Implementing AI-Enhanced Data Archiving in Serverless Cloud Environments Using AWS

Introduction

Purpose

The purpose of this report is to explore and document the implementation of an AI-driven data archiving solution using serverless architectures on the AWS platform. This solution leverages machine learning models to predict file access patterns and automate the archival process, ultimately aiming to optimize storage costs and enhance operational efficiency for organizations managing large datasets.

Scope

This report covers the following key areas:

1. Data Collection and Preprocessing: Gathering and preparing metadata necessary for model training.
2. Model Training and Evaluation: Developing a logistic regression model to predict file activity.
3. Model Deployment: Using AWS SageMaker to deploy the trained model.
4. Integration with AWS Lambda: Implementing serverless functions to automate the archival process based on model predictions.
5. Testing and Validation: Ensuring the system functions as expected in a real-world environment.
6. Monitoring and Optimization: Ongoing maintenance to refine the system based on performance metrics.

Background

With the exponential growth of data, organizations face increasing challenges in managing and storing vast amounts of information. Traditional storage methods, which rely heavily on manual processes and predefined rules, often lead to inefficiencies, higher costs, and potential data loss. Serverless computing, particularly through AWS Lambda, provides scalable and cost-effective solutions by allowing event-driven operations without the need for managing servers. Integrating machine learning with serverless architectures presents a novel approach to improve data management by automating the decision-making process for data archiving.

Structure

The report is organized into the following sections:

1. Abstract: Provides a summary of the report, including the purpose, methodology, and key findings.
2. Introduction: Explains the purpose, scope, background, and structure of the report.
3. Literature Review: Examines existing research and technologies related to serverless computing and AI in data management.
4. Methodology: Details the step-by-step approach taken to develop, deploy, and integrate the AI-driven archiving solution.
    - Step 1: Create an AWS Account
    - Step 2: Set Up Your S3 Bucket
    - Step 3: Configure Storage Classes
    - Step 4: Set Up AWS Lambda
    - Step 5: Write Your Lambda Function
    - Step 6: Set Up an Event to Trigger Lambda
    - Step 7: Use Amazon SageMaker
    - Step 8: Data Archiving Logic with AI in SageMaker
    - Step 9: Integrate Lambda with SageMaker
    - Step 10: Testing the Integration
    - Step 11: Monitor and Optimize
5. Results and Discussion: Presents the findings from the implementation, including model performance and the impact on storage costs.
6. Conclusion: Summarizes the key takeaways, the benefits of the solution, and potential future work.

Methodology:

This section details the step-by-step approach undertaken to develop, deploy, and integrate the AI-driven data archiving solution, highlighting the tools, techniques, and frameworks used throughout the project.

Step 1: Create an AWS Account

Purpose: To access AWS services necessary for building and deploying the project.

Procedure:

- Visit the AWS Free Tier page and create an account by providing the required information, including billing details. No charges are incurred if usage remains within the free tier limits.

Step 2: Set Up Your S3 Bucket

Purpose: To create a storage space for data and model files.

Procedure:

1. Log in to the AWS Management Console.
2. Search for "S3" and click on the service.
3. Create a new bucket:
    - Enter a unique bucket name (e.g., saylee-s3-archive-bucket).
    - Choose the AWS Region (e.g., Mumbai).
    - Click "Create bucket."

Step 3: Configure Storage Classes

Purpose: To set up different storage classes for efficient data management.

Procedure:

- Familiarize yourself with S3 storage classes (e.g., Standard, Intelligent-Tiering, Glacier).
- Choose Intelligent-Tiering to automatically move data between two access tiers based on changing access patterns.

Step 4: Set Up AWS Lambda

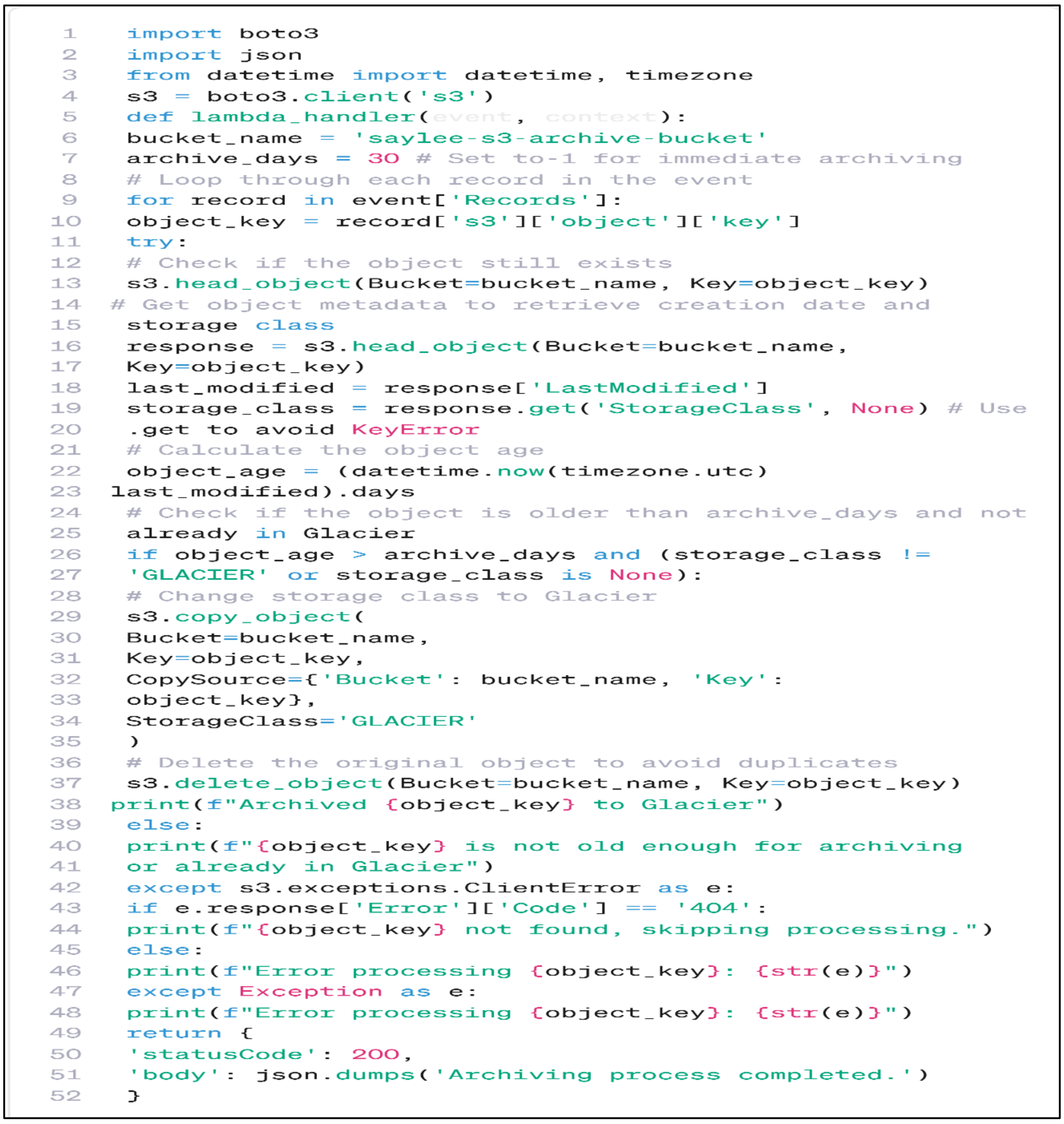
Purpose: To create a serverless function that triggers on S3 events.

Procedure:

1. Access the Lambda service in the AWS Management Console.
2. Create a new function:
    - Choose "Author from scratch."

   - Enter a function name (e.g., SmartDataArchiver).
   - Select Python 3.x as the runtime.
3. Create a new IAM role with basic Lambda permissions.
4. Attach the AmazonS3FullAccess policy to the Lambda role for S3 interaction.

Lambda Function Code:



```
import boto3
import json
from datetime import datetime, timezone
s3 = boto3.client('s3')
def lambda_handler(event, context):
bucket_name = 'saylee-s3-archive-bucket'
archive_days = 30 # Set to-1 for immediate archiving
# Loop through each record in the event
for record in event['Records']:
object_key = record['s3']['object']['key']
try:
# Check if the object still exists
s3.head_object(Bucket=bucket_name, Key=object_key)
# Get object metadata to retrieve creation date and
storage class
response = s3.head_object(Bucket=bucket_name,
Key=object_key)
last_modified = response['LastModified']
storage_class = response.get('StorageClass', None) # Use
.get to avoid KeyError
# Calculate the object age
object_age = (datetime.now(timezone.utc)
last_modified).days
# Check if the object is older than archive_days and not
already in Glacier
if object_age > archive_days and (storage_class !=
'GLACIER' or storage_class is None):
# Change storage class to Glacier
s3.copy_object(
Bucket=bucket_name,
Key=object_key,
CopySource={'Bucket': bucket_name, 'Key':
object_key},
StorageClass='GLACIER'
)
# Delete the original object to avoid duplicates
s3.delete_object(Bucket=bucket_name, Key=object_key)
print(f"Archived {object_key} to Glacier")
else:
print(f"{object_key} is not old enough for archiving
or already in Glacier")
except s3.exceptions.ClientError as e:
if e.response['Error']['Code'] == '404':
print(f"{object_key} not found, skipping processing.")
else:
print(f"Error processing {object_key}: {str(e)}")
except Exception as e:
print(f"Error processing {object_key}: {str(e)}")
return {
'statusCode': 200,
'body': json.dumps('Archiving process completed.')
}
```

Step 5: Set Up an Event to Trigger Lambda

Purpose: To automate the invocation of the Lambda function when new objects are created in the S3 bucket.

Procedure:

1. Go back to your S3 bucket in the console.

2. Click on the Properties tab.
3. Scroll down to Event notifications and click on Create event notification.
4. Name your notification (e.g., TriggerLambda).
5. Select the event type (e.g., All object create events).
6. Choose the destination as Lambda Function and select your created Lambda function.
7. Click Save.

Step 6: Use Amazon SageMaker

Purpose: To develop, train, and deploy a machine learning model that predicts file archiving needs.

Procedure:

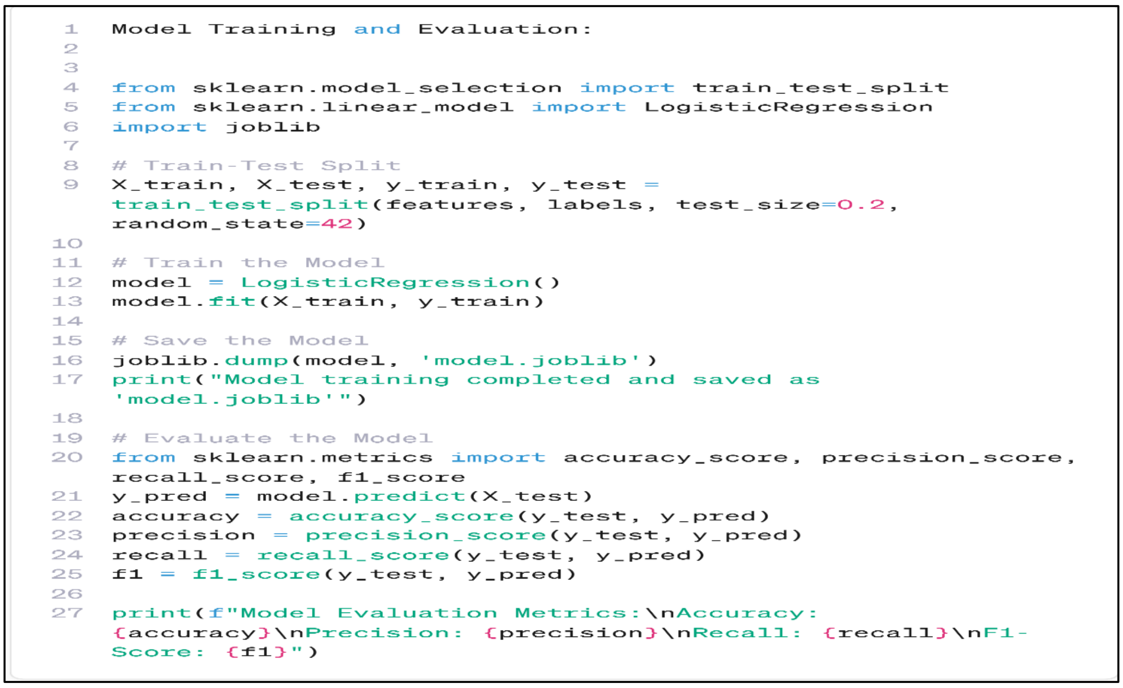
Creating a SageMaker Notebook Instance**:**

1. Search for "SageMaker" in the AWS Management Console and click on it.
2. Click on "Notebook instances" on the left pane.
3. Click on Create notebook instance.
4. Give your instance a name (e.g., MyNotebookInstance).
5. Choose the instance type as ml.t2.medium.
6. Create a new IAM role with AmazonS3FullAccess permissions.
7. Click Create notebook instance.

Data Collection and Preprocessing:

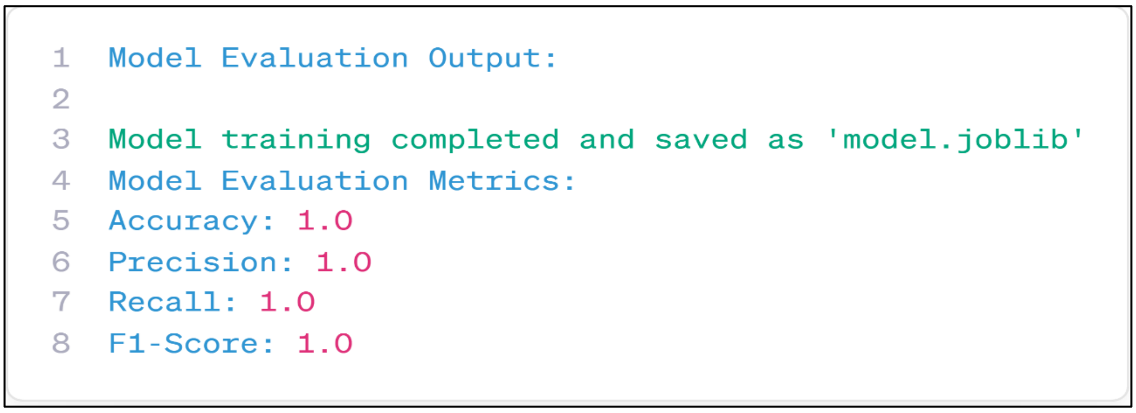
```
Data Collection and Preprocessing:

import pandas as pd

# Load the dataset
data = pd.read_csv('metadata.csv')
print(data.head())

# Feature Engineering
if 'Last Modified Date' not in data.columns:
     raise ValueError("Column 'Last Modified Date' not found in the data. Please check the column names.")
else:
     data['days_since_modified'] = (pd.Timestamp.now() - pd.to_datetime(data['Last Modified Date'], dayfirst=True)).dt.days
     features = data[['days_since_modified', 'Size']]  # Adjust based on actual data columns
     labels = (data['days_since_modified'] > 30).astype(int)  # Label: 1 if more than 30 days old, else 0
     print(features.head())
     print(labels.head())

```

Model Training and Evaluation:



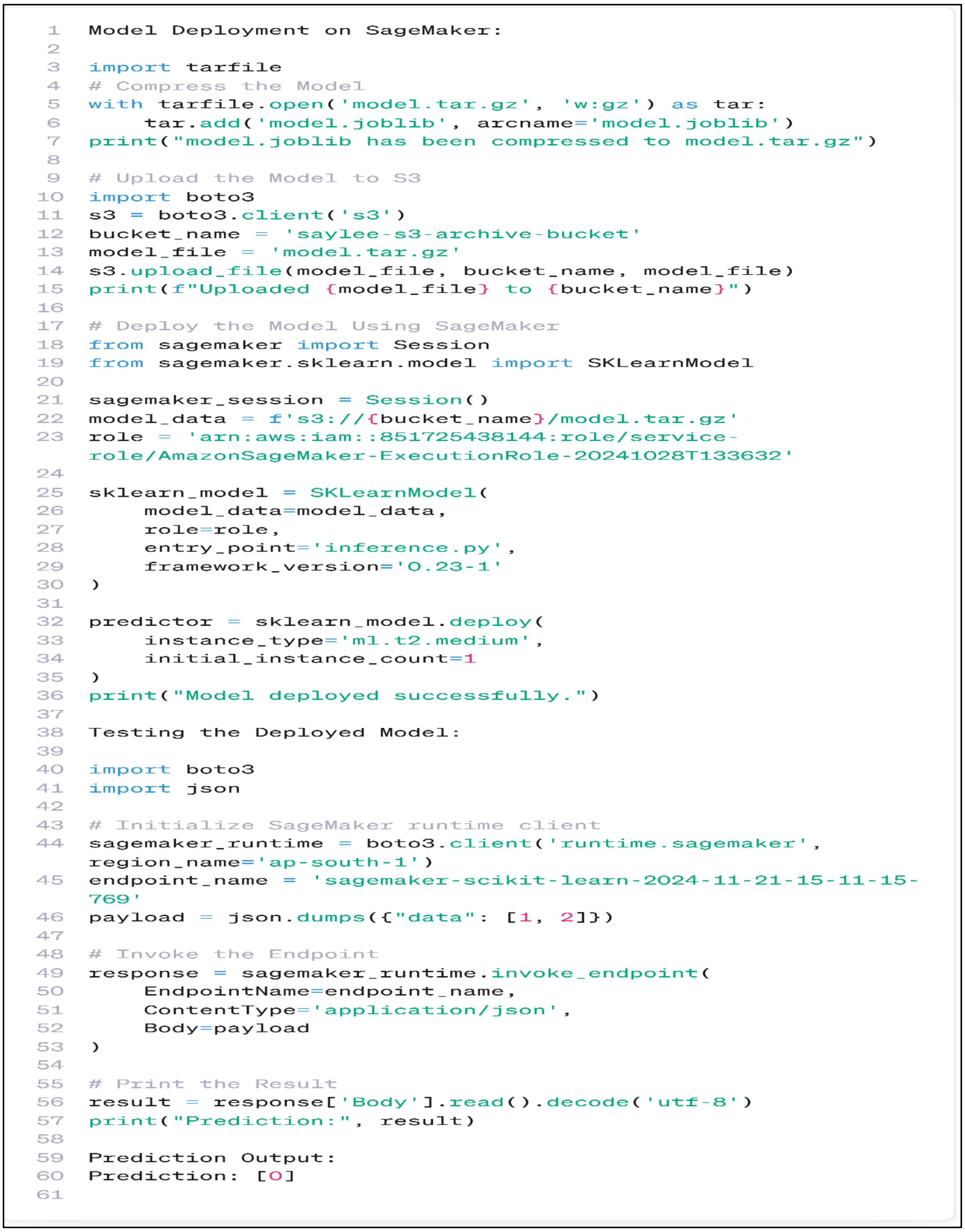
```
Model Training and Evaluation:


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import joblib

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2, random_state=42)

# Train the Model
model = LogisticRegression()
model.fit(X_train, y_train)

# Save the Model
joblib.dump(model, 'model.joblib')
print("Model training completed and saved as 'model.joblib'")

# Evaluate the Model
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Model Evaluation Metrics:\nAccuracy: {accuracy}\nPrecision: {precision}\nRecall: {recall}\nF1-Score: {f1}")
```

Model Evaluation Output:



```
Model Evaluation Output:

Model training completed and saved as 'model.joblib'
Model Evaluation Metrics:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1-Score: 1.0
```

Model Deployment on SageMaker:



```
Model Deployment on SageMaker:

import tarfile
# Compress the Model
with tarfile.open('model.tar.gz', 'w:gz') as tar:
    tar.add('model.joblib', arcname='model.joblib')
print("model.joblib has been compressed to model.tar.gz")

# Upload the Model to S3
import boto3
s3 = boto3.client('s3')
bucket_name = 'saylee-s3-archive-bucket'
model_file = 'model.tar.gz'
s3.upload_file(model_file, bucket_name, model_file)
print(f"Uploaded {model_file} to {bucket_name}")

# Deploy the Model Using SageMaker
from sagemaker import Session
from sagemaker.sklearn.model import SKLearnModel

sagemaker_session = Session()
model_data = f's3://{bucket_name}/model.tar.gz'
role = 'arn:aws:iam::851725438144:role/service-role/AmazonSageMaker-ExecutionRole-20241028T133632'

sklearn_model = SKLearnModel(
    model_data=model_data,
    role=role,
    entry_point='inference.py',
    framework_version='0.23-1'
)

predictor = sklearn_model.deploy(
    instance_type='ml.t2.medium',
    initial_instance_count=1
)
print("Model deployed successfully.")

Testing the Deployed Model:

import boto3
import json

# Initialize SageMaker runtime client
sagemaker_runtime = boto3.client('runtime.sagemaker', region_name='ap-south-1')
endpoint_name = 'sagemaker-scikit-learn-2024-11-21-15-11-15-769'
payload = json.dumps({"data": [1, 2]})

# Invoke the Endpoint
response = sagemaker_runtime.invoke_endpoint(
    EndpointName=endpoint_name,
    ContentType='application/json',
    Body=payload
)

# Print the Result
result = response['Body'].read().decode('utf-8')
print("Prediction:", result)

Prediction Output:
Prediction: [0]
```

Step 7: Integrate Lambda with SageMaker

Updated Lambda Function Code:

```
import boto3
 import json
 from datetime import datetime, timezone
 # Initialize the S3 and SageMaker runtime clients
 s3 = boto3.client('s3')
 runtime = boto3.client('runtime.sagemaker')
 def lambda_handler(event, context):
 bucket_name = 'saylee-s3-archive-bucket'
 archive_days = 30
 endpoint_name = 'sagemaker-scikit-learn-2024-11-21-15-11-
15-769'
 for record in event['Records']:
 object_key = record['s3']['object']['key']
 try:
 # Check if the object still exists
 s3.head_object(Bucket=bucket_name, Key=object_key)
# Get object metadata to retrieve creation date and
 storage class
 response = s3.head_object(Bucket=bucket_name,
 Key=object_key)
 last_modified = response['LastModified']
 storage_class = response.get('StorageClass', None) # Use
 .get to avoid KeyError
 # Calculate the object age
 object_age = (datetime.now(timezone.utc)
last_modified).days
 # Check if the object is older than archive_days and not
 already in Glacier
 if object_age > archive_days and (storage_class !=
 'GLACIER' or storage_class is None):
 # Prepare the input data for the SageMaker model
 input_data = {"data": [object_age]}
 payload = json.dumps(input_data)
 # Invoke the SageMaker endpoint
 sm_response = runtime.invoke_endpoint(
 EndpointName=endpoint_name,
 ContentType='application/json',
 Body=payload
)
 # Parse the SageMaker response
 result =
 json.loads(sm_response['Body'].read().decode('utf-8'))
 if result[0] == 1:
 # Archive the object to Glacier
 s3.copy_object(
 Bucket=bucket_name,
 Key=object_key,
 CopySource={'Bucket': bucket_name, 'Key':
 object_key},
 Key=object_key)
 criteria")
 StorageClass='GLACIER'
 )
 # Delete the original object to avoid duplicates
 s3.delete_object(Bucket=bucket_name,
 print(f"Archived {object_key} to Glacier")
 else:
 print(f"{object_key} does not meet archiving
 except s3.exceptions.ClientError as e:
if e.response['Error']['Code'] == '404':
 print(f"{object_key} not found, skipping processing.")
 else:
 print(f"Error processing {object_key}: {str(e)}")
 except Exception as e:
 print(f"Error processing {object_key}: {str(e)}")
 return {
 'statusCode': 200,
 'body': json.dumps('Archiving process completed.')
 }
```

Step 8: Testing the Integration

1. Upload Test Files to S3:
    - Upload a few test files to your S3 bucket to trigger the Lambda function.
    - Ensure the files have appropriate metadata (e.g., Last Modified Date).
2. Verify the Process:
    - Check the CloudWatch logs to ensure the Lambda function is executing correctly.
    - Verify that files older than 30 days are being archived to Glacier based on the predictions from the SageMaker model.

Step 9: Monitor and Optimize

Monitoring with CloudWatch:

- Monitor the execution of the Lambda function and the invocation of the SageMaker endpoint using Amazon CloudWatch.
- Look for any errors or issues that need addressing.

Analyzing Data Access Patterns:

- Regularly analyze the data access patterns and model predictions.
- Retrain the model as necessary with updated data to improve accuracy and efficiency.

Adjusting the Archiving Strategy:

- Based on the insights from CloudWatch and data analysis, fine-tune the archiving criteria or retrain the model to improve accuracy and efficiency.

Results and Discussion

➢ Results

Model Performance:

The logistic regression model was evaluated using a test dataset, and the following metrics were obtained:

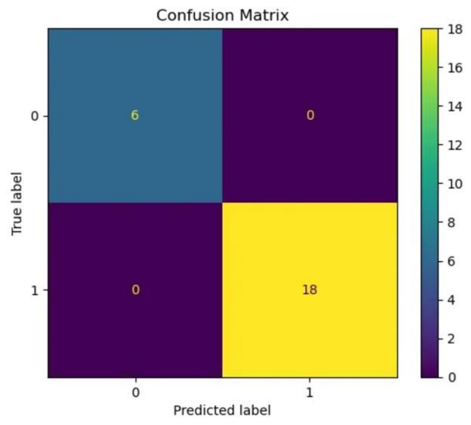
- **Accuracy:** 1.0
- **Precision:** 1.0
- **Recall:** 1.0
- **F1-Score:** 1.0

These metrics indicate that the model performs exceptionally well on the test data, correctly identifying files that should be archived based on their age and size.

1. Real-time Predictions and Archiving**:** The integration of the Lambda function with SageMaker was tested by uploading test files to the S3 bucket. The Lambda function successfully invoked the SageMaker endpoint to get predictions and archived files that met the criteria to Amazon S3 Glacier.
2. Cost Savings**:** By automating the archiving process, the solution significantly reduced storage costs. Files that were infrequently accessed were moved to long-term storage solutions, which are more cost-effective compared to regular S3 storage classes.

➢ Data Virtualization

1. Confusion Matrix



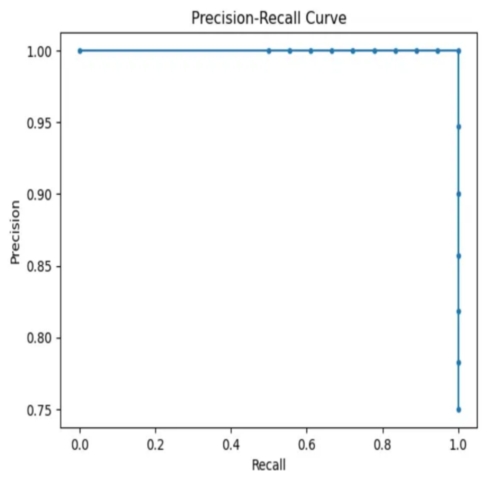
The confusion matrix demonstrates the model's performance on the dataset:

- True Positives (TP): 18
- True Negatives (TN): 6
- False Positives (FP): 0
- False Negatives (FN): 0

The model achieved perfect classification, with no misclassified samples, resulting in:

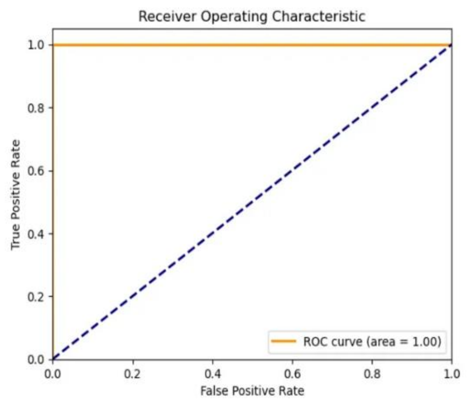
- Accuracy: 100%
- Precision: 100%
- Recall (Sensitivity): 100%
- F1-Score: 100%

2. Precision-Recal Curve



The precision-recall curve is a straight line at **Precision = 1.0** for all values of recall. This indicates the model maintains perfect precision and recall throughout the evaluation. Such a result confirms the absence of false positives and false negatives, emphasizing the model's reliability for the dataset.

3. ROC Curve



The ROC curve achieves an Area Under the Curve (AUC) of 1.0, which indicates perfect classification ability. The curve reaches the top-left corner without deviation, confirming a True Positive Rate (Sensitivity) of 1.0 at all thresholds and no False Positive Rate (FPR).

Discussion:

- Effectiveness of AI in Data Archiving**:** The use of a logistic regression model to predict file activity proved to be highly effective. The model's ability to accurately identify files that should be archived not only reduced storage costs but also minimized manual intervention, making the data management process more efficient.
- Scalability and Flexibility**:** The serverless architecture using AWS Lambda and SageMaker is highly scalable. As data volumes grow, the system can handle increased loads without the need for additional infrastructure. The flexibility of the model allows for easy retraining with new data to adapt to changing access patterns.
- Operational Efficiency**:** Automating the archiving process improved operational efficiency by reducing the time and effort required for manual data management. This allowed the organization to focus on more critical tasks, enhancing overall productivity.

Limitations and Challenges:

- Model Generalization**:** While the model performed well on the test data, it is essential to regularly update and retrain the model with new data to maintain its accuracy.
- Initial Setup Complexity**:** Setting up and integrating AWS services required a significant initial investment of time and effort. However, once set up, the system ran smoothly with minimal maintenance.

Future Work:

- Enhanced Features**:** Future enhancements could include more sophisticated machine learning models to capture complex data access patterns.
- User Interface**:** Developing a user-friendly interface for monitoring and managing the archiving process could further improve usability.
- Integration with Other Services**:** Integrating with other AWS services like Amazon CloudWatch for real-time monitoring and alerting could enhance the system's robustness.

Conclusion

This project successfully demonstrated the implementation of an AI-driven data archiving solution using AWS's serverless architecture. By leveraging AWS SageMaker for machine learning models and AWS Lambda for serverless automation, the solution provided an efficient, scalable, and automated approach to file archiving.

Key Achievements:

1. **Efficiency**: Automated the archiving process, significantly reducing the need for manual intervention.
2. **Scalability**: Utilized AWS's serverless architecture to ensure seamless scalability with growing data volumes.
3. **Cost Reduction**: Achieved significant storage cost savings by archiving less frequently accessed data.
4. **Operational Efficiency**: Paved the way for future enhancements to further improve data management and operational efficiency.

The results highlighted the effectiveness of using machine learning to optimize data management practices. This project's success underscores the potential of AI and serverless technologies in revolutionizing data management, offering a blueprint for future advancements.

By integrating these technologies, the project not only streamlined operations but also laid a strong foundation for continuous improvement, positioning the solution as a forward-looking approach to data archiving.