

NAME : Naviya Dharshini


Roll no : 23AD083

DAY -3

## **CLOUDWATCH , LAMBDA,SAGEMAKER**

TECH STACK:

- S3: Storage for raw & processed data
  - Lambda: Preprocessing & Inference
  - SageMaker Notebook: Model training
  - API Gateway: Serve predictions
  - IAM: Permissions & roles
  - CloudWatch: Debug logs
- 

 STEP 1: S3 Setup

Buckets You'll Use

- **ipl-mlops-data** – Raw and processed data
- Optional: **football-mlops-data** (if you clone for football later)

Successfully created bucket "ipl-mlops-data"  
To upload files and folders, or to configure additional bucket settings, choose [View details](#).

**Account snapshot - updated every 24 hours** All AWS Regions [View Storage Lens dashboard](#)  
Storage lens provides visibility into storage usage and activity trends. Metrics don't include directory buckets. [Learn more](#)

**General purpose buckets** | Directory buckets

**General purpose buckets (1)** All AWS Regions  
Buckets are containers for data stored in S3.

Find buckets by name

	Name	AWS Region	IAM Access Analyzer	Creation date
<input type="radio"/>	<a href="#">ipl-mlops-data</a>	Europe (Stockholm) eu-north-1	<a href="#">View analyzer for eu-north-1</a>	July 4, 2025, 11:52:35 (UTC+05:30)

**ipl-mlops-data** Info

**Objects** | Properties | Permissions | Metrics | Management | Access Points

**Objects (1)** Copy S3 URI Copy URL Download Open Delete Actions Create folder Upload  
Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Find objects by prefix

	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	<a href="#">ipl_teams.csv</a>	csv	July 4, 2025, 11:54:38 (UTC+05:30)	87.0 B	Standard

## Files to Upload

- `ipl_data.csv` → Format:

csv

CopyEdit

team,ipl\_wins,ipl\_team\_score

CSK,5,3000

MI,5,3200

RCB,0,2900

LSG,6,3000 <-- Add this line manually for Exercise 1

---

## STEP 2: IPLDataPreProcessor Lambda

Code :

```
import json
import boto3
import os
import csv
from io import StringIO

s3_client = boto3.client('s3')

def lambda_handler(event, context):
    bucket_name = os.environ.get('S3_BUCKET_NAME')
    input_file_key = 'ipl_teams.csv'
    output_file_key = 'processed_ipl_data.csv'

    print(f"Reading {input_file_key} from bucket {bucket_name}")

    try:
        response = s3_client.get_object(Bucket=bucket_name,
Key=input_file_key)
        csv_content = response['Body'].read().decode('utf-8')
```

```

# Use StringIO to treat the string as a file for csv.reader
csv_file = StringIO(csv_content)
reader = csv.reader(csv_file)
header = next(reader) # Read header row
data = list(reader) # Read remaining data rows

# Add 'can_win' to header
header.append('can_win')

processed_rows = []
for row in data:
    ipl_wins = int(row[1]) # ipl_wins is the second column (index 1)
    ipl_team_score = int(row[2]) # ipl_team_score is the third column
(index 2)

    # Simple preprocessing logic
    can_win = 1 if (ipl_wins >= 3 and ipl_team_score >= 2800) else 0
    new_row = row + [str(can_win)] # Append the new value
    processed_rows.append(new_row)

# Prepare content for writing back to S3
output_csv_file = StringIO()
writer = csv.writer(output_csv_file)
writer.writerow(header) # Write header
writer.writerows(processed_rows) # Write processed rows
processed_csv_content = output_csv_file.getvalue()

s3_client.put_object(Bucket=bucket_name, Key=output_file_key,
Body=processed_csv_content)

print(f"Successfully processed {input_file_key} and saved to
{output_file_key} in {bucket_name}")

return {
    'statusCode': 200,
    'body': json.dumps(f'Successfully processed data and stored in
s3://{bucket_name}/{output_file_key}')
}
except Exception as e:
    print(f"Error processing data: {e}")
    return {

```

```

        'statusCode': 500,
        'body': json.dumps(f'Error processing data: {str(e)}')
    }
}

```

Output :

The top screenshot shows the AWS Lambda console for the function 'iplambda'. It displays a success message: 'Successfully created the function iplambda. You can now change its code and configuration. To invoke your function with a test event, choose "Test".' The function overview shows the function name 'iplambda', its layers (0), and a description. The function ARN is 'arn:aws:lambda:eu-north-1:929690601290:func:iplambda' and the function URL is provided. The bottom screenshot shows the IAM console for the role 'iplambda-role-auqqcgv'. It displays a success message: 'Policy was successfully attached to role.' The permissions policies section shows two attached policies: 'AdministratorAccess' (AWS managed - job function) and 'AWSLambdaBasicExecutionRole' (Customer managed). The permissions boundary is not set.

#### ◆ Role & Permissions:

- IAM Role: **IPLDataPreProcessor-role-xxxx**
- Attach: **AmazonS3FullAccess** (Edit this in Exercise 3)
- Trust policy allows Lambda to assume this role.

#### ◆ Python Code (**lambda\_function.py**)

python

CopyEdit

```
import boto3
```

```
import csv
```

```
import io
```

```
def lambda_handler(event, context):
```

```
    s3 = boto3.client('s3')
```

```
    bucket = 'ipl-mlops-data'
```

```
    input_key = 'ipl_data.csv'
```

```
    output_key = 'processed_ipl_data.csv'
```

```
    # Download raw CSV
```

```
    raw_obj = s3.get_object(Bucket=bucket, Key=input_key)
```

```
    raw_data = raw_obj['Body'].read().decode('utf-8').splitlines()
```

```
    reader = csv.DictReader(raw_data)
```

```
    output = []
```

```
    for row in reader:
```

```
        team = row['team']
```

```
        wins = int(row['ipl_wins'])
```

```
        score = int(row['ipl_team_score'])
```

```
can_win = 1 if wins >= 3 and score >= 2800 else 0
```

```
output.append({'team': team, 'ipl_wins': wins, 'ipl_team_score': score, 'can_win':  
can_win})
```

```
# Write back processed CSV
```

```
output_io = io.StringIO()
```

```
writer = csv.DictWriter(output_io, fieldnames=['team', 'ipl_wins', 'ipl_team_score',  
'can_win'])
```

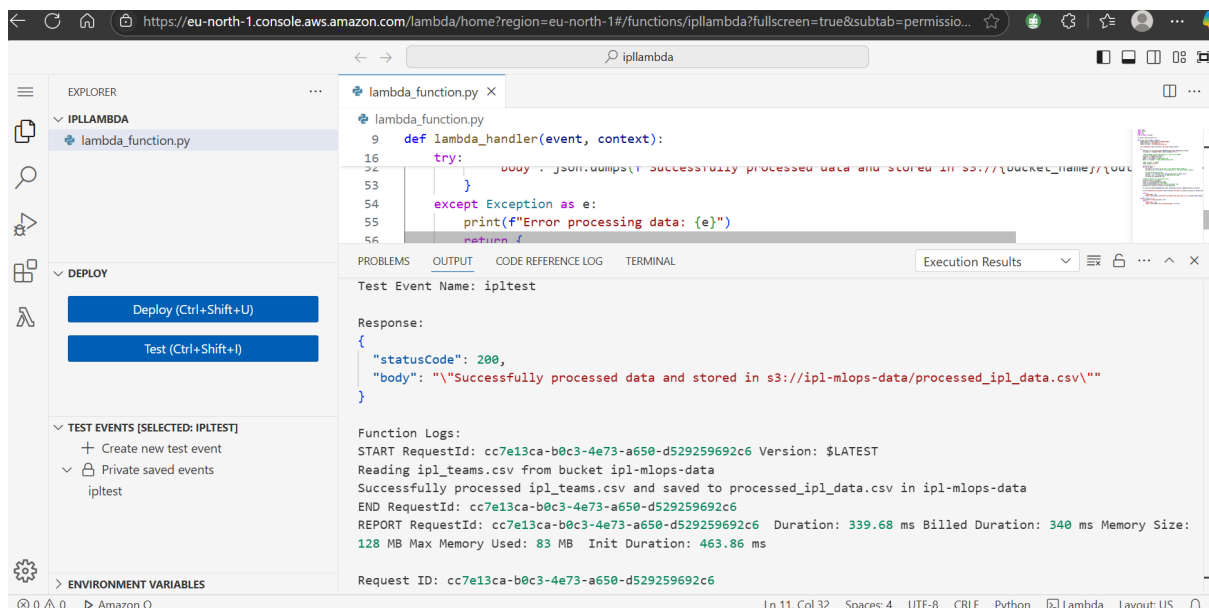
```
writer.writeheader()
```

```
for row in output:
```

```
writer.writerow(row)
```

```
s3.put_object(Bucket=bucket, Key=output_key, Body=output_io.getvalue())
```

```
return {'statusCode': 200, 'body': 'Preprocessing completed'}
```



---

### STEP 3: SageMaker Notebook – Train Your Model

Code :

```
# Line 1: Install scikit-learn if it's not already available in the environment
!pip install scikit-learn
!pip install pandas
```

```
import pandas as pd
import boto3
import io
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import joblib # For saving/loading models
```

```
# Line 1: Define your S3 bucket name
bucket_name = 'ipl-mlops-data' # REPLACE with your S3 bucket name
# Line 2: Define the key for the processed data file
processed_data_key = 'processed_ipl_data.csv'
# Line 3: Define the key for saving the trained model
model_key = 'ipl_winner_predictor.joblib'
```

```
# Line 4: Initialize S3 client
s3 = boto3.client('s3')
```

```
print(f"Downloading {processed_data_key} from {bucket_name}...")
# Line 5: Get the object from S3
obj = s3.get_object(Bucket=bucket_name, Key=processed_data_key)
# Line 6: Read the object body and decode it
body = obj["Body"].read().decode('utf-8')
# Line 7: Use io.StringIO to read the string content as a CSV file
df = pd.read_csv(io.StringIO(body))
```

```
print("Data loaded successfully:")
print(df.head())
print(df.info())
```

```
# Line 1: Define features (X) and target (y)
X = df[['ipl_wins', 'ipl_team_score']] # Features for prediction
y = df['can_win'] # Target variable
```

```
print("Features (X) head:")
print(X.head())
print("Target (y) head:")
```



```

print(y.head())

# Line 2: Split data into training and testing sets (80% train, 20% test)
# random_state for reproducibility
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print(f"Training data shape: {X_train.shape}")
print(f"Testing data shape: {X_test.shape}")

# Line 1: Initialize the Logistic Regression model
model = LogisticRegression(random_state=42)

print("Training model...")
# Line 2: Train the model using the training data
model.fit(X_train, y_train)

print("Model training complete.")

# Line 3: Evaluate the model on the test set
accuracy = model.score(X_test, y_test)
print(f"Model accuracy on test set: {accuracy:.2f}")

# Line 1: Save the trained model locally as a joblib file
local_model_path = '/tmp/ipl_winner_predictor.joblib' # SageMaker notebook instances have
/tmp for temporary storage
joblib.dump(model, local_model_path)

print(f"Model saved locally to {local_model_path}")

# Line 2: Upload the saved model to S3
s3.upload_file(local_model_path, bucket_name, model_key)

print(f"Model uploaded to s3://{bucket_name}/{model_key}")

```

Output:

```
from pandas.core.computation.check import NUMEXPR_INSTALLED
```

```
Downloading processed_ipl_data.csv from ipl-mlops-data...
```

```
Data loaded successfully:
```

```
   ipl_team ipl_wins ipl_team_score can_win
0      CSK         4           2850        1
1       MI         5           2900        1
2      RCB         0           2700        0
3      SRH         2           2750        0
4       KKR         3           2800        1
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 5 entries, 0 to 4
```

```
Data columns (total 4 columns):
```

```
#   Column      Non-Null Count  Dtype
---  -
0  ipl_team      5 non-null     object
1  ipl_wins       5 non-null     int64
2  ipl_team_score 5 non-null     int64
3  can_win        5 non-null     int64
```

```
dtypes: int64(3), object(1)
```

```
memory usage: 288.0+ bytes
```

```
None
```

```
Features (X) head:
```

```
   ipl_wins ipl_team_score
0         4           2850
1         5           2900
2         0           2700
3         2           2750
4         3           2800
```

```
Target (y) head:
```

```
0    1
1    1
2    0
3    0
4    1
```

```
Name: can_win, dtype: int64
```

```
Training data shape: (4, 2)
```

```
Testing data shape: (1, 2)
```

```
Training model...
```

```
Model training complete.
```

```
Model accuracy on test set: 1.00
```

The screenshot shows the Amazon SageMaker console interface. The top navigation bar includes the AWS logo, a search bar, and user information. The main content area is titled 'Notebook instances' and displays a table with columns: Name, Instance, Creation time, Status, and Actions. A single instance, 'ipldata01', is listed with status 'InService'. The left sidebar contains a menu with sections for 'Applications and IDEs' (Studio, Canvas, RStudio, Notebooks, Partner AI Apps), 'Admin configurations' (Domains, Role manager, Images, Lifecycle configurations), and 'SageMaker AI dashboard'.

Setup:

1. Start a new notebook instance (e.g., [ipl-sagemaker-notebook](#)).
2. Attach IAM Role with [AmazonS3FullAccess](#) and [AmazonSageMakerFullAccess](#).

Notebook Code Outline (Python):

```
python
```

```
CopyEdit
```

```
import pandas as pd
```

```
import boto3
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
import joblib
```

```
# Load processed CSV from S3
```

```
bucket = 'ipl-mlops-data'
```

```
key = 'processed_ipl_data.csv'
```

```
s3 = boto3.client('s3')
```

```
obj = s3.get_object(Bucket=bucket, Key=key)
```

```
df = pd.read_csv(obj['Body'])
```

```
# Model training
```

```
X = df[['ipl_wins', 'ipl_team_score']]
```

```
y = df['can_win']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

```
clf = RandomForestClassifier()
```

```
clf.fit(X_train, y_train)
```

```
# Save and upload model
```

```
joblib.dump(clf, 'ipl_model.joblib')
```

```
s3.upload_file('ipl_model.joblib', bucket, 'ipl_model.joblib')
```

---

#### STEP 4: IPLPredictionInference Lambda

- ◆ Permissions:

- Add [AmazonS3ReadOnlyAccess](#)

- ◆ Python Code:

```
python
```

```
CopyEdit
```

```
import boto3
```

```
import joblib
```

```
import json
```

```
import os
```

```
import pandas as pd
```

```
from io import BytesIO
```

```
def lambda_handler(event, context):
```

```
    try:
```

```
        body = json.loads(event['body']) # Exercise 2: Break it by renaming to `body_typo` for debugging
```

```
        ipl_wins = body['ipl_wins']
```

```
        ipl_team_score = body['ipl_team_score']
```

```
        # Load model from S3
```

```
        s3 = boto3.client('s3')
```

```
        bucket = 'ipl-mlops-data'
```

```
        key = 'ipl_model.joblib'
```

```
        model_obj = s3.get_object(Bucket=bucket, Key=key)
```

```
        model = joblib.load(BytesIO(model_obj['Body'].read()))
```

```

# Make prediction

input_df = pd.DataFrame([[ipl_wins, ipl_team_score]], columns=['ipl_wins',
'ipl_team_score'])

prediction = model.predict(input_df)[0]

return {

    'statusCode': 200,

    'body': json.dumps({'can_win': int(prediction)})

}

except Exception as e:

    return {

        'statusCode': 500,

        'body': f"Error: {str(e)}"

    }

```

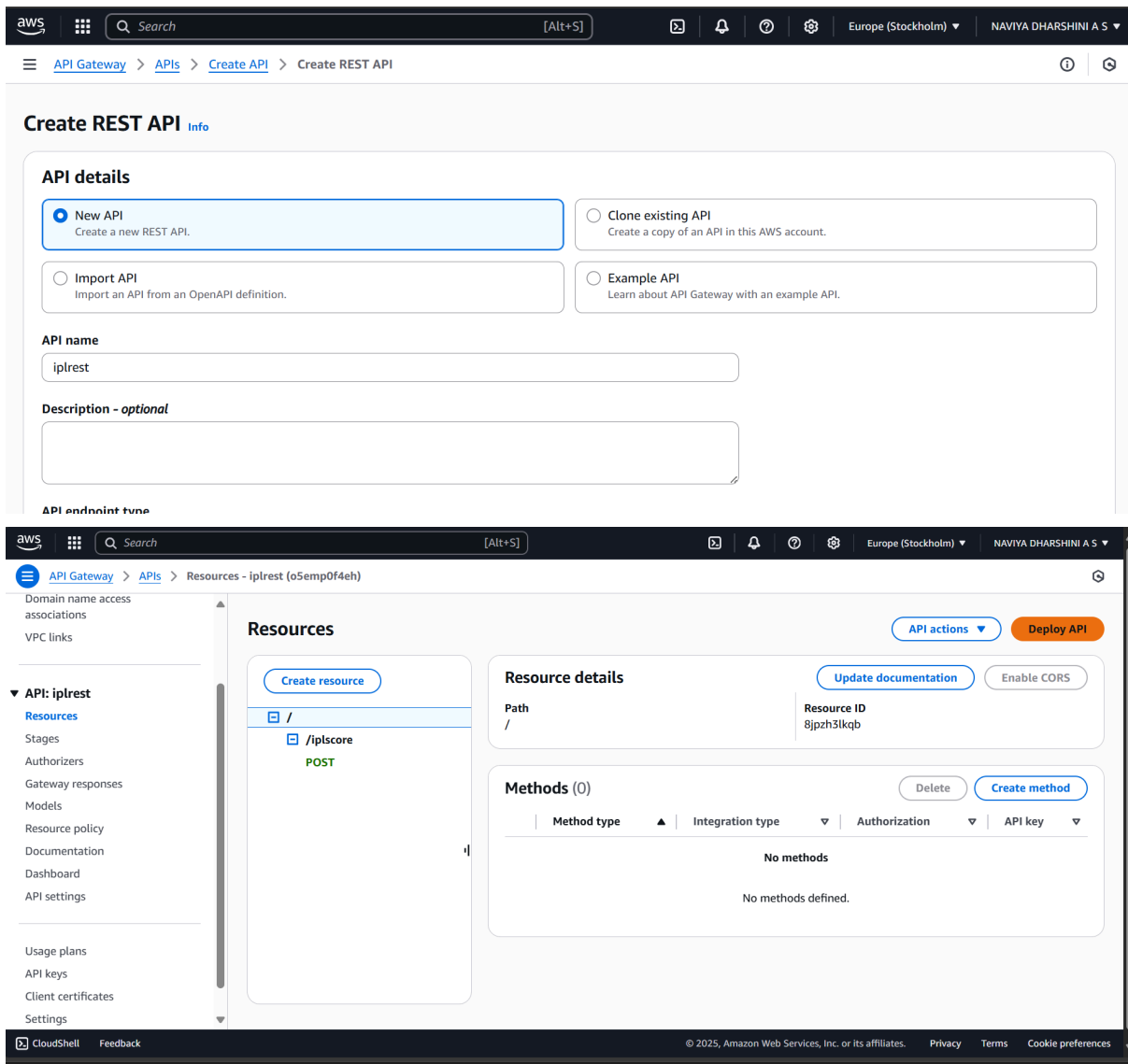
## STEP 4 .B) CREATE TRAINED MODEL - INFERENCE MODEL 👍

### 1. Same env

The screenshot shows the AWS IAM console interface for adding permissions to a function named 'inferenceModel'. The breadcrumb navigation is 'Lambda > Functions > inferenceModel > Add permissions'. The page contains several form fields for configuring the permission statement:

- Service:** A dropdown menu with 'S3' selected.
- Statement ID:** A text input field containing 'trainedlambdaModel'.
- Principal:** A text input field containing 's3.amazonaws.com'.
- Source account:** A text input field containing '929690601290'.
- Source ARN:** A text input field containing 'arn:aws:s3:::ipl-mlops-data'.
- Action:** A dropdown menu with 'lambda:AddPermission' selected.

## STEP 5: API Gateway Setup



The screenshot displays the AWS API Gateway console interface. The top navigation bar includes the AWS logo, a search bar, and user information for NAVIYA DHARSHINI A S in the Europe (Stockholm) region. The breadcrumb trail indicates the path: API Gateway > APIs > Create API > Create REST API.

**Create REST API** Info

**API details**

- ☒ **New API**  
Create a new REST API.
- ☐ **Clone existing API**  
Create a copy of an API in this AWS account.
- ☐ **Import API**  
Import an API from an OpenAPI definition.
- ☐ **Example API**  
Learn about API Gateway with an example API.

**API name**  
iplrest

**Description - optional**

**API endpoint type**

**Resources**

Domain name access associations  
VPC links

▼ **API: iplrest**

- Resources
- Stages
- Authorizers
- Gateway responses
- Models
- Resource policy
- Documentation
- Dashboard
- API settings

Usage plans  
API keys  
Client certificates  
Settings

**Resources**

Create resource

- /
- /iplscore  
POST

**Resource details**

Path: /

Resource ID: 8jpzh3lkqb

API actions: Update documentation, Enable CORS

**Methods (0)**

Delete Create method

Method type Integration type Authorization API key

No methods

No methods defined.

CloudShell Feedback

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Stages deploy ::

The screenshot shows the AWS API Gateway console for an API named 'iplrest (o5emp0f4eh)'. A green banner at the top indicates a successful deployment for the 'prod' stage. The left sidebar shows the navigation menu with 'API: iplrest' selected. The main content area displays the 'Stages' section with a list containing 'prod'. The 'Stage details' panel for 'prod' shows the following configuration:

Stage name	Rate	Web ACL
prod	10000	-

Cache cluster: Inactive (Burst: 5000)

Default method-level caching: Inactive

Invoke URL: <https://o5emp0f4eh.execute-api.eu-north-1.amazonaws.com/prod>

Active deployment: 1u5no2 on July 04, 2025, 14:42 (UTC+05:30)

1. Create a new REST API.
2. POST method → integrate with **IPLPredictionInference** Lambda.
3. Deploy to a stage (e.g., **prod**).
4. Test the endpoint via Postman or CLI:


json

CopyEdit

POST /prod/predict

```
{  
  
  "ipl_wins": 6,  
  
  "ipl_team_score": 3000  
}
```

---

 STEP 6: Debugging with CloudWatch (Exercise 2)



Create resource

/

/ipiscare

POST

Headers

Enter a header name and value separated by a colon (:). Use a new line for each header.

Header1:value1  
Header2:value2

Client certificate

No client certificates have been generated.

Request body

1

1:1 JSON Spaces: 4

Test

POST method test results

Request /ipiscare Latency ms 691 Status 200

Response body

{"errorMessage": "Unable to import module 'lambda\_function': No module named 'joblib'", "errorType": "Runtime.ImportModuleError", "requestId": "", "stackTrace": []}

Response headers

{
"Content-Type": "application/json",
"x-amzn-trace-id": "Root=1-6867a849-8557d9829f04f945793d5a7e;Parent=61c782510b642b5e;Sampled=0;Lineage=1:db35e525:0"
}

Logs

Execution log for request 8018af05-2615-4893-b228-4b486ff86e96  
Fri Jul 04 10:09:13 UTC 2025 : Starting execution for request: 8018af05-2615-4893-b228-4b486ff86e96  
Fri Jul 04 10:09:13 UTC 2025 : HTTP Method: POST, Resource Path: /ipiscare  
Fri Jul 04 10:09:13 UTC 2025 : Method request path: {}  
Fri Jul 04 10:09:13 UTC 2025 : Method request query string: {}  
Fri Jul 04 10:09:13 UTC 2025 : Method request headers: {}  
Fri Jul 04 10:09:13 UTC 2025 : Method request body before transformations:  
Fri Jul 04 10:09:13 UTC 2025 : Endpoint request URL: https://lambda.eu-north-1.amazonaws.com/2015-03-31/functions/arn:aws:lambda:eu-north-1:929608081290:function:inference-model/invocations  
Fri Jul 04 10:09:13 UTC 2025 : Endpoint request headers: {x-amzn-lambda-integration-tag=8018af05-2615-4893-b228-4b486ff86e96, Authorization=\*\*\*\*\*  
\*\*\*\*\*  
x-amzn-date=20250704T100913Z, x-amzn-apigateway-api-id=olmp8f4uh, x-amz-source-arn=arn:aws:execute-api:eu-north-1:929608081290:olmp8f4uh/test-  
invoke-stage/POST/ipiscare, Accept=application/json, User-Agent=AmazonAPIGateway, o5amp8f4uh, x-amz-security-  
Token=IQz0b3p22iux2vj8CEacmVtLW5vcnRULTE1SDBGA1EAvbXkxOZ28UN086ZlF2bERhYp2W09bVtR7fwa5c4CjQ0cD98Z6VXl0qppqaNtt32NP3wG225vYba8Zew2H5q4QgEAMuAD0P5N  
Dv2NDcxMzE2MSlG1Fts8oc1u8doybqtqspF0FR38+cpsgkpfYNOXK1uz6oX51z1g1pr7X5cdvR3cJ38AM1DAE4E5K4quiF7IndUCTe1Ac4F0y2049YbH8t115ka1D2qQe+KawXB2uJvtBL5a1QZ  
h7c0118p36c12lrdd07pviJ5tN992a3ECRL15W/v2 [TRUNCATED]  
Fri Jul 04 10:09:13 UTC 2025 : Endpoint request body after transformations:  
Fri Jul 04 10:09:13 UTC 2025 : Sending request to https://lambda.eu-north-1.amazonaws.com/2015-03-31/functions/arn:aws:lambda:eu-north-1:929608081290:function:inference-model/invocations  
Fri Jul 04 10:09:14 UTC 2025 : Received response. Status: 200, Integration latency: 686 ms  
Fri Jul 04 10:09:14 UTC 2025 : Endpoint response headers: (Date=Fri, 04 Jul 2025 10:09:14 GMT, Content-Type=application/json, Content-Length=164,  
Connection=keep-alive, x-amzn-requestid=db35b0a6-d612-4a6f-99f1-6a720718bcb, X-Amz-Function-Error=okHandled, x-amzn-remapped-content-length=0, X-Amz-  
Executed-Version=\$LATEST, X-Amzn-Trace-Id=Root=1-6867a849-8557d9829f04f945793d5a7e;Parent=61c782510b642b5e;Sampled=0;Lineage=1:db35e525:0)  
Fri Jul 04 10:09:13 UTC 2025 : Endpoint response body before transformations: {"errorMessage": "Unable to import module 'lambda\_function': No module named  
'joblib'", "errorType": "Runtime.ImportModuleError", "requestId": "", "stackTrace": []}  
Fri Jul 04 10:09:14 UTC 2025 : Method response body after transformations: {"errorMessage": "Unable to import module 'lambda\_function': No module named  
'joblib'", "errorType": "Runtime.ImportModuleError", "requestId": "", "stackTrace": []}  
Fri Jul 04 10:09:14 UTC 2025 : Method response headers: {X-Amzn-Trace-Id=Root=1-6867a849-8557d9829f04f945793d5a7e;Parent=61c782510b642b5e;Sampled=0;Lineage=1:db35e525:0, Content-Type=application/json}

Testing stage ..

- Logs auto-generated per Lambda run.
- Go to **CloudWatch** → **Logs** → **Log Group** for Lambda.
- Spot **KeyError: 'body\_typo'** etc.
- Fix code → re-deploy → re-test.

## STEP 7: IAM Access Denied Simulation (Exercise 3)

- Go to **IAM** → **Roles** → **IPLDataPreProcessor-role**.
  - Edit inline S3 policy → Remove **s3:PutObject** access to your S3 bucket.
  - Re-run Lambda → watch **AccessDenied** in CloudWatch logs.
  - Revert permissions to fix.
- 

#### STEP 8 (Optional): Clone for Football

- Duplicate all above setup using a new S3 bucket (**football-mlops-workshop**).
  - CSV: **team,games\_played,goals\_scored**
  - Change logic in processor/inference
  - New model, SageMaker notebook, Lambda, and endpoint.
- 

#### ENDGAME: Final Checklist

- ☒ **ipl\_data.csv** updated
  - ☒ PreProcessor Lambda triggered
  - ☒ SageMaker model retrained
  - ☒ Inference Lambda deployed
  - ☒ API tested successfully
  - ☒ IAM roles tested
  - ☒ CloudWatch debugging practiced
- 





#### GitHub README (Sample)

md

CopyEdit

# AWS MLOps - IPL Win Prediction

Built an end-to-end AWS pipeline using:

-  S3 (raw + processed data)
-  Lambda (preprocess + inference)
-  SageMaker (train model)
-  API Gateway (predictions)

## How to Use

1. Upload IPL data to S3.
2. Trigger preprocess Lambda.
3. Run SageMaker training.
4. Call API to get predictions!

> Tested with: CSK, MI, LSG, RCB