Assignment 5.1: ML System Observability

Geoffrey Fadera

AAI-540 Summer 2025

June 10th 2025

Imports

In [1]:
```
import copy
import json
import random
import time
import pandas as pd

from datetime import datetime, timedelta

from sagemaker import get_execution_role, image_uris, Session
from sagemaker.clarify import (
    BiasConfig,
    DataConfig,
    ModelConfig,
    ModelPredictedLabelConfig,
    SHAPConfig,
)
from sagemaker.model import Model
from sagemaker.model_monitor import (
    BiasAnalysisConfig,
    CronExpressionGenerator,
    DataCaptureConfig,
    EndpointInput,
    ExplainabilityAnalysisConfig,
    ModelBiasMonitor,
    ModelExplainabilityMonitor,
)
from sagemaker.s3 import S3Downloader, S3Uploader
```

Access Configurations for AWS Services

In [2]:
```
role = get_execution_role()
print(f"RoleArn: {role}")

sagemaker_session = Session()
sagemaker_client = sagemaker_session.sagemaker_client
sagemaker_runtime_client = sagemaker_session.sagemaker_runtime_client

region = sagemaker_session.boto_region_name
print(f"AWS region: {region}")

# A different bucket can be used, but make sure the role for this notebook has
# the s3:PutObject permissions. This is the bucket into which the data is captured
bucket = Session().default_bucket()
print(f"Demo Bucket: {bucket}")
prefix = "sagemaker/DEMO-ClarifyModelMonitor-20200901"
s3_key = f"s3://{bucket}/{prefix}"
print(f"S3 key: {s3_key}")

s3_capture_upload_path = f"{s3_key}/datacapture"
ground_truth_upload_path = f"{s3_key}/ground_truth_data/{datetime.now():%Y-%m-%d-%H-%M-%S}"
s3_report_path = f"{s3_key}/reports"

print(f"Capture path: {s3_capture_upload_path}")
print(f"Ground truth path: {ground_truth_upload_path}")
print(f"Report path: {s3_report_path}")

baseline_results_uri = f"{s3_key}/baselining"
print(f"Baseline results uri: {baseline_results_uri}")

endpoint_instance_count = 1
endpoint_instance_type = "ml.m5.xlarge"
schedule_expression = CronExpressionGenerator.hourly()
```

```
RoleArn: arn:aws:iam::324183265896:role/service-role/AmazonSageMaker-ExecutionRole-20250604T045982
AWS region: us-east-1
Demo Bucket: sagemaker-us-east-1-324183265896
S3 key: s3://sagemaker-us-east-1-324183265896/sagemaker/DEMO-ClarifyModelMonitor-20200901
Capture path: s3://sagemaker-us-east-1-324183265896/sagemaker/DEMO-ClarifyModelMonitor-20200901/datacapture
Ground truth path: s3://sagemaker-us-east-1-324183265896/sagemaker/DEMO-ClarifyModelMonitor-20200901/ground_truth_data/2025-06-10-07-11-55
Report path: s3://sagemaker-us-east-1-324183265896/sagemaker/DEMO-ClarifyModelMonitor-20200901/reports
Baseline results uri: s3://sagemaker-us-east-1-324183265896/sagemaker/DEMO-ClarifyModelMonitor-20200901/baselining
```

Models and Datasets

In [3]:
```
#model_file = "model/xgb-churn-prediction-model.tar.gz"
test_file = "test_data/upload-test-file.txt"
test_dataset = "test_data/test-dataset-input-cols.csv"
validation_dataset = "test_data/validation-dataset-with-header.csv"
dataset_type = "text/csv"

with open(validation_dataset) as f:
    headers_line = f.readline().rstrip()
all_headers = headers_line.split(",")
label_header = all_headers[0]
```

In [4]:
```
# Upload a test file
S3Uploader.upload(test_file, f"s3://{bucket}/test_upload")
print("Success! We are all set to proceed.")
```

```
Success! We are all set to proceed.
```

PART A: Capturing real-time inference data from Amazon SageMaker endpoints

Retrieve the url of the pre-trained model uploaded to Amazon S3 in Lab 5.1

In [6]:
```
%store -r lab5_model_url
model_url = lab5_model_url
print(model_url)
```

```
s3://sagemaker-us-east-1-324183265896/sagemaker/Churn-ModelQualityMonitor-20201201/xgb-churn-prediction-model.tar.gz
```

Deploy the model to Amazon SageMaker

In [7]:
```
model_name = f"DEMO-xgb-churn-pred-model-monitor-{datetime.utcnow():%Y-%m-%d-%H%M}"
print("Model name: ", model_name)
endpoint_name = f"DEMO-xgb-churn-model-monitor-{datetime.utcnow():%Y-%m-%d-%H%M}"
print("Endpoint name: ", endpoint_name)
```

```
Model name:  DEMO-xgb-churn-pred-model-monitor-2025-06-10-0734
Endpoint name:  DEMO-xgb-churn-model-monitor-2025-06-10-0734
/tmp/ipykernel_16553/3119141737.py:1: DeprecationWarning: datetime.datetime.utcnow() is deprecated and scheduled for removal in a future version. Use timezone-aware objects to represent datetimes in UTC: datetime.datetime.now(datetime.UTC).
  model_name = f"DEMO-xgb-churn-pred-model-monitor-{datetime.utcnow():%Y-%m-%d-%H%M}"
/tmp/ipykernel_16553/3119141737.py:3: DeprecationWarning: datetime.datetime.utcnow() is deprecated and scheduled for removal in a future version. Use timezone-aware objects to represent datetimes in UTC: datetime.datetime.now(datetime.UTC).
  endpoint_name = f"DEMO-xgb-churn-model-monitor-{datetime.utcnow():%Y-%m-%d-%H%M}"
```

In [8]:
```
image_uri = image_uris.retrieve("xgboost", region, "0.90-1")
print(f"XGBoost image uri: {image_uri}")
model = Model(
    role=role,
    name=model_name,
    image_uri=image_uri,
    model_data=model_url,
    sagemaker_session=sagemaker_session,
)

data_capture_config = DataCaptureConfig(
    enable_capture=True,
    sampling_percentage=100,
    destination_s3_uri=s3_capture_upload_path,
)
print(f"Deploying model {model_name} to endpoint {endpoint_name}")
model.deploy(
    initial_instance_count=endpoint_instance_count,
    instance_type=endpoint_instance_type,
    endpoint_name=endpoint_name,
    data_capture_config=data_capture_config,
)
```

```
XGBoost image uri: 683313688378.dkr.ecr.us-east-1.amazonaws.com/sagemaker-xgboost:0.90-1-cpu-py3
Deploying model DEMO-xgb-churn-pred-model-monitor-2025-06-10-0734 to endpoint DEMO-xgb-churn-model-monitor-2025-06-10-0734
-----!
```

Send Test Data to Deployed Endpoint

In [9]:
```
test_dataset
```

Out[9]: 'test_data/test-dataset-input-cols.csv'

In [10]:
```
# In Lab 5.1, there was a limit in the number of samples used for baselining
print(f"Sending test traffic to the endpoint {endpoint_name}. \nPlease wait", end="")
test_dataset_size = 0  # record the number of rows in data we're sending for inference
with open(test_dataset, "r") as f:
    for row in f:
        if test_dataset_size < 120:
            payload = row.rstrip("\n")
            response = sagemaker_runtime_client.invoke_endpoint(
                EndpointName=endpoint_name,
                Body=payload,
                ContentType=dataset_type,
            )
            prediction = response["Body"].read()
            print(".", end="", flush=True)
            time.sleep(0.5)
        test_dataset_size += 1

print()
print("Done!")
```

```
Sending test traffic to the endpoint DEMO-xgb-churn-model-monitor-2025-06-10-0734.
Please wait........................................................................................................................
Done!
```

In [11]:
```
test_dataset_size
```

Out[11]: 334

View captured data

In [12]:
```
print("Waiting 30 seconds for captures to show up", end="")
for _ in range(30):
    capture_files = sorted(S3Downloader.list(f"{s3_capture_upload_path}/{endpoint_name}"))
    if capture_files:
        break
    print(".", end="", flush=True)
    time.sleep(1)
print()
print("Found Capture Files:")
print("\n ".join(capture_files[-5:]))
```

```
Waiting 30 seconds for captures to show up
Found Capture Files:
s3://sagemaker-us-east-1-324183265896/sagemaker/DEMO-ClarifyModelMonitor-20200901/datacapture/DEMO-xgb-churn-model-monitor-2025-06-10-0734/AllTraffic/2025/06/10/07/54-18-353-f300e498-c6be-4fcf-86c8-d19a83241e73.jsonl
 s3://sagemaker-us-east-1-324183265896/sagemaker/DEMO-ClarifyModelMonitor-20200901/datacapture/DEMO-xgb-churn-model-monitor-2025-06-10-0734/AllTraffic/2025/06/10/07/55-18-587-04520543-3c6c-40ca-a752-24a385b5a7b7.jsonl
```

In [13]:
```
# view the content of a single capture file. Take a quick peek at the first few lines in the captured file.
capture_file = S3Downloader.read_file(capture_files[-1]).split("\n")[-10:-1]
print(capture_file[-1])
```

{"captureData":{"endpointInput":{"observedContentType":"text/csv","mode":"INPUT","data":"93,0,176.1,103,199.7,130,263.9,96,8.5,6,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,1,0","encoding":"CSV"},"endpointOutput":{"observedContentType":"text/csv; charset=utf-8","mode":"OUTPUT","data":"0.007474285550415516","encoding":"CSV"}},"eventMetadata":{"eventId":"315ebf5f-1f54-4d29-804e-4145e9974217","inferenceTime":"2025-06-10T07:55:19Z"},"eventVersion":"0"}

In [14]:
```
# view in json format
print(json.dumps(json.loads(capture_file[-1]), indent=2))
```

```
{
  "captureData": {
    "endpointInput": {
      "observedContentType": "text/csv",
      "mode": "INPUT",
      "data": "93,0,176.1,103,199.7,130,263.9,96,8.5,6,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,1,0",
      "encoding": "CSV"
    },
    "endpointOutput": {
      "observedContentType": "text/csv; charset=utf-8",
      "mode": "OUTPUT",
      "data": "0.007474285550415516",
      "encoding": "CSV"
    }
  },
  "eventMetadata": {
    "eventId": "315ebf5f-1f54-4d29-804e-4145e9974217",
    "inferenceTime": "2025-06-10T07:55:19Z"
  },
  "eventVersion": "0"
}
```


PART B: Model Bias Monitor (this could have been added to the end of Lab 5.1)

Configure Model Bias Monitor using Sagemaker Clarify Configuration Modules for Explainability

In [19]:
```python
# initialize a Sagemaker's Model Bias Monitor object
model_bias_monitor = ModelBiasMonitor(
    role=role,
    sagemaker_session=sagemaker_session,
    max_runtime_in_seconds=1800,
)
```

In [20]:
```python
# Configure the bias monitor using Sagemaker's Clarify Class Objects

# upload path for baselining results
model_bias_baselining_job_result_uri = f"{baseline_results_uri}/model_bias"
# this could have been added to the Lab5.1 bucket under the same predictor but different monitor folder

# Include column header labels for explainability
model_bias_data_config = DataConfig(
    s3_data_input_path=validation_dataset,
    s3_output_path=model_bias_baselining_job_result_uri,
    label=label_header,
    headers=all_headers,
    dataset_type=dataset_type,
)

# Configure Bias # what feature to monitor bias from
model_bias_config = BiasConfig(
    label_values_or_threshold=[1],
    facet_name="Account Length",
    facet_values_or_threshold=[100],
)

# prediction probability threshold: same churn_cutoff rate as in Lab 5.1
model_predicted_label_config = ModelPredictedLabelConfig(
    probability_threshold=0.8,
)

# inference model configurations encapsulated in Clarify class container
model_config = ModelConfig(
    model_name=model_name,
    instance_count=endpoint_instance_count,
    instance_type=endpoint_instance_type,
    content_type=dataset_type,
    accept_type=dataset_type,
)
```

Execute the Baselining Job

In [21]:
```python
model_bias_monitor.suggest_baseline(
    model_config=model_config,
    data_config=model_bias_data_config,
    bias_config=model_bias_config,
    model_predicted_label_config=model_predicted_label_config,
)
print(f"ModelBiasMonitor baselining job: {model_bias_monitor.latest_baselining_job_name}")
```

```
INFO:sagemaker:Creating processing-job with name baseline-suggestion-job-2025-06-10-08-58-00-618
```

```
ModelBiasMonitor baselining job: baseline-suggestion-job-2025-06-10-08-58-00-618
```

In [22]:

```
model_bias_monitor.latest_baselining_job.wait(logs=False)
model_bias_constraints = model_bias_monitor.suggested_constraints()
print()
print(f"ModelBiasMonitor suggested constraints: {model_bias_constraints.file_s3_uri}")
print(S3Downloader.read_file(model_bias_constraints.file_s3_uri))
```

```
.....................................................................................!
ModelBiasMonitor suggested constraints: s3://sagemaker-us-east-1-324183265896/sagemaker/DEMO-ClarifyModelMonitor-20200901/baselining/
model_bias/analysis.json
{
    "version": "1.0",
    "post_training_bias_metrics": {
        "label": "Churn",
        "facets": {
            "Account Length": [
                {
                    "value_or_threshold": "(100, 225]",
                    "metrics": [
                        {
                            "name": "AD",
                            "description": "Accuracy Difference (AD)",
                            "value": 0.03416521605801226
                        },
                        {
                            "name": "CDDPL",
                            "description": "Conditional Demographic Disparity in Predicted Labels (CDDPL)",
                            "value": null,
                            "error": "Group variable is empty or not provided"
                        },
                        {
                            "name": "DAR",
                            "description": "Difference in Acceptance Rates (DAR)",
                            "value": 0.0
                        },
                        {
                            "name": "DCA",
                            "description": "Difference in Conditional Acceptance (DCA)",
                            "value": -0.4137931034482758
                        },
                        {
                            "name": "DCR",
                            "description": "Difference in Conditional Rejection (DCR)",
                            "value": -0.03722943722943728
                        },
                        {
                            "name": "DI",
                            "description": "Disparate Impact (DI)",
                            "value": 0.9762611275964392
                        },
                        {
                            "name": "DPPL",
                            "description": "Difference in Positive Proportions in Predicted Labels (DPPL)",
                            "value": 0.0020924841936269395
                        },
                        {
                            "name": "DRR",
                            "description": "Difference in Rejection Rates (DRR)",
                            "value": -0.03722943722943728
                        },
                        {
                            "name": "FT",
                            "description": "Flip Test (FT)",
                            "value": 0.04154302670623145
                        },
                        {
                            "name": "GE",
                            "description": "Generalized Entropy (GE)",
                            "value": 0.04234527687296418
                        },
                        {
                            "name": "RD",
                            "description": "Recall Difference (RD)",
                            "value": 0.1164268986283038
                        },
                        {
                            "name": "SD",
                            "description": "Specificity Difference (SD)",
                            "value": 0.0
                        },
                        {
                            "name": "TE",
                            "description": "Treatment Equality (TE)",
                            "value": 0.0
                        }
                    ]
                }
            ]
        },
        "label_value_or_threshold": "1"
    },
    "pre_training_bias_metrics": {
```

```
        "label": "Churn",
        "facets": {
            "Account Length": [
                {
                    "value_or_threshold": "(100, 225]",
                    "metrics": [
                        {
                            "name": "CDDL",
                            "description": "Conditional Demographic Disparity in Labels (CDDL)",
                            "value": null,
                            "error": "Group variable is empty or not provided"
                        },
                        {
                            "name": "CI",
                            "description": "Class Imbalance (CI)",
                            "value": -0.012012012012012012
                        },
                        {
                            "name": "DPL",
                            "description": "Difference in Positive Proportions in Labels (DPL)",
                            "value": -0.03207273186438539
                        },
                        {
                            "name": "JS",
                            "description": "Jensen-Shannon Divergence (JS)",
                            "value": 0.0009346452720882569
                        },
                        {
                            "name": "KL",
                            "description": "Kullback-Liebler Divergence (KL)",
                            "value": 0.003645914387951369
                        },
                        {
                            "name": "KS",
                            "description": "Kolmogorov-Smirnov Distance (KS)",
                            "value": 0.03207273186438542
                        },
                        {
                            "name": "LP",
                            "description": "L-p Norm (LP)",
                            "value": 0.04535769238496956
                        },
                        {
                            "name": "TVD",
                            "description": "Total Variation Distance (TVD)",
                            "value": 0.032072731864385404
                        }
                    ]
                }
            ]
        },
        "label_value_or_threshold": "1"
    }
}
```


Add Monitoring Schedule to the baseline bias monitor

In [23]:
```
model_bias_analysis_config = None
if not model_bias_monitor.latest_baselining_job:
    model_bias_analysis_config = BiasAnalysisConfig(
        model_bias_config,
        headers=all_headers,
        label=label_header,
    )
model_bias_monitor.create_monitoring_schedule(
    analysis_config=model_bias_analysis_config,
    output_s3_uri=s3_report_path,
    endpoint_input=EndpointInput(
        endpoint_name=endpoint_name,
        destination="/opt/ml/processing/input/endpoint",
        start_time_offset="-PT1H",
        end_time_offset="-PT0H",
        probability_threshold_attribute=0.8,
    ),
    ground_truth_input=ground_truth_upload_path,
    schedule_cron_expression=schedule_expression,
)
print(f"Model bias monitoring schedule: {model_bias_monitor.monitoring_schedule_name}")
```

```
INFO:sagemaker.model_monitor.clarify_model_monitoring:Uploading analysis config to {s3_uri}.
INFO:sagemaker.model_monitor.model_monitoring:Creating Monitoring Schedule with name: monitoring-schedule-2025-06-10-09-24-36-905
Model bias monitoring schedule: monitoring-schedule-2025-06-10-09-24-36-905
```

Make sure there's constant "ground truth data"

In [27]:
```
import threading


class WorkerThread(threading.Thread):
    def __init__(self, do_run, *args, **kwargs):
        super(WorkerThread, self).__init__(*args, **kwargs)
        self.__do_run = do_run
        self.__terminate_event = threading.Event()

    def terminate(self):
        self.__terminate_event.set()

    def run(self):
        while not self.__terminate_event.is_set():
            self.__do_run(self.__terminate_event)
```

In [28]:
```
def invoke_endpoint(terminate_event):
    with open(test_dataset, "r") as f:
        i = 0
        for row in f:
            payload = row.rstrip("\n")
            response = sagemaker_runtime_client.invoke_endpoint(
                EndpointName=endpoint_name,
                ContentType="text/csv",
                Body=payload,
                InferenceId=str(i),  # unique ID per row
            )
            i += 1
            response["Body"].read()
            time.sleep(1)
            if terminate_event.is_set():
                break


# Keep invoking the endpoint with test data
invoke_endpoint_thread = WorkerThread(do_run=invoke_endpoint)
invoke_endpoint_thread.start()
```

In [29]:
```
# Use Threading to generate artificial feature data and feature labels

def ground_truth_with_id(inference_id):
    random.seed(inference_id)  # to get consistent results
    rand = random.random()
    # format required by the merge container
    return {
        "groundTruthData": {
            "data": "1" if rand < 0.7 else "0",  # randomly generate positive labels 70% of the time
            "encoding": "CSV",
        },
        "eventMetadata": {
            "eventId": str(inference_id),
        },
        "eventVersion": "0",
    }


def upload_ground_truth(upload_time):
    records = [ground_truth_with_id(i) for i in range(test_dataset_size)]
    fake_records = [json.dumps(r) for r in records]
    data_to_upload = "\n".join(fake_records)
    target_s3_uri = f"{ground_truth_upload_path}/{upload_time:%Y/%m/%d/%H/%M%S}.jsonl"
    print(f"Uploading {len(fake_records)} records to", target_s3_uri)
    S3Uploader.upload_string_as_file_body(data_to_upload, target_s3_uri)
```

In [30]:
```
# Generate data for the last hour
upload_ground_truth(datetime.utcnow() - timedelta(hours=1))
```

```
/tmp/ipykernel_16553/1956821817.py:2: DeprecationWarning: datetime.datetime.utcnow() is deprecated and scheduled for removal in a fut
ure version. Use timezone-aware objects to represent datetimes in UTC: datetime.datetime.now(datetime.UTC).
  upload_ground_truth(datetime.utcnow() - timedelta(hours=1))
Uploading 334 records to s3://sagemaker-us-east-1-324183265896/sagemaker/DEMO-ClarifyModelMonitor-20200901/ground_truth_data/2025-06-
10-07-11-55/2025/06/10/08/3126.jsonl
```

In [31]:
```
# Generate data once a hour
def generate_fake_ground_truth(terminate_event):
    upload_ground_truth(datetime.utcnow())
    for _ in range(0, 60):
        time.sleep(60)
        if terminate_event.is_set():
            break
```

```
ground_truth_thread = WorkerThread(do_run=generate_fake_ground_truth)
ground_truth_thread.start()
```

```
Uploading 334 records to s3://sagemaker-us-east-1-324183265896/sagemaker/DEMO-ClarifyModelMonitor-20200901/ground_truth_data/2025-06-10-07-11-55/2025/06/10/09/3134.jsonl
```

```
/tmp/ipykernel_16553/2589954271.py:3: DeprecationWarning: datetime.datetime.utcnow() is deprecated and scheduled for removal in a future version. Use timezone-aware objects to represent datetimes in UTC: datetime.datetime.now(datetime.UTC).
  upload_ground_truth(datetime.utcnow())
```

Start Scheduled Model Bias Monitoring

In [32]:
```
def wait_for_execution_to_start(model_monitor):
    print(
        "A hourly schedule was created above and it will kick off executions ON the hour (plus 0 - 20 min buffer)."
    )

    print("Waiting for the first execution to happen", end="")
    schedule_desc = model_monitor.describe_schedule()
    while "LastMonitoringExecutionSummary" not in schedule_desc:
        schedule_desc = model_monitor.describe_schedule()
        print(".", end="", flush=True)
        time.sleep(60)
    print()
    print("Done! Execution has been created")

    print("Now waiting for execution to start", end="")
    while schedule_desc["LastMonitoringExecutionSummary"]["MonitoringExecutionStatus"] in "Pending":
        schedule_desc = model_monitor.describe_schedule()
        print(".", end="", flush=True)
        time.sleep(10)

    print()
    print("Done! Execution has started")
```

In [33]:
```
# start scheduled bias monitoring and wait until the 1st one executes
wait_for_execution_to_start(model_bias_monitor)
```

```
A hourly schedule was created above and it will kick off executions ON the hour (plus 0 - 20 min buffer).
Waiting for the first execution to happen.............................
Done! Execution has been created
Now waiting for execution to start.........................
Done! Execution has started
```

In [35]:
```
# function: wait for the current monitoring to finish
def wait_for_execution_to_finish(model_monitor):
    schedule_desc = model_monitor.describe_schedule()
    execution_summary = schedule_desc.get("LastMonitoringExecutionSummary")
    if execution_summary is not None:
        print("Waiting for execution to finish", end="")
        while execution_summary["MonitoringExecutionStatus"] not in [
            "Completed",
            "CompletedWithViolations",
            "Failed",
            "Stopped",
        ]:
            print(".", end="", flush=True)
            time.sleep(60)
            schedule_desc = model_monitor.describe_schedule()
            execution_summary = schedule_desc["LastMonitoringExecutionSummary"]
        print()
        print("Done! Execution has finished")
    else:
        print("Last execution not found")
```

In [36]:
```
# wait for the current monitoring to finish
wait_for_execution_to_finish(model_bias_monitor)
```

```
Waiting for execution to finish...
Done! Execution has finished
```

Inspect latest monitoring execution results

In [37]:
```
schedule_desc = model_bias_monitor.describe_schedule()
execution_summary = schedule_desc.get("LastMonitoringExecutionSummary")
if execution_summary and execution_summary["MonitoringExecutionStatus"] in [
    "Completed",
    "CompletedWithViolations",
]:
    last_model_bias_monitor_execution = model_bias_monitor.list_executions()[-1]
    last_model_bias_monitor_execution_report_uri = (
        last_model_bias_monitor_execution.output.destination
    )
    print(f"Report URI: {last_model_bias_monitor_execution_report_uri}")
    last_model_bias_monitor_execution_report_files = sorted(
```

```
        S3Downloader.list(last_model_bias_monitor_execution_report_uri)
    )
    print("Found Report Files:")
    print("\n ".join(last_model_bias_monitor_execution_report_files))
else:
    last_model_bias_monitor_execution = None
    print(
        "====STOP==== \n No completed executions to inspect further. Please wait till an execution completes or investigate previousl
    )
```

```
Report URI: s3://sagemaker-us-east-1-324183265896/sagemaker/DEMO-ClarifyModelMonitor-20200901/reports/DEMO-xgb-churn-model-monitor-20
25-06-10-0734/monitoring-schedule-2025-06-10-09-24-36-905/2025/06/10/10
Found Report Files:
s3://sagemaker-us-east-1-324183265896/sagemaker/DEMO-ClarifyModelMonitor-20200901/reports/DEMO-xgb-churn-model-monitor-2025-06-10-073
4/monitoring-schedule-2025-06-10-09-24-36-905/2025/06/10/10/analysis.json
 s3://sagemaker-us-east-1-324183265896/sagemaker/DEMO-ClarifyModelMonitor-20200901/reports/DEMO-xgb-churn-model-monitor-2025-06-10-07
34/monitoring-schedule-2025-06-10-09-24-36-905/2025/06/10/10/constraint_violations.json
 s3://sagemaker-us-east-1-324183265896/sagemaker/DEMO-ClarifyModelMonitor-20200901/reports/DEMO-xgb-churn-model-monitor-2025-06-10-07
34/monitoring-schedule-2025-06-10-09-24-36-905/2025/06/10/10/report.html
 s3://sagemaker-us-east-1-324183265896/sagemaker/DEMO-ClarifyModelMonitor-20200901/reports/DEMO-xgb-churn-model-monitor-2025-06-10-07
34/monitoring-schedule-2025-06-10-09-24-36-905/2025/06/10/10/report.ipynb
 s3://sagemaker-us-east-1-324183265896/sagemaker/DEMO-ClarifyModelMonitor-20200901/reports/DEMO-xgb-churn-model-monitor-2025-06-10-07
34/monitoring-schedule-2025-06-10-09-24-36-905/2025/06/10/10/report.pdf
/tmp/ipykernel_16553/2589954271.py:3: DeprecationWarning: datetime.datetime.utcnow() is deprecated and scheduled for removal in a fut
ure version. Use timezone-aware objects to represent datetimes in UTC: datetime.datetime.now(datetime.UTC).
  upload_ground_truth(datetime.utcnow())
Uploading 334 records to s3://sagemaker-us-east-1-324183265896/sagemaker/DEMO-ClarifyModelMonitor-20200901/ground_truth_data/2025-06-
10-07-11-55/2025/06/10/10/3134.jsonl
```

In [38]:
```
if last_model_bias_monitor_execution:
    model_bias_violations = last_model_bias_monitor_execution.constraint_violations()
    if model_bias_violations:
        print(model_bias_violations.body_dict)
```

```
{'version': '1.0', 'violations': [{'facet': 'Account Length', 'facet_value': '(100, 232]', 'metric_name': 'DAR', 'constraint_check_ty
pe': 'bias_drift_check', 'description': "Metric value -0.1693989071038252 doesn't meet the baseline constraint requirement 0.0"}, {'f
acet': 'Account Length', 'facet_value': '(100, 232]', 'metric_name': 'DCA', 'constraint_check_type': 'bias_drift_check', 'descriptio
n': "Metric value 3.448816029143897 doesn't meet the baseline constraint requirement -0.4137931034482758"}, {'facet': 'Account Lengt
h', 'facet_value': '(100, 232]', 'metric_name': 'DI', 'constraint_check_type': 'bias_drift_check', 'description': "Metric value 1.333
5650830436463 doesn't meet the baseline constraint requirement 0.9762611275964392"}, {'facet': 'Account Length', 'facet_value': '(10
0, 232]', 'metric_name': 'DPPL', 'constraint_check_type': 'bias_drift_check', 'description': "Metric value -0.01768012807651835 does
n't meet the baseline constraint requirement 0.00209248419362693 95"}, {'facet': 'Account Length', 'facet_value': '(100, 232]', 'metri
c_name': 'GE', 'constraint_check_type': 'bias_drift_check', 'description': "Metric value 1.1181474480151226 doesn't meet the baseline
constraint requirement 0.04234527687296418"}, {'facet': 'Account Length', 'facet_value': '(100, 232]', 'metric_name': 'SD', 'constrai
nt_check_type': 'bias_drift_check', 'description': "Metric value 0.02246107958678889 doesn't meet the baseline constraint requirement
0.0"}, {'facet': 'Account Length', 'facet_value': '(100, 232]', 'metric_name': 'TE', 'constraint_check_type': 'bias_drift_check', 'de
scription': "Metric value 18.366666666666667 doesn't meet the baseline constraint requirement 0.0"}]}
```

In [ ]:
```
```