfigUri,
        content_type='application/json'
    )
)

# Register step
register_step = RegisterModel(
    name="ModelRegisterModel",
    instance_type='ml.m5.xlarge',
    role='arn:aws:iam::123456789012:root',
    instance_count=1,
    container_def=ContainerDefinition(
        entry_point='train.py',
        model_data='s3://my-bucket/model-data.tar.gz',
        container_name='my-model',
        instance_type='ml.m5.xlarge',
        transform_instances='predictor_instance_type',
        model_approval_status='Approved',
        approval_status='Approved',
        approval_status_params={},
        # approval_status_params: parameters that can be used for model monitoring
        model_metrics=model_metrics,
        drift_check_baselines=drift_check_baselines
    )
)
```

Below code is used to deploy model for inference using lambda step function. Lambda step function provide integration with AWS Lambda. Lambda step function is for lightweight model deployment and with maximum run time of 10 min. Here we use high-level SageMaker Python SDK which has a Lambda helper convenience class that can be used to create a new Lambda function along with other code defining your pipeline. This codes defines Lambda handler function and a script to take model attributes, such as model name, and deploys to a real-time endpoint.



```
[1]: # Write file lambda_deployer.py
"""
Lambda Function creates an endpoint configuration and deploys a model to real-time endpoint.
Required parameters for deployment are retrieved from the event object
"""

import json
import boto3
from botocore import UNSIGNED
from botocore.client import Config

def lambda_handler(event, context):
    # Create client for SageMaker in the Pipeline CreateModelStep
    sm_client = boto3.client("sagemaker")
    # Required parameters for deployment are retrieved from the event object
    model_name = event["model_name"]
    model_package_arn = event["model_package_arn"]
    endpoint_config_name = event["endpoint_config_name"]
    endpoint_name = event["endpoint_name"]
    role = event["role"]
    instance_type = event["instance_type"]
    instance_count = event["instance_count"]
    primary_container = {"ModelPackageName": model_package_arn}

    # Create model
    model = sm_client.create_model(
        ModelName=model_name,
        PrimaryContainer=primary_container,
        ExecutionRoleArn=role
    )

    # Create endpoint configuration
    create_endpoint_config_response = sm_client.create_endpoint_config(
        EndpointConfigName=endpoint_config_name,
        ProductionVariants=[
            {
                "VariantName": "altruistic",
                "ModelName": model_name,
                "InitialInstanceCount": instance_count,
                "InstanceType": instance_type,
                "InitialVariantWeight": 1
            }
        ]
    )

    # Create endpoint
    create_endpoint_response = sm_client.create_endpoint(
        EndpointName=endpoint_name,
        EndpointConfigName=endpoint_config_name
    )

Writing lambda_deployer.py
```

The below code create LambdaStep which takes all parameters such as model, endpoint name, and deployment instance type and count are provided using the inputs argument.

```

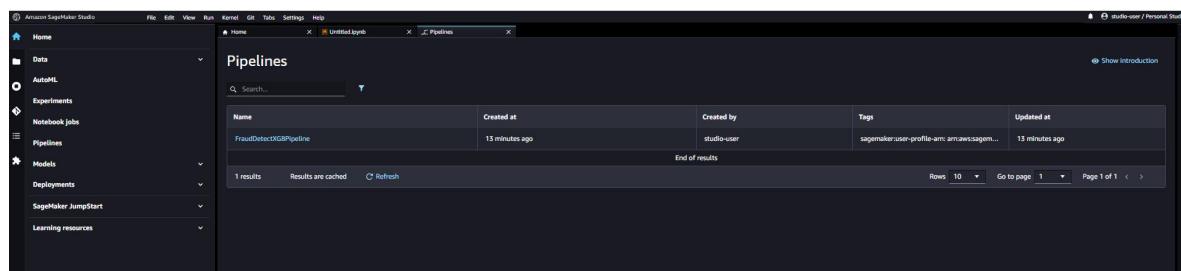
1+| #!/usr/bin/env python3
2+|
3+| # This function can be used to create the Lambda function required in the Lambda step
4+|
5+| Function_name = "sgenerator-fraud-det-demo-lambda-step"
6+| Func = LambdaFunc(
7+|     function_code=FileFunctionCode("lambda_deployer.py"),
8+|     role="arn:aws:iam::123456789012:role/sagemaker-lambda-deployer-lambda_handler",
9+|     timeout=10,
10+|     memory_size=300,
11+| )
12+|
13+| # The inputs used in the Lambda handler are passed through the Inputs argument in the
14+| # LambdaStep and retrieved via the event object within the LambdaHandler function
15+|
16+| LambdaDeployStep = LambdaStep(
17+|     name="LambdaDeployStep",
18+|     lambda_func=Func,
19+|     inputs=ModelInputs(
20+|         model_name="pipeline_model_name",
21+|         endpoint_config_name="fraud_endpoints",
22+|         instance_type="ml.m5.xlarge",
23+|         instance_count=1
24+|     ),
25+|     lambda_package_arn="arn:aws:lambda:us-west-2:123456789012:SGeneratorFraudDetectorLambdaStep"
26+| )
27+|
28+| LambdaDeployStep.register_step_input(PropertyStepInput("model_package_arn"))
29+| LambdaDeployStep.register_step_input(PropertyStepInput("instance_type"))
30+| LambdaDeployStep.register_step_input(PropertyStepInput("instance_count"))
31+|
32+|
33+|
34+|
35+|
36+|
37+|
38+|
39+|
40+|
41+|
42+|
43+|
44+|
45+|
46+|
47+|
48+|
49+|
50+|
51+|
52+|
53+|
54+|
55+|
56+|
57+|
58+|
59+|
60+|
61+|
62+|
63+|
64+|
65+|
66+|
67+|
68+|
69+|
70+|
71+|
72+|
73+|
74+|
75+|
76+|
77+|
78+|
79+|
80+|
81+|
82+|
83+|
84+|
85+|
86+|
87+|
88+|
89+|
90+|
91+|
92+|
93+|
94+|
95+|
96+|
97+|
98+|
99+|
100+|
101+|
102+|
103+|
104+|
105+|
106+|
107+|
108+|
109+|
110+|
111+|
112+|
113+|
114+|
115+|
116+|
117+|
118+|
119+|
120+|
121+|
122+|
123+|
124+|
125+|
126+|
127+|
128+|
129+|
130+|
131+|
132+|
133+|
134+|
135+|
136+|
137+|
138+|
139+|
140+|
141+|
142+|
143+|
144+|
145+|
146+|
147+|
148+|
149+|
150+|
151+|
152+|
153+|
154+|
155+|
156+|
157+|
158+|
159+|
160+|
161+|
162+|
163+|
164+|
165+|
166+|
167+|
168+|
169+|
170+|
171+|
172+|
173+|
174+|
175+|
176+|
177+|
178+|
179+|
180+|
181+|
182+|
183+|
184+|
185+|
186+|
187+|
188+|
189+|
190+|
191+|
192+|
193+|
194+|
195+|
196+|
197+|
198+|
199+|
200+|
201+|
202+|
203+|
204+|
205+|
206+|
207+|
208+|
209+|
210+|
211+|
212+|
213+|
214+|
215+|
216+|
217+|
218+|
219+|
220+|
221+|
222+|
223+|
224+|
225+|
226+|
227+|
228+|
229+|
230+|
231+|
232+|
233+|
234+|
235+|
236+|
237+|
238+|
239+|
240+|
241+|
242+|
243+|
244+|
245+|
246+|
247+|
248+|
249+|
250+|
251+|
252+|
253+|
254+|
255+|
256+|
257+|
258+|
259+|
260+|
261+|
262+|
263+|
264+|
265+|
266+|
267+|
268+|
269+|
270+|
271+|
272+|
273+|
274+|
275+|
276+|
277+|
278+|
279+|
280+|
281+|
282+|
283+|
284+|
285+|
286+|
287+|
288+|
289+|
290+|
291+|
292+|
293+|
294+|
295+|
296+|
297+|
298+|
299+|
300+|
301+|
302+|
303+|
304+|
305+|
306+|
307+|
308+|
309+|
310+|
311+|
312+|
313+|
314+|
315+|
316+|
317+|
318+|
319+|
320+|
321+|
322+|
323+|
324+|
325+|
326+|
327+|
328+|
329+|
330+|
331+|
332+|
333+|
334+|
335+|
336+|
337+|
338+|
339+|
340+|
341+|
342+|
343+|
344+|
345+|
346+|
347+|
348+|
349+|
350+|
351+|
352+|
353+|
354+|
355+|
356+|
357+|
358+|
359+|
360+|
361+|
362+|
363+|
364+|
365+|
366+|
367+|
368+|
369+|
370+|
371+|
372+|
373+|
374+|
375+|
376+|
377+|
378+|
379+|
380+|
381+|
382+|
383+|
384+|
385+|
386+|
387+|
388+|
389+|
390+|
391+|
392+|
393+|
394+|
395+|
396+|
397+|
398+|
399+|
400+|
401+|
402+|
403+|
404+|
405+|
406+|
407+|
408+|
409+|
410+|
411+|
412+|
413+|
414+|
415+|
416+|
417+|
418+|
419+|
420+|
421+|
422+|
423+|
424+|
425+|
426+|
427+|
428+|
429+|
430+|
431+|
432+|
433+|
434+|
435+|
436+|
437+|
438+|
439+|
440+|
441+|
442+|
443+|
444+|
445+|
446+|
447+|
448+|
449+|
450+|
451+|
452+|
453+|
454+|
455+|
456+|
457+|
458+|
459+|
460+|
461+|
462+|
463+|
464+|
465+|
466+|
467+|
468+|
469+|
470+|
471+|
472+|
473+|
474+|
475+|
476+|
477+|
478+|
479+|
480+|
481+|
482+|
483+|
484+|
485+|
486+|
487+|
488+|
489+|
490+|
491+|
492+|
493+|
494+|
495+|
496+|
497+|
498+|
499+|
500+|
501+|
502+|
503+|
504+|
505+|
506+|
507+|
508+|
509+|
510+|
511+|
512+|
513+|
514+|
515+|
516+|
517+|
518+|
519+|
520+|
521+|
522+|
523+|
524+|
525+|
526+|
527+|
528+|
529+|
530+|
531+|
532+|
533+|
534+|
535+|
536+|
537+|
538+|
539+|
540+|
541+|
542+|
543+|
544+|
545+|
546+|
547+|
548+|
549+|
550+|
551+|
552+|
553+|
554+|
555+|
556+|
557+|
558+|
559+|
560+|
561+|
562+|
563+|
564+|
565+|
566+|
567+|
568+|
569+|
570+|
571+|
572+|
573+|
574+|
575+|
576+|
577+|
578+|
579+|
580+|
581+|
582+|
583+|
584+|
585+|
586+|
587+|
588+|
589+|
590+|
591+|
592+|
593+|
594+|
595+|
596+|
597+|
598+|
599+|
600+|
601+|
602+|
603+|
604+|
605+|
606+|
607+|
608+|
609+|
610+|
611+|
612+|
613+|
614+|
615+|
616+|
617+|
618+|
619+|
620+|
621+|
622+|
623+|
624+|
625+|
626+|
627+|
628+|
629+|
630+|
631+|
632+|
633+|
634+|
635+|
636+|
637+|
638+|
639+|
640+|
641+|
642+|
643+|
644+|
645+|
646+|
647+|
648+|
649+|
650+|
651+|
652+|
653+|
654+|
655+|
656+|
657+|
658+|
659+|
660+|
661+|
662+|
663+|
664+|
665+|
666+|
667+|
668+|
669+|
670+|
671+|
672+|
673+|
674+|
675+|
676+|
677+|
678+|
679+|
680+|
681+|
682+|
683+|
684+|
685+|
686+|
687+|
688+|
689+|
690+|
691+|
692+|
693+|
694+|
695+|
696+|
697+|
698+|
699+|
700+|
701+|
702+|
703+|
704+|
705+|
706+|
707+|
708+|
709+|
710+|
711+|
712+|
713+|
714+|
715+|
716+|
717+|
718+|
719+|
720+|
721+|
722+|
723+|
724+|
725+|
726+|
727+|
728+|
729+|
730+|
731+|
732+|
733+|
734+|
735+|
736+|
737+|
738+|
739+|
740+|
741+|
742+|
743+|
744+|
745+|
746+|
747+|
748+|
749+|
750+|
751+|
752+|
753+|
754+|
755+|
756+|
757+|
758+|
759+|
760+|
761+|
762+|
763+|
764+|
765+|
766+|
767+|
768+|
769+|
770+|
771+|
772+|
773+|
774+|
775+|
776+|
777+|
778+|
779+|
779+|
780+|
781+|
782+|
783+|
784+|
785+|
786+|
787+|
788+|
789+|
790+|
791+|
792+|
793+|
794+|
795+|
796+|
797+|
798+|
799+|
800+|
801+|
802+|
803+|
804+|
805+|
806+|
807+|
808+|
809+|
810+|
811+|
812+|
813+|
814+|
815+|
816+|
817+|
818+|
819+|
820+|
821+|
822+|
823+|
824+|
825+|
826+|
827+|
828+|
829+|
830+|
831+|
832+|
833+|
834+|
835+|
836+|
837+|
838+|
839+|
840+|
841+|
842+|
843+|
844+|
845+|
846+|
847+|
848+|
849+|
850+|
851+|
852+|
853+|
854+|
855+|
856+|
857+|
858+|
859+|
860+|
861+|
862+|
863+|
864+|
865+|
866+|
867+|
868+|
869+|
870+|
871+|
872+|
873+|
874+|
875+|
876+|
877+|
878+|
879+|
880+|
881+|
882+|
883+|
884+|
885+|
886+|
887+|
888+|
889+|
890+|
891+|
892+|
893+|
894+|
895+|
896+|
897+|
898+|
899+|
900+|
901+|
902+|
903+|
904+|
905+|
906+|
907+|
908+|
909+|
910+|
911+|
912+|
913+|
914+|
915+|
916+|
917+|
918+|
919+|
920+|
921+|
922+|
923+|
924+|
925+|
926+|
927+|
928+|
929+|
930+|
931+|
932+|
933+|
934+|
935+|
936+|
937+|
938+|
939+|
940+|
941+|
942+|
943+|
944+|
945+|
946+|
947+|
948+|
949+|
950+|
951+|
952+|
953+|
954+|
955+|
956+|
957+|
958+|
959+|
960+|
961+|
962+|
963+|
964+|
965+|
966+|
967+|
968+|
969+|
970+|
971+|
972+|
973+|
974+|
975+|
976+|
977+|
978+|
979+|
980+|
981+|
982+|
983+|
984+|
985+|
986+|
987+|
988+|
989+|
990+|
991+|
992+|
993+|
994+|
995+|
996+|
997+|
998+|
999+|
1000+|
1001+|
1002+|
1003+|
1004+|
1005+|
1006+|
1007+|
1008+|
1009+|
1010+|
1011+|
1012+|
1013+|
1014+|
1015+|
1016+|
1017+|
1018+|
1019+|
1020+|
1021+|
1022+|
1023+|
1024+|
1025+|
1026+|
1027+|
1028+|
1029+|
1030+|
1031+|
1032+|
1033+|
1034+|
1035+|
1036+|
1037+|
1038+|
1039+|
1040+|
1041+|
1042+|
1043+|
1044+|
1045+|
1046+|
1047+|
1048+|
1049+|
1050+|
1051+|
1052+|
1053+|
1054+|
1055+|
1056+|
1057+|
1058+|
1059+|
1060+|
1061+|
1062+|
1063+|
1064+|
1065+|
1066+|
1067+|
1068+|
1069+|
1070+|
1071+|
1072+|
1073+|
1074+|
1075+|
1076+|
1077+|
1078+|
1079+|
1080+|
1081+|
1082+|
1083+|
1084+|
1085+|
1086+|
1087+|
1088+|
1089+|
1090+|
1091+|
1092+|
1093+|
1094+|
1095+|
1096+|
1097+|
1098+|
1099+|
1100+|
1101+|
1102+|
1103+|
1104+|
1105+|
1106+|
1107+|
1108+|
1109+|
1110+|
1111+|
1112+|
1113+|
1114+|
1115+|
1116+|
1117+|
1118+|
1119+|
1120+|
1121+|
1122+|
1123+|
1124+|
1125+|
1126+|
1127+|
1128+|
1129+|
1130+|
1131+|
1132+|
1133+|
1134+|
1135+|
1136+|
1137+|
1138+|
1139+|
1140+|
1141+|
1142+|
1143+|
1144+|
1145+|
1146+|
1147+|
1148+|
1149+|
1150+|
1151+|
1152+|
1153+|
1154+|
1155+|
1156+|
1157+|
1158+|
1159+|
1160+|
1161+|
1162+|
1163+|
1164+|
1165+|
1166+|
1167+|
1168+|
1169+|
1170+|
1171+|
1172+|
1173+|
1174+|
1175+|
1176+|
1177+|
1178+|
1179+|
1180+|
1181+|
1182+|
1183+|
1184+|
1185+|
1186+|
1187+|
1188+|
1189+|
1190+|
1191+|
1192+|
1193+|
1194+|
1195+|
1196+|
1197+|
1198+|
1199+|
1200+|
1201+|
1202+|
1203+|
1204+|
1205+|
1206+|
1207+|
1208+|
1209+|
1210+|
1211+|
1212+|
1213+|
1214+|
1215+|
1216+|
1217+|
1218+|
1219+|
1220+|
1221+|
1222+|
1223+|
1224+|
1225+|
1226+|
1227+|
1228+|
1229+|
1230+|
1231+|
1232+|
1233+|
1234+|
1235+|
1236+|
1237+|
1238+|
1239+|
1240+|
1241+|
1242+|
1243+|
1244+|
1245+|
1246+|
1247+|
1248+|
1249+|
1250+|
1251+|
1252+|
1253+|
1254+|
1255+|
1256+|
1257+|
1258+|
1259+|
1260+|
1261+|
1262+|
1263+|
1264+|
1265+|
1266+|
1267+|
1268+|
1269+|
1270+|
1271+|
1272+|
1273+|
1274+|
1275+|
1276+|
1277+|
1278+|
1279+|
1280+|
1281+|
1282+|
1283+|
1284+|
1285+|
1286+|
1287+|
1288+|
1289+|
1290+|
1291+|
1292+|
1293+|
1294+|
1295+|
1296+|
1297+|
1298+|
1299+|
1300+|
1301+|
1302+|
1303+|
1304+|
1305+|
1306+|
1307+|
1308+|
1309+|
1310+|
1311+|
1312+|
1313+|
1314+|
1315+|
1316+|
1317+|
1318+|
1319+|
1320+|
1321+|
1322+|
1323+|
1324+|
1325+|
1326+|
1327+|
1328+|
1329+|
1330+|
1331+|
1332+|
1333+|
1334+|
1335+|
1336+|
1337+|
1338+|
1339+|
1340+|
1341+|
1342+|
1343+|
1344+|
1345+|
1346+|
1347+|
1348+|
1349+|
1350+|
1351+|
1352+|
1353+|
1354+|
1355+|
1356+|
1357+|
1358+|
1359+|
1360+|
1361+|
1362+|
1363+|
1364+|
1365+|
1366+|
1367+|
1368+|
1369+|
1370+|
1371+|
1372+|
1373+|
1374+|
1375+|
1376+|
1377+|
1378+|
1379+|
1380+|
1381+|
1382+|
1383+|
1384+|
1385+|
1386+|
1387+|
1388+|
1389+|
1390+|
1391+|
1392+|
1393+|
1394+|
1395+|
1396+|
1397+|
1398+|
1399+|
1400+|
1401+|
1402+|
1403+|
1404+|
1405+|
1406+|
1407+|
1408+|
1409+|
1410+|
1411+|
1412+|
1413+|
1414+|
1415+|
1416+|
1417+|
1418+|
1419+|
1420+|
1421+|
1422+|
1423+|
1424+|
1425+|
1426+|
1427+|
1428+|
1429+|
1430+|
1431+|
1432+|
1433+|
1434+|
1435+|
1436+|
1437+|
1438+|
1439+|
1440+|
1441+|
1442+|
1443+|
1444+|
1445+|
1446+|
1447+|
1448+|
1449+|
1450+|
1451+|
1452+|
1453+|
1454+|
1455+|
1456+|
1457+|
1458+|
1459+|
1460+|
1461+|
1462+|
1463+|
1464+|
1465+|
1466+|
1467+|
1468+|
1469+|
1470+|
1471+|
1472+|
1473+|
1474+|
1475+|
1476+|
1477+|
1478+|
1479+|
1480+|
1481+|
1482+|
1483+|
1484+|
1485+|
1486+|
1487+|
1488+|
1489+|
1490+|
1491+|
1492+|
1493+|
1494+|
1495+|
1496+|
1497+|
1498+|
1499+|
1500+|
1501+|
1502+|
1503+|
1504+|
1505+|
1506+|
1507+|
1508+|
1509+|
1510+|
1511+|
1512+|
1513+|
1514+|
1515+|
1516+|
1517+|
1518+|
1519+|
1520+|
1521+|
1522+|
1523+|
1524+|
1525+|
1526+|
1527+|
1528+|
1529+|
1530+|
1531+|
1532+|
1533+|
1534+|
1535+|
1536+|
1537+|
1538+|
1539+|
1540+|
1541+|
1542+|
1543+|
1544+|
1545+|
1546+|
1547+|
1548+|
1549+|
1550+|
1551+|
1552+|
1553+|
1554+|
1555+|
1556+|
1557+|
1558+|
1559+|
1560+|
1561+|
1562+|
1563+|
1564+|
1565+|
1566+|
1567+|
1568+|
1569+|
1570+|
1571+|
1572+|
1573+|
1574+|
1575+|
1576+|
1577+|
1578+|
1579+|
1580+|
1581+|
1582+|
1583+|
1584+|
1585+|
1586+|
1587+|
1588+|
1589+|
1590+|
1591+|
1592+|
1593+|
1594+|
1595+|
1596+|
1597+|
1598+|
1599+|
1600+|
1601+|
1602+|
1603+|
1604+|
1605+|
1606+|
1607+|
1608+|
1609+|
1610+|
1611+|
1612+|
1613+|
1614+|
1615+|
1616+|
1617+|
1618+|
1619+|
1620+|
1621+|
1622+|
1623+|
1624+|
1625+|
1626+|
1627+|
1628+|
1629+|
1630+|
1631+|
1632+|
1633+|
1634+|
1635+|
1636+|
1637+|
1638+|
1639+|
1640+|
1641+|
1642+|
1643+|
1644+|
1645+|
1646+|
1647+|
1648+|
1649+|
1650+|
1651+|
1652+|
1653+|
1654+|
1655+|
1656+|
1657+|
1658+|
1659+|
1660+|
1661+|
1662+|
1663+|
1664+|
1665+|
1666+|
1667+|
1668+|
1669+|
1670+|
1671+|
1672+|
1673+|
1674+|
1675+|
1676+|
1677+|
1678+|
1679+|
1680+|
1681+|
1682+|
1683+|
1684+|
1685+|
1686+|
1687+|
1688+|
1689+|
1690+|
1691+|
1692+|
1693+|
1694+|
1695+|
1696+|
1697+|
1698+|
1699+|
1700+|
1701+|
1702+|
1703+|
1704+|
1705+|
1706+|
1707+|
1708+|
1709+|
1710+|
1711+|
1712+|
1713+|
1714+|
1715+|
1716+|
1717+|
1718+|
1719+|
1720+|
1721+|
1722+|
1723+|
1724+|
1725+|
1726+|
1727+|
1728+|
1729+|
1730+|
1731+|
1732+|
1733+|
1734+|
1735+|
1736+|
1737+|
1738+|
1739+|
1740+|
1741+|
1742+|
1743+|
1744+|
1745+|
1746+|
1747+|
1748+|
1749+|
1750+|
1751+|
1752+|
1753+|
1754+|
1755+|
1756+|
1757+|
1758+|
1759+|
1760+|
1761+|
1762+|
1763+|
1764+|
1765+|
1766+|
1767+|
1768+|
1769+|
1770+|
1771+|
1772+|
1773+|
1774+|
1775+|
1776+|
1777+|
1778+|
1779+|
1780+|
1781+|
1782+|
1783+|
1784+|
1785+|
1786+|
1787+|
1788+|
1789+|
1790+|
1791+|
1792+|
1793+|
1794+|
1795+|
1796+|
1797+|
1798+|
1799+|
1800+|
1801+|
1802+|
1803+|
1804+|
1805+|
1806+|
1807+|
1808+|
1809+|
1810+|
1811+|
1812+|
1813+|
1814+|
1815+|
1816+|
1817+|
1818+|
1819+|
1820+|
1821+|
1822+|
1823+|
1824+|
1825+|
1826+|
1827+|
1828+|
1829+|
1830+|
1831+|
1832+|
1833+|
1834+|
1835+|
1836+|
1837+|
1838+|
1839+|
1840+|
1841+|
1842+|
1843+|
1844+|
1845+|
1846+|
1847+|
1848+|
1849+|
1850+|
1851+|
1852+|
1853+|
1854+|
1855+|
1856+|
1857+|
1858+|
1859+|
1860+|
1861+|
1862+|
1863+|
1864+|
1865+|
1866+|
1867+|
1868+|
1869+|
1870+|
1871+|
1872+|
1873+|
1874+|
1875+|
1876+|
1877+|
1878+|
1879+|
1880+|
1881+|
1882+|
1883+|
1884+|
1885+|
1886+|
1887+|
1888+|
1889+|
1890+|
1891+|
1892+|
1893+|
1894+|
1895+|
1896+|
1897+|
1898+|
1899+|
1900+|
1901+|
1902+|
1903+|
1904+|
1905+|
1906+|
1907+|
1908+|
1909+|
1910+|
1911+|
1912+|
1913+|
1914+|
1915+|
1916+|
1917+|
1918+|
1919+|
1920+|
1921+|
1922+|
1923+|
1924+|
1925+|
1926+|
1927+|
1928+|
1929+|
1930+|
1931+|
1932+|
1933+|
1934+|
1935+|
1936+|
1937+|
1938+|
1939+|
1940+|
1941+|
1942+|
1943+|
1944+|
1945+|
1946+|
1947+|
1948+|
1949+|
1950+|
1951+|
1952+|
1953+|
1954+|
1955+|
1956+|
1957+|
1958+|
1959+|
1960+|
1961+|
1962+|
1963+|
1964+|
1965+|
1966+|
1967+|
1968+|
1969+|
1970+|
1971+|
1972+|
1973+|
1974+|
1975+|
1976+|

```

The below code is used to update pipeline if some pipeline already exists or else create new.

Step 5: Test the pipeline by invoking the endpoint

On the left hand side of the screen of SageMaker studio click-on pipelines and then select the pipeline name.



The below code executes the pipeline.

The screenshot displays two main windows within the Amazon SageMaker Studio interface.

Top Window (Jupyter Notebook):

- Title Bar:** Home, Data, AutoML, Experiments, Notebook jobs, Pipelines.
- Code Editor:** Contains Python code for executing a pipeline:

```
# Execute pipeline
start_pipeline_start(parameters={})
```

```
start_pipeline_start(parameters={})
```

```
skipProfilingCheck=True,
skipModelBiasCheck=True,
skipModelDriftCheck=True,
skipModelIngestibilityCheck=True,
registerNewModelExplainabilityBaseline=True)
```
- Toolbar:** File, Edit, View, Run, Kernel, GR, Tabs, Settings, Help.
- Status Bar:** Cluster 0 Data Science Python 3 2 vCPU + 4 GB.

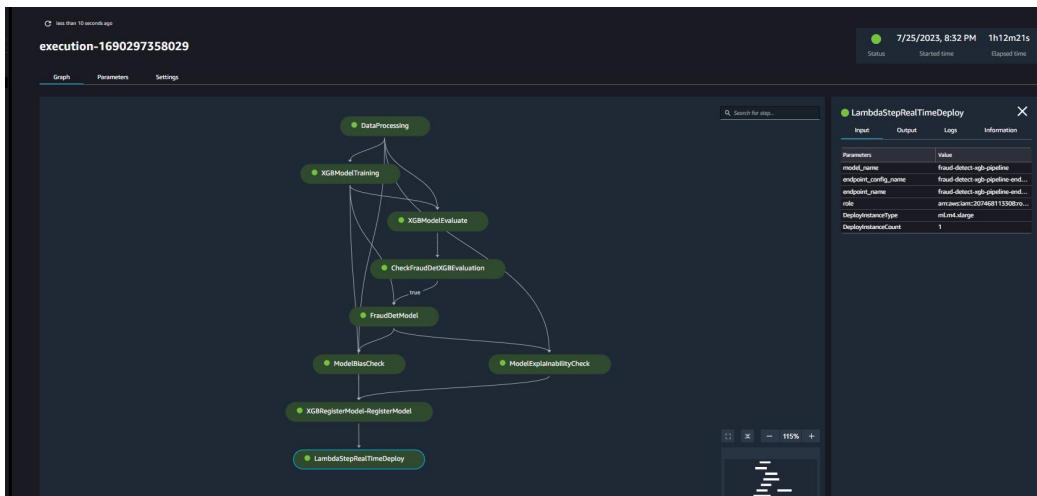
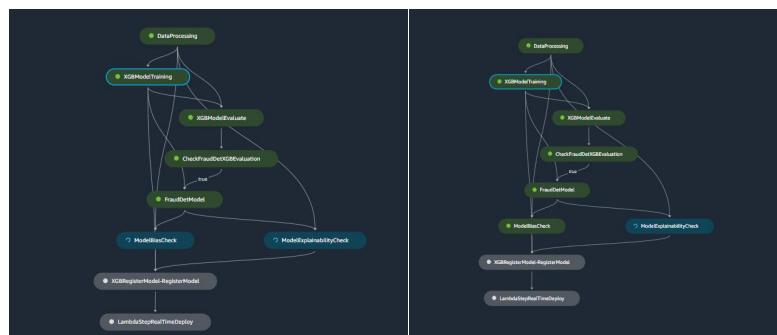
Bottom Window (FraudDetectionPipeline):

- Title Bar:** Home, Data, AutoML, Experiments, Notebook jobs, Pipelines, FraudDetectionPipeline.
- Left Sidebar:** Home, Data, AutoML, Experiments, Notebook jobs, Pipelines, Models, Deployments, SageMaker JumpStart, Learning resources.
- Central Area:** Shows the "FraudDetectionPipeline" graph. The graph consists of several nodes connected by arrows:
 - Nodes include: DataProcessing, XGBoostTraining, XGBoostInference, CheckFraudulentCBBValidation, FraudModel, ModelAccuracyCheck, ModelIngestibilityCheck, ICGBestPracticesRegimeModel, and LambdaIngestionFunction.
 - Connections show a flow from DataProcessing to XGBoostTraining, XGBoostTraining to XGBoostInference, XGBoostInference to CheckFraudulentCBBValidation, CheckFraudulentCBBValidation to FraudModel, FraudModel to ModelAccuracyCheck, FraudModel to ModelIngestibilityCheck, ModelAccuracyCheck to ICGBestPracticesRegimeModel, and ModelIngestibilityCheck to ICGBestPracticesRegimeModel.
- Toolbar:** Run, Stop, Refresh, Save, View, Help, Studio User / Personal State.
- Status Bar:** Studio User / Personal State.

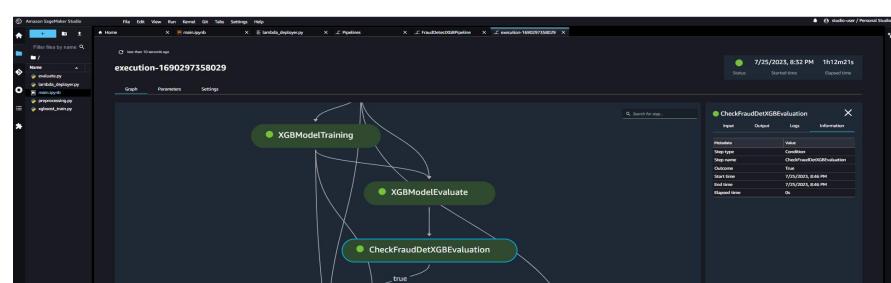
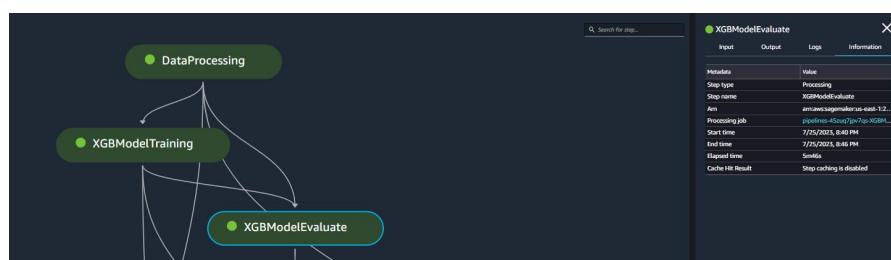
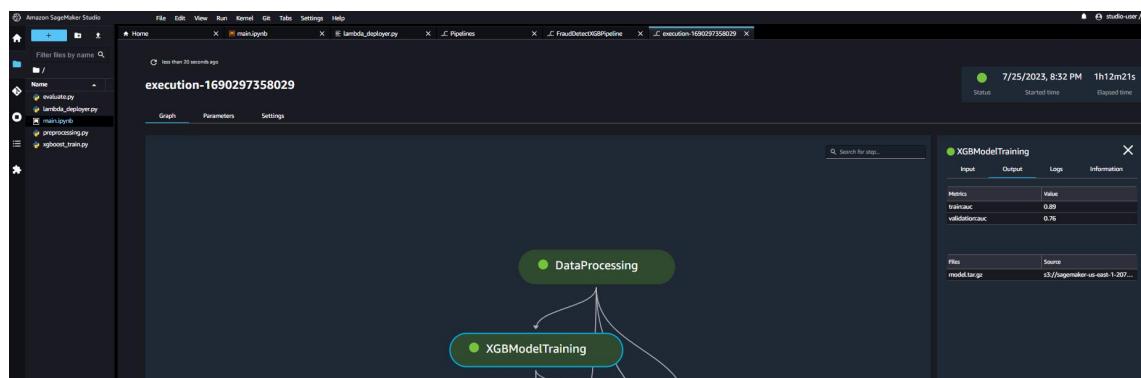
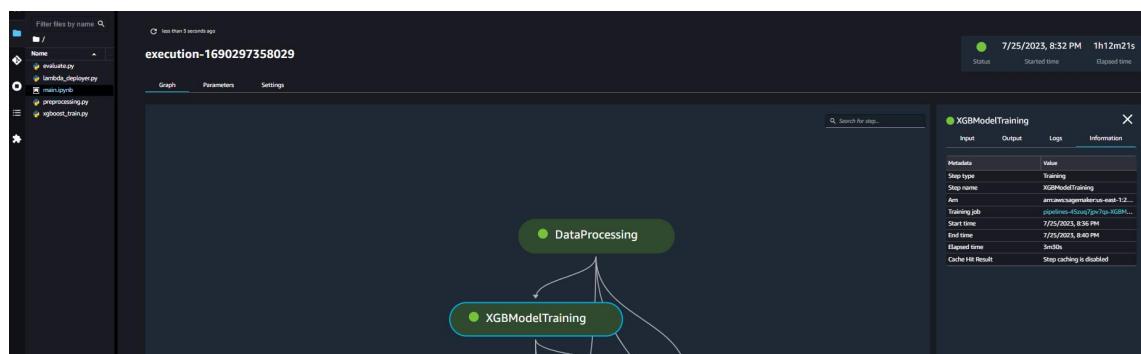
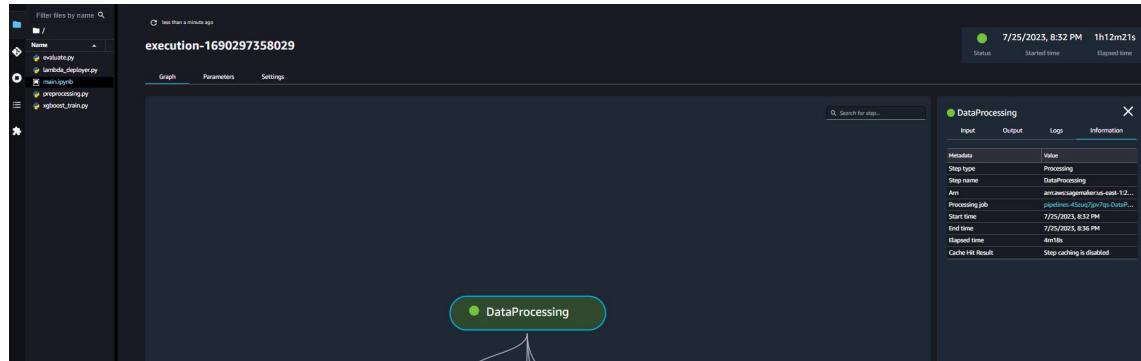
The screenshot shows the 'FraudDetectXGPipeline' configuration page in Amazon SageMaker Studio. It displays a table of parameters with their types and values. Key parameters include:

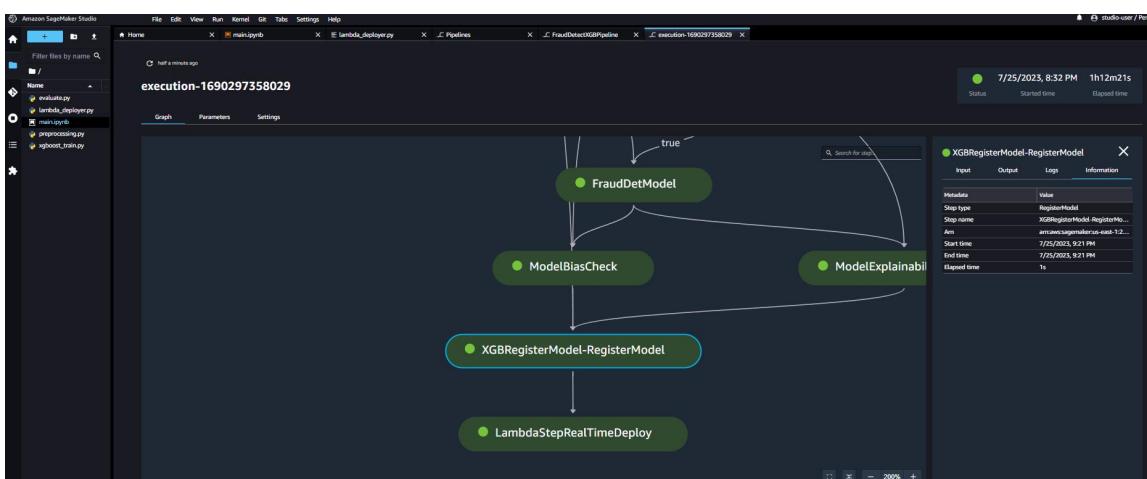
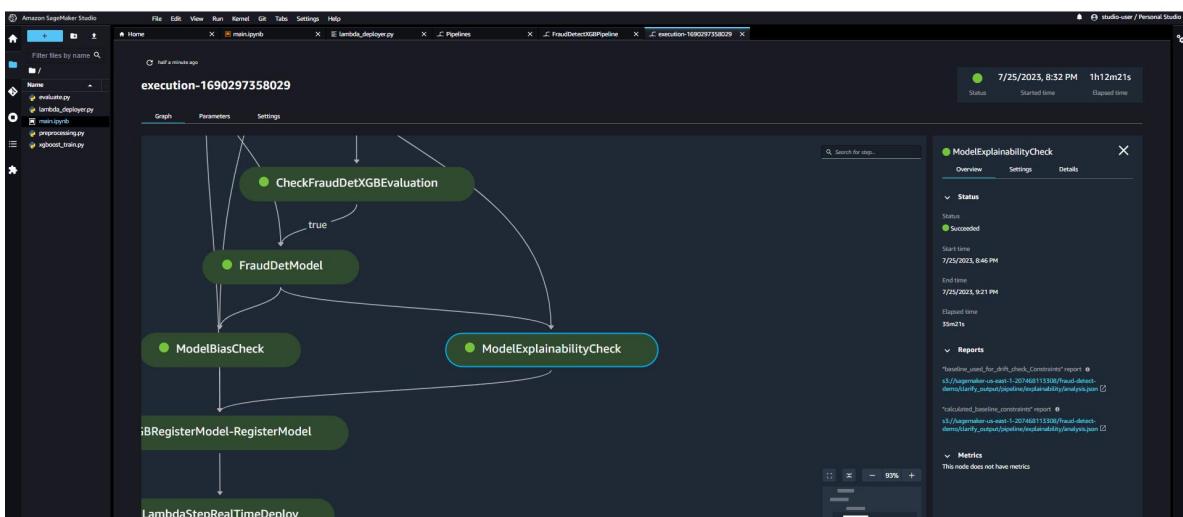
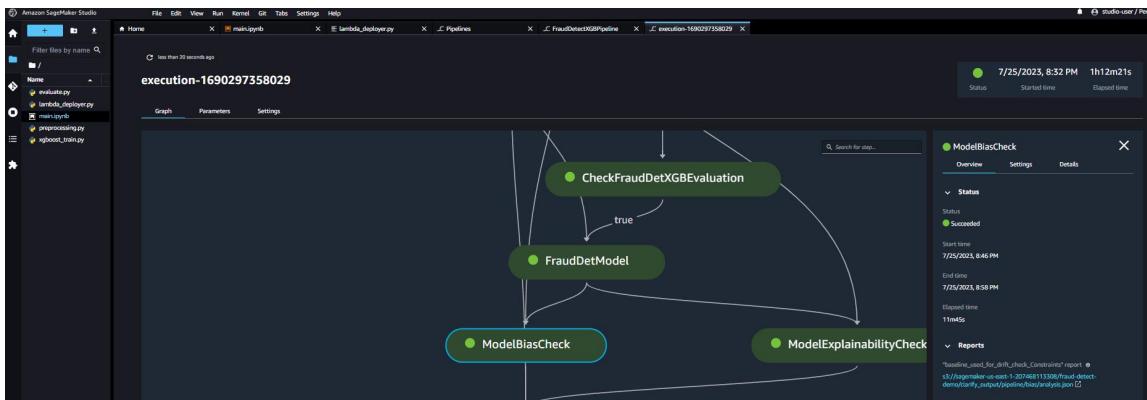
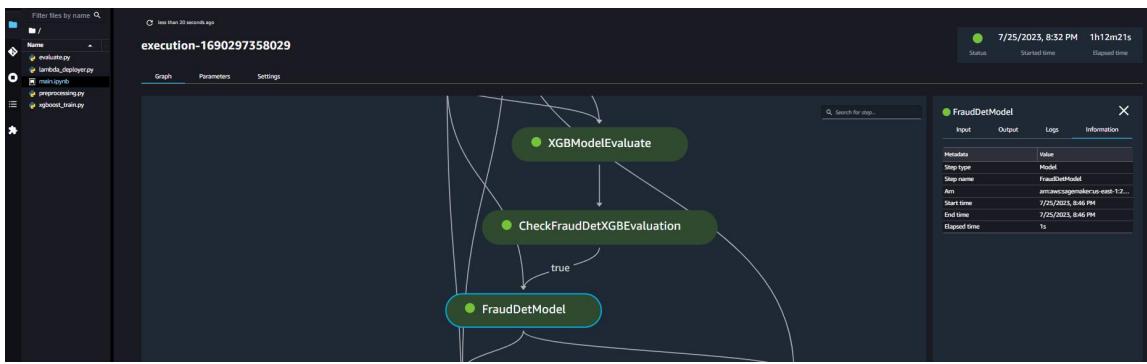
| Parameter | Type | Value |
|---------------------------|---------|-------------|
| ProcessorInstanceType | String | ml.c5.large |
| TrainingInstanceType | String | ml.m5.large |
| TrainingInstancesCount | Integer | 1 |
| DeploymentInstanceType | String | ml.m5.large |
| DeploymentInstancesCount | Integer | 1 |
| ContainerType | String | ml.m5.large |
| SkipModelHealthCheck | Boolean | false |
| RegisterNewModelJobSource | Boolean | false |
| ModelApprovalStatus | String | None |
| ModelApprovalType | Boolean | false |
| ModelApprovalType | String | None |
| ModelApprovalType | String | None |
| ModelApprovalStatus | String | Approved |

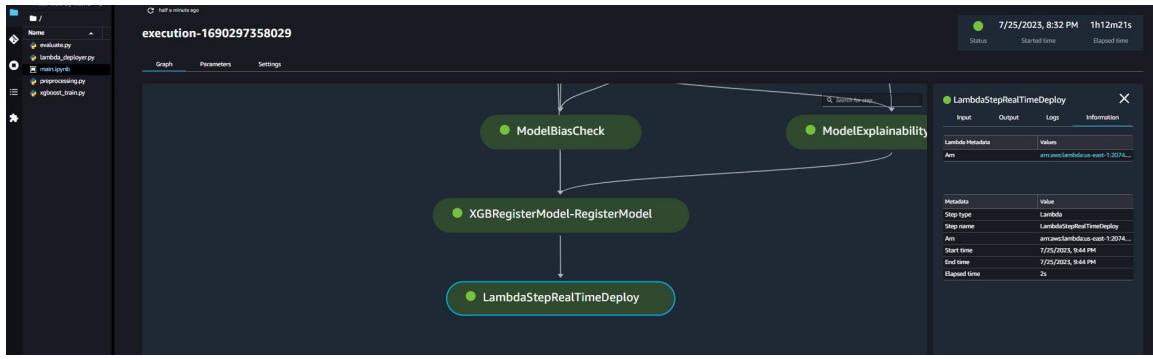
Processing flow:



Details of each steps:







Invoke the endpoint created, the below code return model predictions of first five samples of test set.

The screenshot shows the AWS SageMaker console's "Endpoints" page. The left sidebar includes sections like "JumpStart", "Inference", and "Endpoints". The main area displays a table with one row for the endpoint. The endpoint name is "fraud-detect-xgb-pipeline-endpoint" and its ARN is "arn:aws:sagemaker:us-east-1:207468113308:endpoint/fraud-detect-xgb-pipeline-endpoint". The "Creation time" is "7/25/2023, 9:44:57 PM", "Status" is "InService", and "Last updated" is "7/25/2023, 9:47:54 PM". There is a "Create endpoint" button at the top right.

The screenshot shows an Amazon SageMaker Studio Jupyter Notebook cell containing Python code. The code imports pandas and sagemaker, creates a predictor from a specific endpoint, and then uses it to make predictions on a test dataset. The output of the code shows the first five rows of the prediction results.

```

# Fetch test data to run predictions with the endpoint
test_df = pd.read_csv('processing_output_uris/test_data/test.csv')

# Create SageMaker Predictor from the deployment
predictor = sagemaker.predictor.Predictor(endpoint_name,
                                           sagemaker_session=sess,
                                           serializer=CSVSerializer(),
                                           deserializer=CSVDeserializer())

# Test endpoint with payload of 5 samples
payload = test_df.drop(['Fraud'], axis=1).iloc[0:5]
result = predictor.predict(payload)
prediction_df = pd.DataFrame(result)

prediction_df['Prediction'] = result
prediction_df['Label'] = test_df['Fraud'].iloc[0:5].values
prediction_df

```

| | Prediction | Label |
|---|-----------------------|-------|
| 0 | [0.919172540140152] | 0 |
| 1 | [0.0828506740140915] | 0 |
| 2 | [0.044510705719223] | 0 |
| 3 | [0.03737560051402085] | 0 |
| 4 | [0.0628506740140915] | 0 |

Endpoint monitoring

SageMaker dashboard
Images
Lifecycle configurations
Search
JumpStart
Foundation models (1)
Computer vision models
Natural language processing models
Governance
Ground Truth
Notebook
Processing
Training
Inference
Compilation jobs
Marketplace model packages
Models

Name: fraud-detect-xgb-pipeline-endpoint Status: InService Type: Real-time
ARN: arn:aws:sagemaker:us-east-1:207468113308:endpoint/fraud-detect-xgb-pipeline-endpoint Creation time: Tue Jul 25 2023 21:44:57 GMT+0530 (India Standard Time) Last updated: Tue Jul 25 2023 21:47:54 GMT+0530 (India Standard Time)
URL: http://runtime.sagemaker.us-east-1.amazonaws.com/endpoints/fraud-detect-xgb-pipeline-endpoint/invoke
Model container logs: /www/sagemaker/endpoints/fraud-detect-xgb-pipeline-endpoint/invocations
Learn more about the API

Monitor Settings Alarms

Operational Metrics

CPU Utilization (info) Percentage: 0.20 (Jul 25 21:50), 0.10 (Jul 25 21:55), 0.10 (Jul 25 22:00), 0.10 (Jul 25 22:05), 0.10 (Jul 25 22:10), 0.10 (Jul 25 22:15), 0.10 (Jul 25 22:20)
Memory Utilization (info) Percentage: 4.00 (Jul 25 21:50), 3.50 (Jul 25 21:55), 3.50 (Jul 25 22:00), 3.50 (Jul 25 22:05), 3.50 (Jul 25 22:10), 3.50 (Jul 25 22:15), 3.50 (Jul 25 22:20)
Disk Utilization (info) Percentage: 0.00 (Jul 25 21:50), 0.00 (Jul 25 21:55), 0.00 (Jul 25 22:00), 0.00 (Jul 25 22:05), 0.00 (Jul 25 22:10), 0.00 (Jul 25 22:15), 0.00 (Jul 25 22:20)
GPU Utilization (info) No data available
There is no data available

1h 3h 12h 1d 3d 1w 1 Minute Average + Add widget

Domains SageMaker dashboard Images Lifecycle configurations Search JumpStart Foundation models (1) Computer vision models Natural language processing models Governance Ground Truth Notebook Processing Training Inference Compilation jobs Marketplace model packages Models Endpoint configurations Endpoints Batch transform jobs Shadow tests Inference Recommender Edge Manager Augmented AI AWS Marketplace Tutorials Documentation

Dashboard Feedback Language 21°C Mostly cloudy © 2023, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookies preferences ENG US 10:28 PM 7/25/2023

Getting started Studio Studio List (2) Create Inference Dashboard

JumpStart Foundation models (1) Computer vision models Natural language processing models Governance Ground Truth Notebook Processing Inference Compilation jobs Marketplace model packages Models Endpoint configurations Endpoints Batch transform jobs Shadow tests Inference Recommender Edge Manager Augmented AI AWS Marketplace Tutorials Documentation

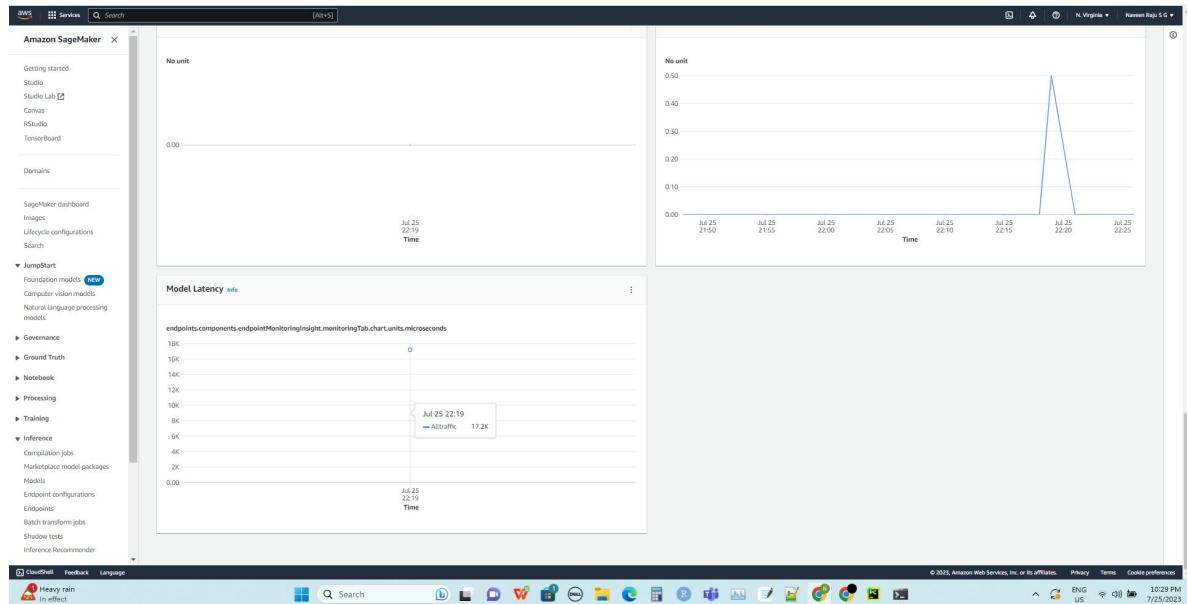
Invocation Metrics

Invocation 4XX Errors (info) No unit
Invocation 5XX Errors (info) No unit

Invocation Model Errors (info) No unit
Invocation Per Instance (info) No unit
Jul 25 22:19 (Alerts: 0.00)

1h 3h 12h 1d 3d 1w 1 Minute Average + Add widget

© 2023, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookies preferences ENG US 10:28 PM 7/25/2023



The screenshot shows the Amazon SageMaker Settings Tab. It includes sections for Data capture settings, Endpoint runtime settings, and Endpoint configuration settings. In the Endpoint configuration settings section, there's a table for 'Endpoint configuration'. One row is visible for 'fraud-detect-xgb-pipeline-endpoint-config' with columns: Name, ARN, Encryption key, and Creation time (7/25/2023, 9:44:57 PM). Below this is a 'Data capture' section with fields for enable data capture, current sampling percentage (%), S3 location to store data collected, and capture content type. At the bottom, there are sections for 'Variants' (Production and Shadow) and 'Shadow' invocation configuration.

This screenshot shows the 'Shadow' section of the Settings Tab. It has a table for 'Model name' with columns: Training job, Variant name, Instance type, Elastic Inference, Initial instance count, and Initial weight. All values are currently set to '-' or 'No'. Below this is an 'Async invocation configuration' section with fields for Max concurrent invocations per instance, S3 output path, Success notification location (which is required), and Error notification location (which is required).

S3 buckets structure

The screenshot shows the AWS S3 console interface. On the left, the navigation pane includes 'Amazon S3', 'Buckets', 'Access Points', 'Object Lambda Access Points', 'Multi-Region Access Points', 'Batch Operations', 'IAM Access Analyzer for S3', 'Block Public Access settings for this account', 'Storage Lens', 'Dashboards', 'AWS Organizations settings', 'Feature spotlight', and 'AWS Marketplace for S3'. The main content area displays the 'fraud-detect-demo/' bucket. It shows a table titled 'Objects (6)' with columns: Name, Type, Last modified, Size, and Storage class. The objects listed are: clarify_output/ (Folder), model_eval/ (Folder), model_monitor/ (Folder), processing_jobs/ (Folder), scripts/ (Folder), and training_jobs/ (Folder). At the top right of the main area, there is a 'Copy S3 URI' button.

Step 6: Clean up the resources

This below code deletes the following lambda function, endpoint, endpoint config, model and the pipeline.

```
# Delete the lambda function
func.delete()

# Delete the endpoint
sm_client.delete_endpoint(EndpointName=endpoint_name)

# Delete the EndpointConfig
sm_client.delete_endpoint_config(EndpointConfigName=endpoint_config_name)

# Delete the model
sm_client.delete_model(ModelName=pipeline_model_name)

# Delete the pipeline
sm_client.delete_pipeline(PipelineName=pipeline_name)

PipelineArn: 'arn:aws:sagemaker:us-east-1:207468113308:pipeline/fraudDetectXGBoostPipeline',
'ResponseMetadata': {'RequestId': 'ac197900-b5b5-4f72-9f32-6888fb78ecab',
'HTTPHeaders': {'x-amzn-requestid': 'ac197900-b5b5-4f72-9f32-6888fb78ecab',
'content-type': 'application/x-amz-json-1.1',
'content-length': '80',
'date': 'Tue, 25 Jul 2023 17:53:06 GMT'},
'	date': 'Tue, 25 Jul 2023 17:53:06 GMT'},
'		Attempts': 0}}
```

After which we can delete all s3 buckets created

The screenshot shows the AWS S3 console interface, identical to the one above but with a different scroll position. The main content area displays the 'fraud-detect-demo/' bucket. It shows a table titled 'Objects (6)' with columns: Name, Type, Last modified, Size, and Storage class. The objects listed are: clarify_output/ (Folder), model_eval/ (Folder), model_monitor/ (Folder), processing_jobs/ (Folder), scripts/ (Folder), and training_jobs/ (Folder). At the top right of the main area, there is a 'Copy S3 URI' button.

Following which we can terminate all apps running in SageMaker Studio under created domain after which we can delete the stack created through CloudFormation.