Certified AI Practitioner Week 02 Call 02 - Deploy and Test Your First ML Endpoint

Learning Objectives

- Explain the difference between real-time, batch, and async inference in SageMaker
- Locate and load a trained model artifact from S3
- Deploy a scikit-learn model to a real-time SageMaker endpoint
- Send a test payload using `boto3` and interpret the prediction result
- Access and review endpoint logs using CloudWatch
- Identify common errors and performance metrics in deployed inference
- Delete SageMaker endpoints to avoid unnecessary cost

Infrastructure Setup with CloudFormation

To train models in SageMaker Studio, we first need to provision the necessary infrastructure.

In this step, we'll use an automated **CloudFormation template** to create:

- A **SageMaker Studio domain** for running cloud-based notebooks
- An **IAM execution role** with S3 and SageMaker permissions
- A dedicated **S3 bucket** to store training data and model artifacts

Instead of clicking through the AWS Console, we'll deploy this setup programmatically using `boto3`. The stack will output everything you need - including the bucket name and role ARN - to use in the next steps of this notebook.

In [3]:
```
import boto3
import json

def stack_exists(name):
    try:
```

```
        cf.describe_stacks(StackName=name)
        return True
    except cf.exceptions.ClientError as e:
        if "does not exist" in str(e):
            return False
        raise  # re-raise any unexpected error

def deploy_stack(stack_name, template_body, parameters):
    if stack_exists(stack_name):
        print(f"🔄 Updating stack: {stack_name}")
        try:
            response = cf.update_stack(
                StackName=stack_name,
                TemplateBody=template_body,
                Parameters=parameters,
                Capabilities=["CAPABILITY_NAMED_IAM"]
            )
            waiter = cf.get_waiter("stack_update_complete")
        except cf.exceptions.ClientError as e:
            if "No updates are to be performed" in str(e):
                print("✅ No changes detected.")
                # Print outputs
                outputs = cf.describe_stacks(StackName=stack_name)["Stacks"][0]["Outputs"]
                print("🔗 Stack Outputs:")
                print(json.dumps({o["OutputKey"]: o["OutputValue"] for o in outputs}, indent=2))
                return outputs
            else:
                raise
    else:
        print(f"🚀 Creating stack: {stack_name}")
        response = cf.create_stack(
            StackName=stack_name,
            TemplateBody=template_body,
            Parameters=parameters,
            Capabilities=["CAPABILITY_NAMED_IAM"]
        )
        waiter = cf.get_waiter("stack_create_complete")

    print(f"⏳ Waiting for {stack_name} to complete...")
    waiter.wait(StackName=stack_name)
    print("✅ Stack operation completed.")
```

```
    # Print outputs
    outputs = cf.describe_stacks(StackName=stack_name)["Stacks"][0]["Outputs"]
    print("🔧 Stack Outputs:")
    print(json.dumps({o["OutputKey"]: o["OutputValue"] for o in outputs}, indent=2))
    return outputs

ec2 = boto3.client("ec2")
cf = boto3.client("cloudformation")

# Get the default VPC ID
vpc_id = ec2.describe_vpcs(Filters=[{"Name": "isDefault", "Values": ["true"]}])["Vpcs"][0]["VpcId"]

# Get a public subnet ID in that VPC
subnets = ec2.describe_subnets(Filters=[{"Name": "vpc-id", "Values": [vpc_id]}])
subnet_id = subnets["Subnets"][0]["SubnetId"]

# Load your template
with open("cf_templates/sagemaker_infra.yaml") as f:
    template_body = f.read()

bucketNameSuffix = "zali"
stack_name = "caip02-cloud-ml-stack"

parameters = [
    {"ParameterKey": "BucketNameSuffix", "ParameterValue": bucketNameSuffix},
    {"ParameterKey": "VpcId", "ParameterValue": vpc_id},
    {"ParameterKey": "SubnetId", "ParameterValue": subnet_id}
]

outputs = deploy_stack(stack_name, template_body, parameters)
```

```
🔄 Updating stack: caip02-cloud-ml-stack
✅ No changes detected.
🔧 Stack Outputs:
{
  "StudioUserName": "caip02-user",
  "BucketName": "caip02-ml-bucket-zali",
  "DomainId": "d-u5qzlht4ntsi",
  "RoleArn": "arn:aws:iam::458806987020:role/caip02-execution-role-zali"
}
🔧 Stack Outputs:
{
  "StudioUserName": "caip02-user",
  "BucketName": "caip02-ml-bucket-zali",
  "DomainId": "d-u5qzlht4ntsi",
  "RoleArn": "arn:aws:iam::458806987020:role/caip02-execution-role-zali"
}
```

Upload Data to S3

We'll upload the previously prepared Titanic dataset to our dedicated S3 bucket so it can be used by SageMaker for training.

Make sure you're using the bucket created by your CloudFormation stack. You can retrieve it from the stack outputs.

In [7]:
```python
import boto3

bucket_name = {o["OutputKey"]: o["OutputValue"] for o in outputs if o["OutputKey"] == "BucketName"}["BucketName"]

key = "inputs/cleaned_titanic.csv"

s3 = boto3.client("s3")
s3.upload_file("cleaned_titanic.csv", bucket_name, key)

print(f"Uploaded cleaned_titanic.csv to s3://{bucket_name}/{key}")
```

```
Uploaded cleaned_titanic.csv to s3://caip02-ml-bucket-zali/inputs/cleaned_titanic.csv
```

Write the Training Script (train_model.py)

SageMaker training jobs run inside isolated containers. Instead of writing code cell by cell, we'll create a standalone script that SageMaker will execute in the cloud.

This script will:

- Load the Titanic dataset from the input channel
- Train a Decision Tree model
- Evaluate performance using a confusion matrix and classification report
- Print feature importances
- Save the trained model to the `/opt/ml/model/` directory for SageMaker to capture

We'll save this script locally so we can pass it into a SageMaker training job.

In [29]:
```python
from IPython.display import Markdown, display

def show_code(filepath):
    with open(filepath, "r") as f:
        code = f.read()
    display(Markdown(f"```python\n{code}\n```"))

# Call this after writing the file
show_code("train_model.py")
```

```python
import pandas as pd
import joblib
import os

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix

def main():
    # SageMaker passes input as /opt/ml/input/data/train/
    input_path = "/opt/ml/input/data/train/cleaned_titanic.csv"
    df = pd.read_csv(input_path)

    # Split features and target
    X = df.drop("Survived", axis=1)
    y = df["Survived"]

    # Train/test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_state=42)

    # Scale features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Train model
    clf = DecisionTreeClassifier(max_depth=3, random_state=42)
    clf.fit(X_train_scaled, y_train)

    # Evaluate
    y_pred = clf.predict(X_test_scaled)
```

```
    print("\n=== Classification Report ===\n")
    print(classification_report(y_test, y_pred))

    print("\n=== Confusion Matrix ===\n")
    print(confusion_matrix(y_test, y_pred))

    print("\n=== Feature Importances ===\n")
    importances = pd.Series(clf.feature_importances_, index=X.columns)
    print(importances.sort_values(ascending=False))

    # Save model
    os.makedirs("/opt/ml/model", exist_ok=True)
    joblib.dump(clf, "/opt/ml/model/model.joblib")
    joblib.dump(scaler, "/opt/ml/model/scaler.joblib")

if __name__ == "__main__":
    main()
```

Launch a SageMaker Training Job

We'll use the **SKLearn Estimator**, which runs our `train_model.py` script that we previously developed.

In [30]:
```
from sagemaker.sklearn.estimator import SKLearn
import sagemaker
import boto3

# Values from your CloudFormation stack
bucket = {o["OutputKey"]: o["OutputValue"] for o in outputs if o["OutputKey"] == "BucketName"}["BucketName"]
role = {o["OutputKey"]: o["OutputValue"] for o in outputs if o["OutputKey"] == "RoleArn"}["RoleArn"]
region = boto3.Session().region_name

# Input and output locations in S3
input_path = f"s3://{bucket}/inputs/"
```

```
output_path = f"s3://{bucket}/models/"

# Set up the SageMaker session
session = sagemaker.Session()

# Create SKLearn estimator
estimator = SKLearn(
    entry_point="train_model.py",
    role=role,
    instance_type="ml.m5.large",
    framework_version="1.0-1",
    sagemaker_session=session,
    output_path=output_path,
    base_job_name="titanic-decision-tree"
)

# Launch the training job
estimator.fit({"train": input_path})
```

```
INFO:sagemaker.telemetry.telemetry_logging:SageMaker Python SDK will collect telemetry to help us better understand o
ur user's needs, diagnose issues, and deliver additional features.
To opt out of telemetry, please disable via TelemetryOptOut parameter in SDK defaults config. For more information, r
efer to https://sagemaker.readthedocs.io/en/stable/overview.html#configuring-and-using-defaults-with-the-sagemaker-py
thon-sdk.
INFO:sagemaker:Creating training-job with name: titanic-decision-tree-2025-06-06-19-47-07-355
```

```
2025-06-06 19:47:08 Starting - Starting the training job...
2025-06-06 19:47:32 Starting - Preparing the instances for training...
2025-06-06 19:47:59 Downloading - Downloading input data...
2025-06-06 19:48:24 Downloading - Downloading the training image.....2025-06-06 19:49:24,564 sagemaker-containers INF
O     Imported framework sagemaker_sklearn_container.training
2025-06-06 19:49:24,568 sagemaker-training-toolkit INFO     No GPUs detected (normal if no gpus installed)
2025-06-06 19:49:24,571 sagemaker-training-toolkit INFO     No Neurons detected (normal if no neurons installed)
2025-06-06 19:49:24,589 sagemaker_sklearn_container.training INFO     Invoking user training script.
2025-06-06 19:49:24,812 sagemaker-training-toolkit INFO     No GPUs detected (normal if no gpus installed)
2025-06-06 19:49:24,815 sagemaker-training-toolkit INFO     No Neurons detected (normal if no neurons installed)
2025-06-06 19:49:24,833 sagemaker-training-toolkit INFO     No GPUs detected (normal if no gpus installed)
2025-06-06 19:49:24,836 sagemaker-training-toolkit INFO     No Neurons detected (normal if no neurons installed)
2025-06-06 19:49:24,857 sagemaker-training-toolkit INFO     No GPUs detected (normal if no gpus installed)
2025-06-06 19:49:24,860 sagemaker-training-toolkit INFO     No Neurons detected (normal if no neurons installed)
2025-06-06 19:49:24,876 sagemaker-training-toolkit INFO     Invoking user script
Training Env:
{
    "additional_framework_parameters": {},
    "channel_input_dirs": {
        "train": "/opt/ml/input/data/train"
    },
    "current_host": "algo-1",
    "current_instance_group": "homogeneousCluster",
    "current_instance_group_hosts": [
        "algo-1"
    ],
    "current_instance_type": "ml.m5.large",
    "distribution_hosts": [],
    "distribution_instance_groups": [],
    "framework_module": "sagemaker_sklearn_container.training:main",
    "hosts": [
        "algo-1"
    ],
    "hyperparameters": {},
    "input_config_dir": "/opt/ml/input/config",
    "input_data_config": {
        "train": {
            "TrainingInputMode": "File",
            "S3DistributionType": "FullyReplicated",
            "RecordWrapperType": "None"
        }
    },
```

```
"input_dir": "/opt/ml/input",
"instance_groups": [
    "homogeneousCluster"
],
"instance_groups_dict": {
    "homogeneousCluster": {
        "instance_group_name": "homogeneousCluster",
        "instance_type": "ml.m5.large",
        "hosts": [
            "algo-1"
        ]
    }
},
"is_hetero": false,
"is_master": true,
"is_modelparallel_enabled": null,
"is_smddpmprun_installed": false,
"is_smddprun_installed": false,
"job_name": "titanic-decision-tree-2025-06-06-19-47-07-355",
"log_level": 20,
"master_hostname": "algo-1",
"model_dir": "/opt/ml/model",
"module_dir": "s3://caip02-ml-bucket-zali/titanic-decision-tree-2025-06-06-19-47-07-355/source/sourcedir.tar.gz",
"module_name": "train_model",
"network_interface_name": "eth0",
"num_cpus": 2,
"num_gpus": 0,
"num_neurons": 0,
"output_data_dir": "/opt/ml/output/data",
"output_dir": "/opt/ml/output",
"output_intermediate_dir": "/opt/ml/output/intermediate",
"resource_config": {
    "current_host": "algo-1",
    "current_instance_type": "ml.m5.large",
    "current_group_name": "homogeneousCluster",
    "hosts": [
        "algo-1"
    ],
    "instance_groups": [
        {
            "instance_group_name": "homogeneousCluster",
            "instance_type": "ml.m5.large",
```

```
                "hosts": [
                    "algo-1"
                ]
            }
        ],
        "network_interface_name": "eth0",
        "topology": null
    },
    "user_entry_point": "train_model.py"
}
Environment variables:
SM_HOSTS=["algo-1"]
SM_NETWORK_INTERFACE_NAME=eth0
SM_HPS={}
SM_USER_ENTRY_POINT=train_model.py
SM_FRAMEWORK_PARAMS={}
SM_RESOURCE_CONFIG={"current_group_name":"homogeneousCluster","current_host":"algo-1","current_instance_type":"ml.m5.
large","hosts":["algo-1"],"instance_groups":[{"hosts":["algo-1"],"instance_group_name":"homogeneousCluster","instance
_type":"ml.m5.large"}],"network_interface_name":"eth0","topology":null}
SM_INPUT_DATA_CONFIG={"train":{"RecordWrapperType":"None","S3DistributionType":"FullyReplicated","TrainingInputMod
e":"File"}}
SM_OUTPUT_DATA_DIR=/opt/ml/output/data
SM_CHANNELS=["train"]
SM_CURRENT_HOST=algo-1
SM_CURRENT_INSTANCE_TYPE=ml.m5.large
SM_CURRENT_INSTANCE_GROUP=homogeneousCluster
SM_CURRENT_INSTANCE_GROUP_HOSTS=["algo-1"]
SM_INSTANCE_GROUPS=["homogeneousCluster"]
SM_INSTANCE_GROUPS_DICT={"homogeneousCluster":{"hosts":["algo-1"],"instance_group_name":"homogeneousCluster","instanc
e_type":"ml.m5.large"}}
SM_DISTRIBUTION_INSTANCE_GROUPS=[]
SM_IS_HETERO=false
SM_MODULE_NAME=train_model
SM_LOG_LEVEL=20
SM_FRAMEWORK_MODULE=sagemaker_sklearn_container.training:main
SM_INPUT_DIR=/opt/ml/input
SM_INPUT_CONFIG_DIR=/opt/ml/input/config
SM_OUTPUT_DIR=/opt/ml/output
SM_NUM_CPUS=2
SM_NUM_GPUS=0
SM_NUM_NEURONS=0
SM_MODEL_DIR=/opt/ml/model
```

```
SM_MODULE_DIR=s3://caip02-ml-bucket-zali/titanic-decision-tree-2025-06-06-19-47-07-355/source/sourcedir.tar.gz
SM_TRAINING_ENV={"additional_framework_parameters":{},"channel_input_dirs":{"train":"/opt/ml/input/data/train"},"curr
ent_host":"algo-1","current_instance_group":"homogeneousCluster","current_instance_group_hosts":["algo-1"],"current_i
nstance_type":"ml.m5.large","distribution_hosts":[],"distribution_instance_groups":[],"framework_module":"sagemaker_s
klearn_container.training:main","hosts":["algo-1"],"hyperparameters":{},"input_config_dir":"/opt/ml/input/config","in
put_data_config":{"train":{"RecordWrapperType":"None","S3DistributionType":"FullyReplicated","TrainingInputMode":"Fil
e"}},"input_dir":"/opt/ml/input","instance_groups":["homogeneousCluster"],"instance_groups_dict":{"homogeneousCluste
r":{"hosts":["algo-1"],"instance_group_name":"homogeneousCluster","instance_type":"ml.m5.large"}},"is_hetero":fals
e,"is_master":true,"is_modelparallel_enabled":null,"is_smddpmprun_installed":false,"is_smddprun_installed":false,"job
_name":"titanic-decision-tree-2025-06-06-19-47-07-355","log_level":20,"master_hostname":"algo-1","model_dir":"/opt/m
l/model","module_dir":"s3://caip02-ml-bucket-zali/titanic-decision-tree-2025-06-06-19-47-07-355/source/sourcedir.tar.
gz","module_name":"train_model","network_interface_name":"eth0","num_cpus":2,"num_gpus":0,"num_neurons":0,"output_dat
a_dir":"/opt/ml/output/data","output_dir":"/opt/ml/output","output_intermediate_dir":"/opt/ml/output/intermediate","r
esource_config":{"current_group_name":"homogeneousCluster","current_host":"algo-1","current_instance_type":"ml.m5.lar
ge","hosts":["algo-1"],"instance_groups":[{"hosts":["algo-1"],"instance_group_name":"homogeneousCluster","instance_ty
pe":"ml.m5.large"}],"network_interface_name":"eth0","topology":null},"user_entry_point":"train_model.py"}
SM_USER_ARGS=[]
SM_OUTPUT_INTERMEDIATE_DIR=/opt/ml/output/intermediate
SM_CHANNEL_TRAIN=/opt/ml/input/data/train
PYTHONPATH=/opt/ml/code:/miniconda3/bin:/miniconda3/lib/python38.zip:/miniconda3/lib/python3.8:/miniconda3/lib/python
3.8/lib-dynload:/miniconda3/lib/python3.8/site-packages
Invoking script with the following command:
/miniconda3/bin/python train_model.py
2025-06-06 19:49:24,877 sagemaker-training-toolkit INFO     Exceptions not imported for SageMaker Debugger as it is n
ot installed.
2025-06-06 19:49:24,879 sagemaker-training-toolkit INFO     Exceptions not imported for SageMaker TF as Tensorflow is
not installed.
=== Classification Report ===
              precision    recall  f1-score   support

         0.0       0.80      0.90      0.84       106
         1.0       0.81      0.67      0.73        72

    accuracy                           0.80       178
   macro avg       0.81      0.78      0.79       178
weighted avg       0.80      0.80      0.80       178

=== Confusion Matrix ===
[[95 11]
 [24 48]]
=== Feature Importances ===
Sex         0.592333
Pclass      0.266138
Age         0.074534
Fare        0.042353
```

```
SibSp       0.024641
Embarked    0.000000
Parch       0.000000
dtype: float64
2025-06-06 19:49:25,862 sagemaker-containers INFO     Reporting training SUCCESS

2025-06-06 19:49:43 Training - Training image download completed. Training in progress.
2025-06-06 19:49:43 Uploading - Uploading generated training model
2025-06-06 19:49:43 Completed - Training job completed
Training seconds: 105
Billable seconds: 105
```

Intro to Inference in SageMaker

After training a model, the next step is **inference** — making predictions from new, unseen data. SageMaker provides multiple deployment options, each suited for different use cases:

Inference Options

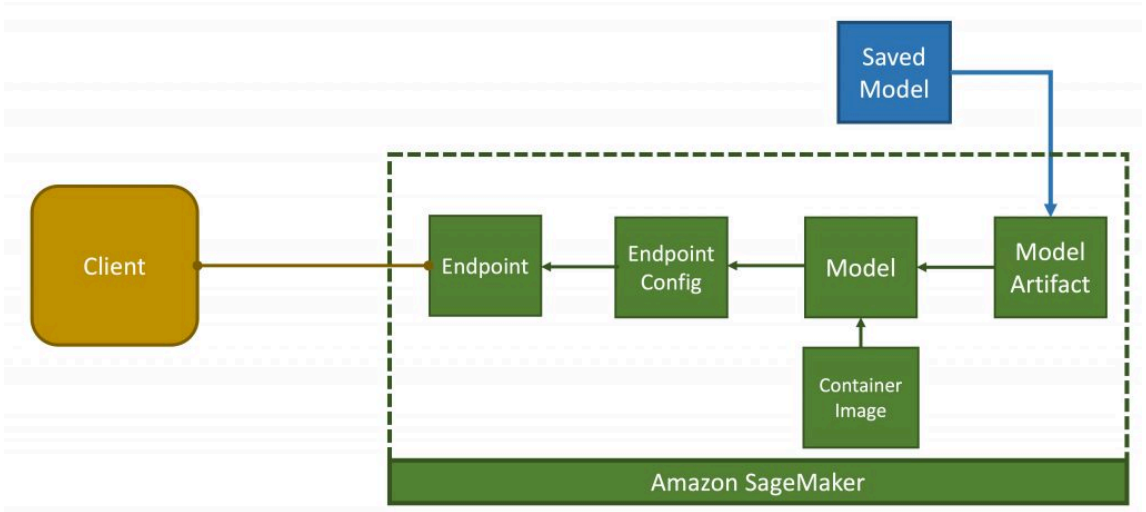
Option	Use Case
Real-Time Endpoint	Instant predictions via API calls (e.g., fraud detection, recommendation systems)
Batch Transform	Run predictions on large datasets stored in S3 (e.g., nightly scoring jobs)
Async Inference	Queue-based, low-latency jobs for large or complex inputs (e.g., documents, images)

In this call, we'll focus on **real-time inference** using a SageMaker-hosted endpoint. This will:

- Load your trained model into memory
- Expose it as a live HTTPS endpoint
- Accept and respond to `InvokeEndpoint` requests with predictions

You'll also learn how to monitor inference activity in **CloudWatch Logs** and how to delete the endpoint to avoid unnecessary charges.

Real-Time Inference Architecture



Load the Trained Model Artifact from S3

Before we can deploy a model to a SageMaker endpoint, we need to locate the **trained model artifact** that was saved during our previous training job.

When you ran the `SKLearn` Estimator in Call 1, SageMaker saved your model (as a `.tar.gz` archive) to the **S3 output path** you specified, typically something like:

`s3://your-bucket-name/models/<training-job-name>/output/model.tar.gz`
This file contains the serialized model object (e.g., a `joblib` or `pickle` dump), and SageMaker uses it to load the model into the deployed container.

In this section, we'll:

- Locate your model artifact in S3
- Confirm that it exists and is ready to deploy
- (Optionally) download and inspect it locally to verify the contents

In [31]:
```python
import boto3

# If using CloudFormation outputs
bucket_name = {o["OutputKey"]: o["OutputValue"] for o in outputs if o["OutputKey"] == "BucketName"}["BucketName"]
model_prefix = "models/"  # Adjust if needed

s3 = boto3.client("s3")

# List all model artifacts under the prefix
response = s3.list_objects_v2(Bucket=bucket_name, Prefix=model_prefix)

# Filter to only model.tar.gz files
model_files = [
    obj for obj in response.get("Contents", [])
    if obj["Key"].endswith("model.tar.gz")
]

# Sort by last modified date (latest first)
model_files_sorted = sorted(model_files, key=lambda x: x["LastModified"], reverse=True)

# Get the most recent one
if model_files_sorted:
    model_artifact = model_files_sorted[0]["Key"]
    print(f"Found latest model artifact: s3://{bucket_name}/{model_artifact}")
else:
    print("No model.tar.gz files found.")
```

```
Found latest model artifact: s3://caip02-ml-bucket-zali/models/titanic-decision-tree-2025-06-06-19-47-07-355/output/model.tar.gz
```

Write the Inference Script (`inference.py`)

By default, SageMaker expects your deployed model to accept preprocessed input. But in our case, the model was trained on **scaled data**, and the raw input we want to send (e.g., `[Pclass, Sex, Age, ...]`) hasn't been scaled yet.

To solve this, we'll bundle a custom `inference.py` script with our model that:

- Loads both the trained model and the fitted `StandardScaler`
- Scales the incoming input using the same logic as training

- Makes and returns predictions

This script runs **inside the container** at inference time and ensures your endpoint can accept **raw, unscaled data** directly.

What You'll Need

1. A copy of the fitted `StandardScaler` saved as `scaler.joblib`
2. A trained model saved as `model.joblib`
3. A custom script named `inference.py` that defines:
   - `model_fn(model_dir)` — loads model and scaler
   - `predict_fn(input_data, model_and_scaler)` — preprocesses and returns prediction

We'll deploy this using the `entry_point` + `source_dir` parameters in the `SKLearnModel`.

In [62]:
```python
from IPython.display import Markdown, display

def show_code(filepath):
    with open(filepath, "r") as f:
        code = f.read()
    display(Markdown(f"```python\n{code}\n```"))

# Call this after writing the file
show_code("inference.py")
```

```python
import joblib
import numpy as np
import os
import json

# Called when the model is loaded
def model_fn(model_dir):
    model = joblib.load(os.path.join(model_dir, "model.joblib"))
    scaler = joblib.load(os.path.join(model_dir, "scaler.joblib"))
    return (model, scaler)

# Called when a prediction is made
def predict_fn(input_data, model_and_scaler):
    model, scaler = model_and_scaler

    # Ensure input is a NumPy array
    if isinstance(input_data, list):
        input_data = np.array(input_data)

    # Apply scaling
    scaled = scaler.transform(input_data)

    # Make prediction
    prediction = model.predict(scaled)

    # Log everything to CloudWatch
    print("Input received:", json.dumps(input_data.tolist()))
    print("Scaled input:", json.dumps(scaled.tolist()))
    print("Prediction result:", json.dumps(prediction.tolist()))

    return prediction.tolist()
```

Deploy a Real-Time Endpoint using `SKLearnModel.deploy()`

Once we have the trained model artifact in S3, we can deploy it to a real-time endpoint using SageMaker's `SKLearnModel` class.

This allows us to:

- Load the `.tar.gz` model artifact into a container
- Launch a managed HTTPS endpoint hosted by SageMaker
- Invoke the endpoint with real-time prediction requests

What Happens Under the Hood

1. SageMaker creates an **endpoint configuration** that links the model, container image, and instance type
2. It launches an **endpoint** that pulls the container + model into memory
3. The endpoint becomes a live API you can call with `invoke_endpoint()` from `boto3`

In this step, we'll:

- Define a `SKLearnModel` object using the path to your model artifact in S3
- Deploy it to an endpoint (e.g., `titanic-endpoint`)
- Print the endpoint name so we can use it in the next step

In [63]:
```
from sagemaker.sklearn.model import SKLearnModel
import sagemaker
import boto3

# Set required parameters
bucket_name = {o["OutputKey"]: o["OutputValue"] for o in outputs if o["OutputKey"] == "BucketName"}["BucketName"]
role = {o["OutputKey"]: o["OutputValue"] for o in outputs if o["OutputKey"] == "RoleArn"}["RoleArn"]
region = boto3.Session().region_name
endpoint_name = "titanic-endpoint"

sm = boto3.client("sagemaker")
```

```python
try:
    sm.delete_endpoint(EndpointName=endpoint_name)
    sm.delete_endpoint_config(EndpointConfigName=endpoint_name)
    print(f"Deleted existing endpoint and config: {endpoint_name}")
except sm.exceptions.ClientError as e:
    if "Could not find" not in str(e):
        raise

# Define model object
model = SKLearnModel(
    model_data=f"s3://{bucket}/{model_artifact}",
    role=role,
    entry_point="inference.py",
    framework_version="1.0-1",
    sagemaker_session=sagemaker.Session()
)


# Deploy the model
predictor = model.deploy(
    initial_instance_count=1,
    instance_type="ml.m5.large",
    endpoint_name="titanic-endpoint"
)

print(f"Deployed endpoint: {endpoint_name}")
```

```
Deleted existing endpoint and config: titanic-endpoint
INFO:sagemaker:Creating model with name: sagemaker-scikit-learn-2025-06-06-21-29-51-030
INFO:sagemaker:Creating endpoint-config with name titanic-endpoint
INFO:sagemaker:Creating endpoint with name titanic-endpoint
------!Deployed endpoint: titanic-endpoint
```

Call the Endpoint

Once your model is deployed as a real-time endpoint, you can make predictions by calling it through SageMaker's `invoke_endpoint` API.

In this step, we'll:

- Prepare a sample payload (one row of Titanic-style input features)
- Call the endpoint using `boto3`
- Parse the prediction result
- (Optional) Display the predicted class and confidence score

This simulates what a client app or API gateway would do to request live predictions from your deployed model.

In [64]:
```python
import boto3
import json
import numpy as np

def predict_survival(payload, endpoint_name="titanic-endpoint"):
    """
    Sends a payload to the specified SageMaker endpoint and returns the prediction.

    Parameters:
        payload (list of list): A 2D list representing one or more passengers.
            Each row should follow the format:
            [Pclass, Sex, Age, SibSp, Parch, Fare, Embarked]
        endpoint_name (str): The name of the deployed SageMaker endpoint.

    Returns:
        float or int: The first predicted value from the endpoint response.
        Typically 0.0 or 1.0, or a probability depending on model output.
    """

    # Convert payload to JSON and call endpoint
    runtime = boto3.client("sagemaker-runtime")
    response = runtime.invoke_endpoint(
        EndpointName=endpoint_name,
        ContentType="application/json",
        Body=json.dumps(payload)
    )

    # Read and decode response
    result = response["Body"].read().decode("utf-8")
    prediction = json.loads(result)
    return prediction[0]
```

```python
def print_prediction_summary(data_rows, prediction):
    """
    Pretty-prints the passenger data and model prediction.

    Parameters:
        data_rows (list): A list of input rows. Each row is a list of:
            [Pclass, Sex, Age, SibSp, Parch, Fare, Embarked]
        prediction (float): The raw prediction score (e.g., probability of survival).
    """
    columns = ["Pclass", "Sex", "Age", "SibSp", "Parch", "Fare", "Embarked"]

    # Mappings for readability
    pclass_map = {1: "1st Class", 2: "2nd Class", 3: "3rd Class"}
    sex_map = {0: "Male", 1: "Female"}
    embarked_map = {0: "Southampton (S)", 1: "Cherbourg (C)", 2: "Queenstown (Q)"}

    predicted_class = int(np.round(prediction))

    for row in data_rows:
        if len(row) != len(columns):
            print("Invalid data row: must contain 7 values.")
            continue

        print("Passenger Info:")
        for col, val in zip(columns, row):
            readable = val
            if col == "Pclass":
                readable = pclass_map.get(val, f"Unknown ({val})")
            elif col == "Sex":
                readable = sex_map.get(val, f"Unknown ({val})")
            elif col == "Embarked":
                readable = embarked_map.get(val, f"Unknown ({val})")
            print(f"  {col}: {val} -> {readable}")

        outcome = "Survived" if predicted_class == 1 else "Did Not Survive"
        print(f"\nPrediction: {prediction:.4f} → Class: {predicted_class} ({outcome})")
        print("-" * 40)


# Sample input: [Pclass, Sex, Age, SibSp, Parch, Fare, Embarked]
# This passenger is a 22-year-old male in 3rd class, traveling alone with a low fare.
# Historically, this profile had a lower survival rate on the Titanic.
```

```
test_passenger = [[3, 0, 22.0, 1, 0, 7.25, 0]]
prediction = predict_survival(test_passenger)
print_prediction_summary(test_passenger, prediction)
```

```
Passenger Info:
  Pclass: 3 -> 3rd Class
  Sex: 0 -> Male
  Age: 22.0 -> 22.0
  SibSp: 1 -> 1
  Parch: 0 -> 0
  Fare: 7.25 -> 7.25
  Embarked: 0 -> Southampton (S)

Prediction: 0.0000 → Class: 0 (Did Not Survive)
----------------------------------------
```

In [65]:
```
test_passenger = [[1, 0, 65.0, 0, 0, 82.0, 1]]  # 1st class, male, 65 y/o, traveling alone
prediction = predict_survival(test_passenger)
print_prediction_summary(test_passenger, prediction)
```

```
Passenger Info:
  Pclass: 1 -> 1st Class
  Sex: 0 -> Male
  Age: 65.0 -> 65.0
  SibSp: 0 -> 0
  Parch: 0 -> 0
  Fare: 82.0 -> 82.0
  Embarked: 1 -> Cherbourg (C)

Prediction: 0.0000 → Class: 0 (Did Not Survive)
----------------------------------------
```

In [66]:
```
test_passenger = [[3, 1, 19.0, 0, 0, 7.75, 2]]  # 3rd class, female, 19 y/o, low fare
prediction = predict_survival(test_passenger)
print_prediction_summary(test_passenger, prediction)
```

```
Passenger Info:
  Pclass: 3 -> 3rd Class
  Sex: 1 -> Female
  Age: 19.0 -> 19.0
  SibSp: 0 -> 0
  Parch: 0 -> 0
  Fare: 7.75 -> 7.75
  Embarked: 2 -> Queenstown (Q)

Prediction: 1.0000 → Class: 1 (Survived)
----------------------------------------
```

In [67]:
```
test_passenger = [[1, 1, 38.0, 1, 1, 153.46, 1]]  # 1st class, female, adult with child, high fare
prediction = predict_survival(test_passenger)
print_prediction_summary(test_passenger, prediction)
```

```
Passenger Info:
  Pclass: 1 -> 1st Class
  Sex: 1 -> Female
  Age: 38.0 -> 38.0
  SibSp: 1 -> 1
  Parch: 1 -> 1
  Fare: 153.46 -> 153.46
  Embarked: 1 -> Cherbourg (C)

Prediction: 1.0000 → Class: 1 (Survived)
----------------------------------------
```

Review Logs in CloudWatch

Just like with training jobs, SageMaker captures logs from deployed inference endpoints and sends them to **Amazon CloudWatch**. These logs include:

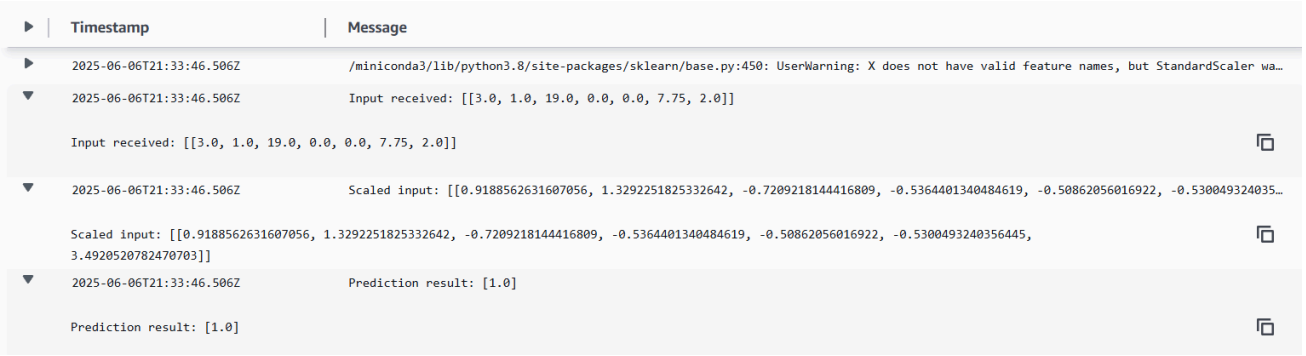
- Container startup events
- Any `print()` statements inside your inference script (if using a custom container)
- Internal SageMaker logging (including input/output size, latency, errors)

How to Access Endpoint Logs

1. Go to the SageMaker Console
2. Click **Endpoints** in the left sidebar
3. Select your endpoint name (e.g., `titanic-endpoint`)
4. Scroll down to **Monitor**
5. Click the **View logs** link next to the production variant
6. You'll be redirected to **CloudWatch Logs** where you can inspect activity

---

Inference Log Screenshot (3rd class, female, 19 y/o, low fare)



Timestamp	Message
2025-06-06T21:33:46.506Z	/miniconda3/lib/python3.8/site-packages/sklearn/base.py:450: UserWarning: X does not have valid feature names, but StandardScaler wa…
2025-06-06T21:33:46.506Z	Input received: [[3.0, 1.0, 19.0, 0.0, 0.0, 7.75, 2.0]]
	Input received: [[3.0, 1.0, 19.0, 0.0, 0.0, 7.75, 2.0]]
2025-06-06T21:33:46.506Z	Scaled input: [[0.9188562631607056, 1.3292251825332642, -0.7209218144416809, -0.5364401340484619, -0.50862056016922, -0.530049324035…
	Scaled input: [[0.9188562631607056, 1.3292251825332642, -0.7209218144416809, -0.5364401340484619, -0.50862056016922, -0.530049324035 6445, 3.4920520782470703]]
2025-06-06T21:33:46.506Z	Prediction result: [1.0]
	Prediction result: [1.0]

CloudWatch logs are especially helpful for catching input shape mismatches, serialization issues, or performance bottlenecks.

> Pro tip: Set up **CloudWatch Alarms** to alert if an endpoint returns too many errors or takes too long to respond.

Common Errors and Metrics in Endpoint Logs

When calling a SageMaker endpoint, your logs in CloudWatch can reveal a lot about what's working — and what's broken.

Common Inference Errors

Error Type	Cause / Fix
ModelError	Your script raised an unhandled exception (e.g., input shape mismatch, bad `joblib` file)

Error Type	Cause / Fix
ValidationError	You sent invalid JSON, missing required fields, or wrong `ContentType`
InternalFailure	A container crash — often due to missing files, dependencies, or bad serialization
ThrottlingException	You're sending too many requests for your instance size (scale up or batch inputs)
EndpointConnectionError	Your endpoint was deleted or not yet ready (check deployment status)

Metrics to Monitor

Metric	What It Tells You
`ModelLatency`	Time spent inside the model container (ms)
`OverheadLatency`	Time spent outside the container (network, etc.)
`Invocation4XXErrors`	Client-side errors (bad input, wrong format)
`Invocation5XXErrors`	Server-side errors (model crashed, etc.)
`Invocations`	Number of successful requests
`InvocationThrottled`	You're hitting your instance's request limit
`MemoryUtilization`	If your container is running out of RAM

You can view these metrics under **SageMaker → Endpoints → Monitoring** or directly in **CloudWatch Metrics**.

> In production, it's best practice to build CloudWatch **Alarms** or **Dashboards** around these metrics — especially `Invocation5XXErrors` and `ModelLatency`.

Review Metrics in CloudWatch

In addition to logs, SageMaker automatically sends a variety of **real-time performance and operational metrics** to **Amazon CloudWatch**.

These metrics help you monitor how your endpoint is behaving — whether it's receiving traffic, how fast it's responding, and whether it's throwing errors.

How to Access Endpoint Metrics

1. Go to the CloudWatch Console → Metrics
2. Choose **SageMaker**
3. Select **Endpoints** → **EndpointName, VariantName**
4. Choose your endpoint (e.g., `titanic-endpoint`)
5. You can now graph metrics and set alarms

Common Metrics to Monitor

Metric Name	Description
`Invocations`	Number of inference requests received
`ModelLatency`	Time (ms) spent running the model in the container
`OverheadLatency`	Time spent outside the model (e.g., networking)
`Invocation4XXErrors`	Client errors (bad input, invalid format, etc.)
`Invocation5XXErrors`	Server errors (container crashes, exceptions)
`MemoryUtilization` *	Memory usage of the container (if enabled)

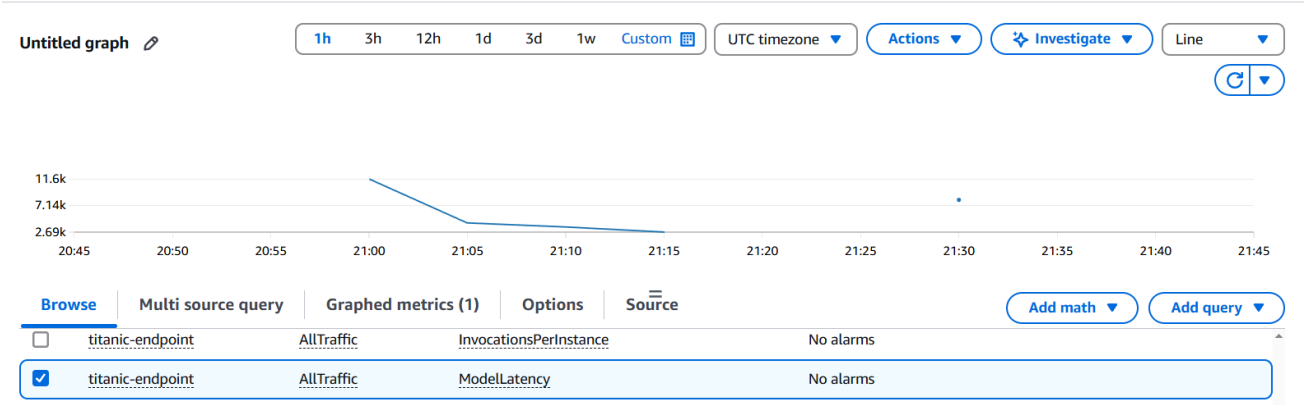
> * `MemoryUtilization` only shows up if your container is configured to emit resource metrics.

These metrics help you:

- Detect performance issues (e.g., slow inference or memory spikes)
- Identify input or model errors
- Avoid idle costs or unexpected spikes in usage

You can also build dashboards or alarms directly from these metrics.

Cloudwatch Metrics Screenshot (ModelLatency)



Delete the Endpoint Using `.delete_endpoint()` or `boto3`

SageMaker real-time endpoints are **always-on** and billed by the hour, even if they're idle. To avoid unnecessary charges, it's important to delete the endpoint when you're done testing.

You can delete the endpoint in two ways:

- Use the `delete_endpoint()` method from your deployed `predictor` object
- Use the `boto3` SDK to delete it by name

We'll show both methods below.

In [68]:
```
import boto3

sm = boto3.client("sagemaker")

try:
    sm.delete_endpoint(EndpointName=endpoint_name)
    sm.delete_endpoint_config(EndpointConfigName=endpoint_name)
    print(f"Deleted existing endpoint and config: {endpoint_name}")
```

```
except sm.exceptions.ClientError as e:
    if "Could not find" not in str(e):
        raise
print(f"Endpoint '{endpoint_name}' deleted using boto3.")
```

```
Deleted existing endpoint and config: titanic-endpoint
Endpoint 'titanic-endpoint' deleted using boto3.
```

Cleanup

Now that you've successfully trained and evaluated a model in the cloud, you've completed your first end-to-end managed ML workflow using SageMaker.

Cleanup (Don't Leave the Lights On)

To avoid unnecessary AWS charges, we'll delete all the resources we created:

- **CloudFormation Stack**:
  This will automatically delete:
  - The SageMaker Studio domain
  - The execution role
  - The S3 bucket
    You can do this programmatically using `boto3` (recommended), or...
    if you absolutely must... use the CloudFormation Console like a savage.
- **CloudWatch Logs**:
  These are optional to clean up, but can be removed from the CloudWatch Console if desired.

In [69]:
```
import boto3

def delete_stack_and_wait(stack_name):
    print(f"🧨 Deleting stack: {stack_name}")

    # Initiate deletion
    cf.delete_stack(StackName=stack_name)
```

```
    # Wait until stack deletion is complete
    waiter = cf.get_waiter("stack_delete_complete")
    print("⏳ Waiting for stack to be fully deleted...")
    waiter.wait(StackName=stack_name)

    print(f"✅ Stack '{stack_name}' successfully deleted.")

cf = boto3.client("cloudformation")
s3 = boto3.client('s3')

bucket_name = {o["OutputKey"]: o["OutputValue"] for o in outputs if o["OutputKey"] == "BucketName"}["BucketName"]

# List and delete objects in batches of 1000
while True:
    response = s3.list_objects_v2(Bucket=bucket_name)
    if 'Contents' not in response:
        break

    delete_keys = [{'Key': obj['Key']} for obj in response['Contents']]

    s3.delete_objects(
        Bucket=bucket_name,
        Delete={'Objects': delete_keys}
    )

    if not response.get('IsTruncated'):
        break

print(f"All objects in bucket '{bucket_name}' deleted.")

# Call it
delete_stack_and_wait(stack_name)
```

```
All objects in bucket 'caip02-ml-bucket-zali' deleted.
🧨 Deleting stack: caip02-cloud-ml-stack
⏳ Waiting for stack to be fully deleted...
✅ Stack 'caip02-cloud-ml-stack' successfully deleted.
```

Wrap-Up & Takeaways

In this notebook, you built and deployed your first real-time ML inference service using Amazon SageMaker. You trained a model, deployed it to a secure endpoint, sent a live prediction request, and reviewed logs — all using scalable, cloud-native tools.

This workflow reflects what real ML teams do in production:

- Train models offline and save versioned artifacts to cloud storage
- Deploy models to managed endpoints with autoscaling and monitoring
- Invoke models from client apps or internal APIs using standardized formats
- Monitor latency, errors, and prediction volume with CloudWatch
- Tear down unused endpoints to control cost and surface stale deployments
- Use repeatable infrastructure (e.g., scripts, IaC, automation) instead of manual clicking