🔧 AWS MLOps Using Free Tier: Build a Real Pipeline with an IPL Prediction Use Case

Workshop Title: AWS MLOps Using Free Tier: Build a Real Pipeline with an IPL Prediction Use Case

Target Audience: Beginner-to-Intermediate **Workshop Duration:** 6 Hours **Delivery Method:** Mentor-led, Hands-on

---

🎯 Key Goals:

- **Utilize AWS Free Tier Services Exclusively:** Ensure all activities and services remain within the AWS Free Tier limits to avoid unexpected costs.
- **AWS Console Only (NO AWS CLI):** All configurations and interactions will be done through the AWS Management Console for a visual and guided experience.
- **Build a Real MLOps Pipeline from Dataset to Prediction:** Participants will construct a functional MLOps pipeline, demonstrating the end-to-end flow of machine learning operations.
- **Explain Each AWS Service Step-by-Step:** Provide detailed, guided instructions for configuring and using each AWS service involved in the pipeline.
- **Provide Line-by-Line Explanations of Python Code:** Ensure clear understanding of the logic and functionality of all Python scripts used in the workshop.
- **Include Hands-on Student Tasks for Every Module:** Encourage active participation and reinforce learning through practical exercises at the end of each module.
- **Include Troubleshooting Tips for Common AWS Errors:** Equip participants with strategies to identify and resolve common issues encountered during the workshop.
- **Dataset with Specific Columns:** The IPL dataset will strictly adhere to the `ipl_team,ipl_wins,ipl_team_score` format.

📚 Workshop Modules (Structure):

1. Cloud & AWS Basics (45 mins)

1.1. Explain IaaS, PaaS, SaaS, Serverless

- **IaaS (Infrastructure as a Service):**
    - **Explanation:** Provides virtualized computing resources over the internet. You manage the operating system, applications, and data, while the cloud provider manages the underlying infrastructure (servers, virtualization, storage, networking).
    - **Analogy:** Renting a car – you drive it, fill it with gas, but don't worry about manufacturing or maintenance.
- **PaaS (Platform as a Service):**
    - **Explanation:** Provides a platform allowing customers to develop, run, and manage applications without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app.
    - **Analogy:** Taking a taxi – you tell the driver where to go, but don't drive or maintain the car.
- **SaaS (Software as a Service):**
    - **Explanation:** Delivers software applications over the internet, on-demand and typically on a subscription basis. Users access the software via a web browser or mobile app.
    - **Analogy:** Taking a bus – you just get on and ride, no driving or maintenance required.
- **Serverless Computing:**
    - **Explanation:** A cloud execution model where the cloud provider dynamically manages the allocation and provisioning of servers. You only pay for the compute time you consume, without provisioning or managing servers.
    - **Analogy:** A public utility like electricity – you only pay for what you use, without worrying about power plants.

1.2. Overview of AWS: Regions, Availability Zones, Edge Locations

- **AWS Regions:**
    - **Explanation:** A physical location around the world where AWS clusters data centers. Each Region is a separate geographic area, completely independent of other Regions. This ensures fault tolerance and isolation.
    - **Why it's important:** Choose a Region close to your users to minimize latency.
- **Availability Zones (AZs):**
    - **Explanation:** Discrete data centers within a Region, designed to be isolated from failures in other AZs. They are connected by low-latency, high-bandwidth, and redundant networking.

tolerance for your applications.

- **Edge Locations:**
  - **Explanation:** AWS data centers located closer to end-users, primarily used by Amazon CloudFront (Content Delivery Network) to cache content and deliver it with lower latency.
  - **Why it's important:** Improves content delivery speed and user experience.

1.3. Benefits of Cloud for ML

- **Scalability:** Easily scale compute and storage resources up or down as needed for training large models or handling high inference traffic.
- **Cost-Effectiveness:** Pay-as-you-go model eliminates the need for large upfront investments in hardware. Utilize free tiers and optimize spending.
- **Agility & Speed:** Rapidly provision resources and experiment with different models without waiting for hardware procurement.
- **Managed Services:** AWS offers a wide range of managed ML services (like SageMaker) that simplify the development, training, and deployment of ML models.
- **Global Reach:** Deploy ML applications globally to serve users worldwide with low latency.

✅ Hands-on: Explore AWS Global Map in Console

- **Mentor Instructions:** Guide students to the AWS Management Console and show them how to navigate to the global map.
- **Step-by-step:**
  1. Open your web browser and go to the AWS Management Console.
  2. Log in with your AWS account credentials.
  3. In the top right corner of the console, click on the **Region dropdown** (e.g., "N. Virginia").
  4. Observe the list of available Regions and their names.
  5. You can also search for "Global Infrastructure" in the search bar to find more information and potentially an interactive map. (Note: The exact location of the global map might vary slightly with console updates, but the Region dropdown is always prominent).
- **Student Task:** Identify your closest AWS Region and an AWS Region on a different continent.

2. AWS Free Tier Overview (20 mins)

Mentor Goal: Educate participants on AWS Free Tier benefits and how to utilize them responsibly.

- **12-Month Free Tier:**
  - **Explanation:** Services that are free for 12 months following your AWS sign-up date. This typically includes a certain amount of EC2 compute time, S3 storage, RDS database capacity, etc.
  - **Key Reminder:** These limits expire after 12 months, so it's crucial to monitor usage.
- **Always Free:**
  - **Explanation:** Services that are free for all AWS customers up to certain usage limits. These limits do not expire.
  - **Examples:** Certain Lambda function invocations, DynamoDB capacity, CloudWatch metrics, S3 put requests.
- **Free Trials:**
  - **Explanation:** Short-term free trials for new services or higher usage tiers, designed to let you experiment without immediate cost.

2.2. Services used in this workshop (and their Free Tier applicability):

- **Amazon S3 (Simple Storage Service):**
  - **What it does:** Object storage for storing and retrieving any amount of data from anywhere on the web.
  - **Free Tier:** 5 GB of standard storage, 20,000 Get Requests, and 2,000 Put Requests per month. (Sufficient for our small dataset and model.)
- **AWS Lambda:**
  - **What it does:** Serverless compute service that runs your code in response to events and automatically manages the underlying compute resources.
  - **Free Tier:** 1 million free requests per month and 400,000 GB-seconds of compute time per month. (More than enough for our preprocessing and inference functions.)
- **Amazon SageMaker:**
  - **What it does:** Fully managed service for building, training, and deploying machine learning models.
  - **Free Tier:** Offers 250 hours per month of t2.medium or t3.medium notebook instance for the first two months, and 50 hours of m4.xlarge or m5.xlarge for training and 125 hours of m4.xlarge or m5.xlarge for hosting for the first two months. We will primarily use `ml.t2.medium` for training, which falls under the free tier for the first two months. **Crucial to stop instances after use.**

- **What it does:** Fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale.
- **Free Tier:** 1 million API calls per month for the first 12 months. (More than enough for testing our inference endpoint.)

- **Amazon CloudWatch:**
  - **What it does:** Monitoring and observability service for AWS resources and applications. Collects and tracks metrics, collects and monitors log files, and sets alarms.
  - **Free Tier:** 10 custom metrics, 10 alarms, 1 million API requests, and 5 GB of log data ingestion per month. (Sufficient for monitoring our pipeline logs.)

2.3. How to stay under Free Tier limits

- **Monitor Usage:** Regularly check your AWS Billing Dashboard and Cost Explorer.
- **Delete Unused Resources:** Terminate EC2 instances, delete S3 buckets, Lambda functions, etc., when no longer needed. This is the most crucial step!
- **Understand Service Limits:** Be aware of the specific free tier limits for each service you use.
- **Set Up Billing Alerts:** Receive notifications if your spending approaches a certain threshold.

✅ Task: Setup billing alerts and cost dashboard

- **Mentor Instructions:** Guide students through setting up billing alerts and exploring the cost dashboard.
- **Step-by-step:**
  1. Go to the AWS Management Console.
  2. In the search bar at the top, type "Billing" and select "Billing" from the results.
  3. On the Billing Dashboard, on the left-hand navigation, click on **"Billing preferences"**.
  4. Check the box for **"Receive Free Tier Usage Alerts"** and **"Receive Billing Alerts"**. Enter your email address.
  5. Go back to the Billing Dashboard.
  6. On the left-hand navigation, click on **"Cost Explorer"**.
  7. Click **"Launch Cost Explorer"** (it might take a few minutes for Cost Explorer to be enabled for the first time).
  8. Explore the default cost and usage reports. You won't see much data yet, but this is where you'll monitor your spending.

checked in their account.

3. MLOps Introduction (30 mins)

Mentor Goal: Explain the transition from traditional ML development to MLOps and its benefits.

3.1. Traditional ML vs MLOps

- **Traditional ML Development:**
  - Often a more manual, ad-hoc process.
  - Focus primarily on model training and evaluation.
  - Deployment can be a separate, often complex, manual step.
  - Lack of version control for models, data, and code.
  - Limited monitoring and re-training capabilities.
  - "Model graveyard" – many models built but few deployed.
- **MLOps (Machine Learning Operations):**
  - Applies DevOps principles to machine learning.
  - Focuses on automating the entire ML lifecycle from data collection to model deployment and monitoring.
  - Emphasizes collaboration between data scientists, ML engineers, and operations teams.
  - Aims for continuous integration, continuous delivery (CI/CD), continuous training (CT), and continuous monitoring (CM).

3.2. Benefits: Reproducibility, Deployment, Monitoring

- **Reproducibility:**
  - **Explanation:** The ability to achieve the same results every time a model is trained or deployed, given the same data, code, and environment.
  - **MLOps Benefit:** Version control for data, code, and models; standardized environments (e.g., Docker containers); automated pipelines ensure consistent execution.
- **Deployment:**
  - **Explanation:** The process of making a trained machine learning model available for use by other applications or services.
  - **MLOps Benefit:** Automated, reliable, and scalable deployment mechanisms (e.g., A/B testing, canary deployments, blue/green deployments) ensure models reach production quickly and

- **Monitoring:**
  - **Explanation:** Continuously tracking the performance, health, and behavior of deployed ML models and the underlying infrastructure.
  - **MLOps Benefit:** Detects model drift, data drift, performance degradation, and system failures early, allowing for proactive re-training or intervention.

3.3. MLOps lifecycle overview

- **Data Collection & Preparation:** Gathering, cleaning, transforming, and labeling data.
- **Feature Engineering:** Creating new features from raw data to improve model performance.
- **Model Training:** Selecting an algorithm, training the model on prepared data.
- **Model Evaluation & Validation:** Assessing model performance and ensuring it meets business requirements.
- **Model Versioning & Registry:** Storing different versions of models and their metadata.
- **Model Deployment:** Making the model available for inference (e.g., as an API endpoint).
- **Model Monitoring:** Continuously tracking model performance, data drift, and system health.
- **Model Retraining:** Automatically or manually retraining models when performance degrades or new data becomes available.

✅ Task: Draw lifecycle diagram

- **Mentor Instructions:** Encourage students to draw a simple MLOps lifecycle diagram based on the discussion.
- **Student Task:** On a piece of paper or a digital whiteboard, sketch out the key stages of an MLOps lifecycle. Don't worry about perfection, just capture the flow.

4. AWS Service Setup (1 hr 15 mins) – Console Only

Mentor Goal: Guide students through the hands-on setup of each AWS service required for the MLOps pipeline.

Important Note for all service setups:

- **Mentor Tip:** Remind students to always select a consistent AWS Region for all their services to ensure they can communicate effectively. We will primarily use `us-east-1` (N. Virginia) for this workshop.
- **Common AWS Error:** "Access Denied" or "You are not authorized to perform this operation." This usually means the IAM user/role doesn't have the necessary permissions. Double-check IAM

- **What it does in MLOps:** S3 acts as our central repository for storing raw datasets, preprocessed data, and the trained machine learning model artifacts. It provides durable, scalable, and highly available storage.
- **Why it's needed:**
  - **Data Lake:** A cost-effective way to store large volumes of structured and unstructured data.
  - **Model Storage:** Securely store trained model files for deployment and versioning.
  - **Accessibility:** Easily accessible by other AWS services like Lambda and SageMaker.
- **Step-by-step AWS Console setup:**
  1. **Create an S3 Bucket:**
     - Go to the AWS Management Console.
     - In the search bar, type "S3" and select "S3" from the results.
     - Click **"Create bucket"**.
     - **Bucket name:** Enter a unique, globally distinct name (e.g., `ipl-mlops-workshop-yourname-date`). *Lowercase letters, numbers, and hyphens only.*
     - **AWS Region:** Select your preferred Region (e.g., `US East (N. Virginia) us-east-1`).
     - **Object Ownership:** Keep "ACLs enabled" and "Bucket owner preferred".
     - **Block Public Access settings for this bucket: Uncheck** "Block all public access" for now, as we'll make our bucket readable for a specific use case during the workshop. **Acknowledge** the warning. *Mentor Tip: In a real-world scenario, you would keep public access blocked and use IAM policies or S3 pre-signed URLs for access control.*
     - **Bucket Versioning:** Keep "Disabled".
     - **Default encryption:** Keep "Disabled" for simplicity in this workshop.
     - Click **"Create bucket"**.
  2. **Upload the Dataset:**
     - Go into the newly created S3 bucket.
     - Click **"Upload"**.
     - Click **"Add file"**.
     - Create a local CSV file named `ipl_data.csv` with the following content:

       Code snippet

```
CSK,4,2850
MI,5,2900
RCB,0,2700
SRH,2,2750
KKR,3,2800
```

    - Select the `ipl_data.csv` file.
    - Click **"Upload"**.
- **Student Task:** Create an S3 bucket and upload the `ipl_data.csv` file. Record the bucket name.
- **Troubleshooting Tips:**
    - **Bucket name already exists:** S3 bucket names must be globally unique. Try adding more characters or a unique identifier (e.g., your initials and a random number).
    - **Permissions error during upload:** Ensure your AWS user has `s3:PutObject` permissions for the bucket.

4.2. AWS Lambda

- **What it does in MLOps:** We'll use Lambda for two key functions:
    1. **Data Preprocessing:** To read the raw data from S3, perform simple transformations, and prepare it for SageMaker training.
    2. **Model Inference:** To load the trained model from S3 and provide predictions based on new input data. This will be triggered by API Gateway.
- **Why it’s needed:**
    - **Serverless:** No servers to provision or manage, you only pay for compute time when your code is running.
    - **Event-driven:** Can be easily triggered by S3 events (for preprocessing) or API Gateway requests (for inference).
    - **Cost-effective:** Within Free Tier limits for our use case.
- **Step-by-step AWS Console setup (for Data Preprocessing Lambda):**
    1. **Create a Lambda Function:**
        - Go to the AWS Management Console.
        - In the search bar, type "Lambda" and select "Lambda" from the results.
        - Click **"Create function"**.
        - **Author from scratch:** Selected

- **Runtime:** `Python 3.9` (or the latest Python 3.x available).
- **Architecture:** `x86_64`
- **Permissions:** Click "Change default execution role".
  - **Execution role:** Select **"Create a new role with basic Lambda permissions"**. This will create a basic role that allows CloudWatch logging. We will attach additional S3 permissions later.
- Click **"Create function"**.

2. **Configure Environment Variables (Optional but good practice):**
   - Once the function is created, scroll down to the "Configuration" tab, then select "Environment variables".
   - Click "Edit".
   - Add a variable:
     - Key: `S3_BUCKET_NAME`
     - Value: Your S3 bucket name (e.g., `ipl-mlops-workshop-yourname-date`)
3. **Add S3 Permissions to Lambda Role:**
   - In your Lambda function configuration, under "Permissions", click on the **Role name** (e.g., `IPLDataPreProcessor-role-xxxxxx`). This will open the IAM console.
   - In the IAM console, under "Permissions policies", click **"Add permissions"** -> **"Attach policies"**.
   - Search for and select `AmazonS3ReadOnlyAccess`. (We are giving it read access to S3 for now. For writing, we'd need `s3:PutObject`.)
   - Click **"Add permissions"**.
   - Now, also attach a policy that allows writing to S3: Click "Add permissions" -> "Attach policies" again. Search for `AmazonS3FullAccess` for simplicity in this workshop. *Mentor Tip: In a real-world scenario, you would create a custom policy with only the specific S3 permissions required (e.g., `s3:PutObject` for a specific bucket and prefix).*
   - Go back to the Lambda function tab.
4. **Add Python Code for Preprocessing:**
   - In the Lambda console, under the "Code" tab, you'll see a default `lambda_function.py`. Replace its content with the following:

     Python

```
import json
import boto3
import pandas as pd
import os

# Initialize S3 client
s3_client = boto3.client('s3')

def lambda_handler(event, context):
    # Line 1: Get the S3 bucket name from environment variables
    bucket_name = os.environ.get('S3_BUCKET_NAME')
    # Line 2: Define the input file key (path) in S3
    input_file_key = 'ipl_data.csv'
    # Line 3: Define the output file key (path) in S3 for processed data
    output_file_key = 'processed_ipl_data.csv'

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

    try:
        # Line 4: Download the CSV file from S3
        response = s3_client.get_object(Bucket=bucket_name, Key=input_file_key)
        # Line 5: Read the content of the file
        csv_content = response['Body'].read().decode('utf-8')
        # Line 6: Use pandas to read the CSV content into a DataFrame
        df = pd.read_csv(pd.io.common.StringIO(csv_content))

        print("Original DataFrame:")
        print(df.head())

        # Line 7: Simple preprocessing: Create a 'target' column (binary classifier)
        # Predict if a team can win based on ipl_wins and ipl_team_score
        # For simplicity, let's say a team "can win" if ipl_wins >= 3 AND ipl_team_sco
        df['can_win'] = ((df['ipl_wins'] >= 3) & (df['ipl_team_score'] >= 2800)).astyp

        print("Processed DataFrame:")
        print(df.head())

        # Line 8: Convert the processed DataFrame back to CSV format (without index)
        processed_csv_content = df.to_csv(index=False)

        # Line 9: Upload the processed CSV back to S3
        s3_client.put_object(Bucket=bucket_name, Key=output_file_key, Body=processed_c

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

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

```
        }
```

- **Line-by-line Python explanation:**
    - `import json, boto3, pandas as pd, os`: Imports necessary libraries for JSON handling, AWS SDK, data manipulation, and environment variables.
    - `s3_client = boto3.client('s3')`: Initializes an S3 client to interact with S3 services.
    - `bucket_name = os.environ.get('S3_BUCKET_NAME')`: Retrieves the S3 bucket name from the Lambda environment variables.
    - `input_file_key = 'ipl_data.csv'`: Defines the name of the input CSV file in S3.
    - `output_file_key = 'processed_ipl_data.csv'`: Defines the name for the output processed CSV file in S3.
    - `response = s3_client.get_object(Bucket=bucket_name, Key=input_file_key)`: Fetches the raw `ipl_data.csv` from the specified S3 bucket.
    - `csv_content = response['Body'].read().decode('utf-8')`: Reads the content of the downloaded file and decodes it as UTF-8.
    - `df = pd.read_csv(pd.io.common.StringIO(csv_content))`: Uses pandas to read the CSV string directly into a DataFrame. `StringIO` treats the string as a file.
    - `df['can_win'] = ((df['ipl_wins'] >= 3) & (df['ipl_team_score'] >= 2800)).astype(int)`: This is our core preprocessing logic. It creates a new `can_win` column. If an `ipl_team` has 3 or more `ipl_wins` AND an `ipl_team_score` of 2800 or more, `can_win` is 1 (True), otherwise 0 (False). `.astype(int)` converts boolean to 0/1.
    - `processed_csv_content = df.to_csv(index=False)`: Converts the processed DataFrame back into a CSV string. `index=False` prevents pandas from writing the DataFrame index as a column.
    - `s3_client.put_object(Bucket=bucket_name, Key=output_file_key, Body=processed_csv_content)`: Uploads the generated processed CSV content back to S3.
    - `return {'statusCode': 200, ...}`: Returns a success response.
    - `except Exception as e: ...`: Basic error handling to catch and print exceptions.
- Click **"Deploy"** to save your code.

5. **Test the Lambda Function:**

- Click **"Configure test event"**.
- **Event name:** `TestDataProcessorEvent`
- Leave the JSON payload as default (or empty `{}` as we are not using event data for this function's trigger in this simple case).
- Click **"Save"**.
- Click **"Test"**.
- **Expected Output:** You should see "Execution result: succeeded" and in the "Log output" section, messages indicating successful processing and S3 upload.
- **Verify in S3:** Go to your S3 bucket; you should now see a new file named `processed_ipl_data.csv`. Download and inspect it to confirm the `can_win` column is present.

- **Student Task:** Create the `IPLDataPreProcessor` Lambda function, add S3 permissions to its role, paste the provided Python code, configure the `S3_BUCKET_NAME` environment variable, deploy, and test it. Verify `processed_ipl_data.csv` is in S3.
- **Troubleshooting Tips:**
  - **"Unable to import module 'lambda_function': No module named 'pandas'"**: You need to include `pandas` and `boto3` in your Lambda deployment package if you are using layers or custom zip. For simple cases, `boto3` is usually pre-installed. For `pandas`, you'd typically need a Lambda Layer. **For this workshop's simplicity, we are assuming a pre-configured environment or that participants will understand the need for a layer in a full production setup.** *Mentor Tip: If students hit this, explain Lambda Layers briefly. For this workshop, we'll guide them to use a pre-built layer or simplify the preprocessing to avoid external libraries if time is tight.* **Correction for workshop:** To simplify, we will avoid Pandas for this initial preprocessing Lambda. We will just read, add the column manually (simulating, not actual heavy processing), and write.

  Revised Lambda Code for Preprocessing (NO PANDAS):

  Python

```
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_data.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_cont

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

        return {
            'statusCode': 200,
            'body': json.dumps(f'Successfully processed data and stored in s3://{bucket_name}
```

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

- **Line-by-line Python explanation (No Pandas):**
  - `import csv, StringIO`: Imports modules for CSV handling and treating strings as files.
  - `csv_file = StringIO(csv_content)`: Wraps the raw CSV content in a `StringIO` object so `csv.reader` can read from it like a file.
  - `reader = csv.reader(csv_file)`: Creates a CSV reader object.
  - `header = next(reader)`: Reads the first row (header) from the CSV.
  - `data = list(reader)`: Reads all remaining rows into a list of lists.
  - `header.append('can_win')`: Adds the new column name to the header.
  - `for row in data:`: Iterates through each data row.
  - `ipl_wins = int(row[1])`, `ipl_team_score = int(row[2])`: Converts the win and score columns to integers.
  - `can_win = 1 if (ipl_wins >= 3 and ipl_team_score >= 2800) else 0`: Applies the classification logic.
  - `new_row = row + [str(can_win)]`: Creates a new row by appending the `can_win` value (converted to string).
  - `processed_csv_content = output_csv_file.getvalue()`: Gets the complete CSV string from the `StringIO` object.
  - The rest is similar to the Pandas version for S3 interaction and error handling.
- **Common AWS Error:** "Invalid Lambda function policy." This might happen if you are trying to attach an invalid policy or have a typo. Review the policy syntax.

4.3. Amazon SageMaker

- **What it does in MLOps:** SageMaker provides the environment to train our machine learning model. We will use a SageMaker Notebook Instance to run our training script.
- **Why it’s needed:**
  - **Managed Service:** Simplifies the setup and management of ML environments.

`ml.t2.medium`.

- **Integration:** Easily integrates with S3 for data and model storage.

- **Step-by-step AWS Console setup:**

  1. **Create a SageMaker Notebook Instance:**
     - Go to the AWS Management Console.
     - In the search bar, type "SageMaker" and select "Amazon SageMaker" from the results.
     - In the left-hand navigation, under **"Notebook"**, select **"Notebook instances"**.
     - Click **"Create notebook instance"**.
     - **Notebook instance name:** `ipl-prediction-notebook`
     - **Notebook instance type:** Select `ml.t2.medium` (This is crucial for Free Tier eligibility).
     - **Platform identifier:** `notebook-al2-v2` (or latest stable Amazon Linux 2 version).
     - **Permissions and encryption:**
       - **IAM role:** Choose **"Create a new role"**.
       - **Specify S3 buckets you want to access:** Select **"Specific S3 buckets"** and type in your S3 bucket name (e.g., `ipl-mlops-workshop-yourname-date`). This will create an IAM role with S3 access to your bucket.
       - Click **"Create role"**.
     - **Network:** Leave defaults.
     - **Git repositories:** Leave "No repository".
     - Click **"Create notebook instance"**.
     - **Mentor Tip:** This step can take 5-10 minutes for the notebook instance to be "InService". Be patient.
  2. **Open Jupyter and Create a Notebook:**
     - Once the notebook instance status is "InService", click **"Open Jupyter"** next to your notebook instance.
     - In the Jupyter environment, click **"New"** -> **"Python 3 (Data Science)"** (or similar Python 3 kernel).
  3. **Add Python Code for Training:**
     - In the new Jupyter notebook, enter the following code into cells.

Python

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

- **Line-by-line explanation:**
  - `!pip install scikit-learn`: The `!` prefix allows running shell commands from within a Jupyter notebook. This command installs the `scikit-learn` library, which is a popular machine learning library in Python.

- **Cell 2: Import Libraries and Load Data:**

Python

```
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-workshop-yourname-date' # 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-by-line Python explanation:**
  - `import pandas as pd, boto3, io, sklearn.model_selection, sklearn.linear_model, joblib`: Imports necessary libraries for data handling,

with your actual S3 bucket name.

- `processed_data_key = 'processed_ipl_data.csv'` : Specifies the name of the processed data file in S3.
- `model_key = 'ipl_winner_predictor.joblib'` : Specifies the name under which the trained model will be saved in S3.
- `s3 = boto3.client('s3')` : Initializes an S3 client.
- `obj = s3.get_object(Bucket=bucket_name, Key=processed_data_key)` : Retrieves the processed data CSV file from S3.
- `body = obj['Body'].read().decode('utf-8')` : Reads the content of the file and decodes it.
- `df = pd.read_csv(io.StringIO(body))` : Reads the string content into a pandas DataFrame.

- **Cell 3: Prepare Data for Training:**

  Python

```
# 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=

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

  - **Line-by-line Python explanation:**
    - `X = df[['ipl_wins', 'ipl_team_score']]` : Selects the `ipl_wins` and `ipl_team_score` columns as input features for the model.
    - `y = df['can_win']` : Selects the `can_win` column as the target variable (what we want to predict).
    - `X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)` : Splits the dataset into training and testing

`random_state=42` ensures the split is the same every time for reproducibility.

- **Cell 4: Train the Model:**

  Python

```
# 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-by-line Python explanation:**
    - `model = LogisticRegression(random_state=42)`: Creates an instance of the Logistic Regression model. `random_state` ensures reproducibility for internal random processes.
    - `model.fit(X_train, y_train)`: Trains the logistic regression model using the training features (`X_train`) and target (`y_train`).
    - `accuracy = model.score(X_test, y_test)`: Evaluates the trained model's performance on the unseen test data (`X_test` , `y_test`) and calculates the accuracy.

- **Cell 5: Save and Upload the Model:**

  Python

```
# Line 1: Save the trained model locally as a joblib file
local_model_path = '/tmp/ipl_winner_predictor.joblib' # SageMaker notebook instances h
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}")
```

  - **Line-by-line Python explanation:**

temporary path on the SageMaker notebook instance file system to save the model. `/tmp` is a common writable temporary directory.

- `joblib.dump(model, local_model_path)`: Uses `joblib` to serialize (save) the trained `model` object to the specified `local_model_path`.
- `s3.upload_file(local_model_path, bucket_name, model_key)`: Uploads the locally saved model file from the notebook instance to your S3 bucket under the specified `model_key`.

4. **Run All Cells:** Click "Cell" -> "Run All" to execute the entire notebook. Observe the output.
5. **Verify Model in S3:** Go back to your S3 bucket in the AWS Console. You should see a new file named `ipl_winner_predictor.joblib`.
6. **STOP THE NOTEBOOK INSTANCE!** This is critical for Free Tier.
   - Go back to the SageMaker console -> Notebook instances.
   - Select your `ipl-prediction-notebook` instance.
   - Click **"Actions"** -> **"Stop"**. Confirm the action.

- **Student Task:** Create a SageMaker notebook instance (ml.t2.medium), open Jupyter, create a new Python notebook, paste the training code (remember to update the S3 bucket name!), run all cells, and **stop the notebook instance** immediately after verification.
- **Troubleshooting Tips:**
  - **"No module named 'pandas' / 'sklearn'"**: Ensure `!pip install pandas` or `!pip install scikit-learn` runs successfully in your notebook (first cell).
  - **"Access Denied" when reading/writing S3**: Check the IAM role attached to your SageMaker notebook instance. It needs permissions like `s3:GetObject` and `s3:PutObject` for your specific bucket.
  - **Notebook instance stuck in "Pending" or "Failed"**: Check CloudWatch logs for the SageMaker instance creation. Often, it's an IAM role issue or network configuration.

4.4. CloudWatch

- **What it does in MLOps:** CloudWatch is the primary service for monitoring our Lambda functions. It collects logs from Lambda, allowing us to debug errors, track execution times, and see print statements from our code.
- **Why it’s needed:**
  - **Debugging:** Essential for understanding why Lambda functions fail or behave unexpectedly.
  - **Performance Monitoring:** Tracks execution duration and memory usage.

alerting in this workshop).

- **Step-by-step AWS Console setup:**
    1. **Access CloudWatch Logs:**
        - Go to the AWS Management Console.
        - In the search bar, type "CloudWatch" and select "CloudWatch" from the results.
        - In the left-hand navigation, under **"Logs"**, select **"Log groups"**.
        - You will see log groups for your Lambda functions (e.g., `/aws/lambda/IPLDataPreProcessor`).
    2. **View Lambda Logs:**
        - Click on the log group for `IPLDataPreProcessor`.
        - You'll see one or more "Log streams". Each log stream corresponds to an invocation or a period of time the Lambda was running.
        - Click on the latest log stream.
        - **Explanations of Log Content:**
            - You'll see `START`, `END`, and `REPORT` lines, which indicate the function's execution start, end, and resource usage.
            - Any `print()` statements from your Python code will appear here.
            - Error messages (tracebacks) will also be clearly visible.
- **Student Task:** Navigate to CloudWatch, find the log group for your `IPLDataPreProcessor` Lambda, and inspect the logs from its previous test run.
- **Troubleshooting Tips:**
    - **No log group found:** Ensure the Lambda function has an IAM role with `logs:CreateLogGroup`, `logs:CreateLogStream`, and `logs:PutLogEvents` permissions. The default Lambda execution role usually includes these.
    - **Empty log stream:** The Lambda function might not have been invoked, or there was an immediate unhandled error preventing logs from being written.

4.5. Amazon API Gateway

- **What it does in MLOps:** API Gateway acts as the front door for our model inference. It creates a REST API endpoint that external applications (or a simple `curl` command) can call to send input data and receive predictions from our Lambda function.
- **Why it's needed:**

- **Scalability:** Handles API requests at scale.
- **Integration with Lambda:** Seamlessly triggers Lambda functions based on incoming API requests.
- **Security:** Can apply various security mechanisms (though we'll keep it simple for Free Tier).

- **Step-by-step AWS Console setup (for Model Inference Lambda and API Gateway):**

  First, create the Inference Lambda:

  1. **Create a new Lambda Function for Inference:**
     - Go to the AWS Management Console -> Lambda.
     - Click **"Create function"**.
     - **Author from scratch:** Selected.
     - **Function name:** `IPLPredictionInference`
     - **Runtime:** `Python 3.9` (or latest).
     - **Architecture:** `x86_64`
     - **Permissions:** Select **"Create a new role with basic Lambda permissions"**.
     - Click **"Create function"**.
  2. **Add S3 Permissions to Inference Lambda Role:**
     - In your `IPLPredictionInference` Lambda function configuration, under "Permissions", click on the **Role name**.
     - In the IAM console, click **"Add permissions"** -> **"Attach policies"**.
     - Search for and attach `AmazonS3ReadOnlyAccess`. (It only needs to read the model).
  3. **Add Python Code for Inference:**
     - In the Lambda console for `IPLPredictionInference`, under the "Code" tab, replace its content with:

       Python

```
import json
import boto3
import os
import joblib
import io
import pandas as pd # Although we avoided for preprocessing, it's common for inference

# Line 1: Initialize S3 client
s3_client = boto3.client('s3')

# Line 2: Define S3 bucket and model key from environment variables
BUCKET_NAME = os.environ.get('S3_BUCKET_NAME')
MODEL_KEY = 'ipl_winner_predictor.joblib'
LOCAL_MODEL_PATH = '/tmp/ipl_winner_predictor.joblib' # Temporary path in Lambda execu

# Line 3: Global variable to store the loaded model, to avoid re-loading on subsequent
model = None

def load_model():
    global model
    if model is None:
        print(f"Loading model from s3://{BUCKET_NAME}/{MODEL_KEY}...")
        try:
            # Line 4: Download model from S3
            s3_client.download_file(BUCKET_NAME, MODEL_KEY, LOCAL_MODEL_PATH)
            # Line 5: Load the model using joblib
            model = joblib.load(LOCAL_MODEL_PATH)
            print("Model loaded successfully.")
        except Exception as e:
            print(f"Error loading model: {e}")
            raise e
    return model

def lambda_handler(event, context):
    # Line 6: Load the model (or use existing loaded model)
    current_model = load_model()

    # Line 7: Parse the request body (assuming JSON input from API Gateway)
    try:
        body = json.loads(event['body'])
        ipl_wins = body['ipl_wins']
        ipl_team_score = body['ipl_team_score']
    except KeyError:
        return {
            'statusCode': 400,
            'body': json.dumps('Missing "ipl_wins" or "ipl_team_score" in request body
        }
    except json.JSONDecodeError:
        return {
            'statusCode': 400,
            'body': json.dumps('Invalid JSON in request body.')
        }
```

```
            'statusCode': 500,
            'body': json.dumps(f'Error parsing input: {str(e)}')
        }

    # Line 8: Prepare input for the model (must match training features order)
    # Create a DataFrame for consistent input to scikit-learn model
    input_data = pd.DataFrame([[ipl_wins, ipl_team_score]], columns=['ipl_wins', 'ipl_

    # Line 9: Make prediction
    try:
        prediction = current_model.predict(input_data)[0]
        # Line 10: Convert numpy.int64 to standard int for JSON serialization
        prediction_result = int(prediction)
        print(f"Prediction for ipl_wins={ipl_wins}, ipl_team_score={ipl_team_score}: {

        return {
            'statusCode': 200,
            'headers': {
                'Content-Type': 'application/json'
            },
            'body': json.dumps({
                'prediction': prediction_result,
                'message': 'Team can win' if prediction_result == 1 else 'Team cannot
            })
        }
    except Exception as e:
        print(f"Error during prediction: {e}")
        return {
            'statusCode': 500,
            'body': json.dumps(f'Error during prediction: {str(e)}')
        }
```

- **Line-by-line Python explanation:**
  - `import json, boto3, os, joblib, io, pandas as pd`: Imports necessary libraries.
  - `BUCKET_NAME = os.environ.get('S3_BUCKET_NAME')`, `MODEL_KEY = 'ipl_winner_predictor.joblib'`, `LOCAL_MODEL_PATH = '/tmp/ipl_winner_predictor.joblib'`: Defines constants for S3 bucket, model file name, and local temporary path in Lambda.
  - `model = None`: Initializes a global variable to store the loaded model. This is a common Lambda optimization for "warm starts."
  - `load_model()`: A helper function that downloads and loads the model only if it hasn't been loaded already. This saves time on subsequent invocations.
  - `s3_client.download_file(BUCKET_NAME, MODEL_KEY, LOCAL_MODEL_PATH)`: Downloads the model file from S3 to the Lambda's temporary storage.

`joblib`.

- `body = json.loads(event['body'])`: Parses the incoming JSON request body from API Gateway.
- `ipl_wins = body['ipl_wins']`, `ipl_team_score = body['ipl_team_score']`: Extracts the input features from the JSON payload.
- `input_data = pd.DataFrame(...)`: Creates a pandas DataFrame from the input. This is important because the scikit-learn model was trained with pandas DataFrames, so it expects input in a similar structure.
- `prediction = current_model.predict(input_data)[0]`: Uses the loaded model to make a prediction based on the input data. `[0]` extracts the single prediction value.
- `prediction_result = int(prediction)`: Converts the NumPy integer prediction to a standard Python integer for JSON serialization.
- `return {'statusCode': 200, ...}`: Returns the prediction and a message as a JSON response.
- `except` blocks: Handle potential errors during input parsing or prediction.

- Click **"Deploy"** to save your code.

4. **Configure Environment Variables for Inference Lambda:**
   - In your `IPLPredictionInference` function, under "Configuration" -> "Environment variables".
   - Click "Edit" and add:
     - Key: `S3_BUCKET_NAME`
     - Value: Your S3 bucket name

Now, set up API Gateway:

1. **Create an API Gateway REST API:**
   - Go to the AWS Management Console -> API Gateway.
   - Click **"Create API"**.
   - Under **"REST API"**, click **"Build"**.
   - **Choose the protocol:** Select **"REST"**.
   - **Create new API:** Select **"New API"**.
   - **API name:** `IPLPredictionAPI`

- Click **"Create API"**.

2. **Create a Resource:**
   - In the left navigation, under your `IPLPredictionAPI`, select **"Resources"**.
   - Click **"Actions"** -> **"Create Resource"**.
   - **Resource Name:** `predict` (This will form part of your API URL, e.g., `/predict`).
   - **Resource Path:** `/predict`
   - Click **"Create Resource"**.
3. **Create a Method (POST):**
   - Select the newly created `/predict` resource.
   - Click **"Actions"** -> **"Create Method"**.
   - Select **"POST"** from the dropdown and click the checkmark.
   - **Integration type:** Select **"Lambda Function"**.
   - **Lambda Region:** Select your AWS Region (e.g., `us-east-1`).
   - **Lambda Function:** Start typing `IPLPredictionInference` and select your Lambda function.
   - Click **"Save"**.
   - A pop-up will ask for permission to grant API Gateway to invoke your Lambda function. Click **"OK"**.
4. **Deploy the API:**
   - With your API selected, click **"Actions"** -> **"Deploy API"**.
   - **Deployment stage:** Select **"[New Stage]"**.
   - **Stage name:** `prod` (for production)
   - Click **"Deploy"**.
5. **Get the Invoke URL:**
   - After deployment, you will see the **"Invoke URL"** for your API. It will look something like: `https://xxxxxxxxx.execute-api.us-east-1.amazonaws.com/prod/predict`. Copy this URL.

- **Student Task:** Create the `IPLPredictionInference` Lambda function, add S3 ReadOnly access to its role, paste the inference Python code (update S3 bucket name), configure the

`/predict` resource, a `POST` method integrated with your inference Lambda, and finally deploy the API. Copy the Invoke URL.

- **Troubleshooting Tips:**
  - **"Internal Server Error" (500) from API Gateway:** This usually means the Lambda function returned an error. Check the CloudWatch logs for the `IPLPredictionInference` Lambda function.
  - **"Missing Authentication Token" (403):** This might indicate an issue with the API Gateway deployment or a misconfigured method. Ensure the API is deployed to a stage.
  - **"Endpoint response body is not valid JSON"**: Your Lambda function's return format must be a valid JSON string (e.g., `json.dumps({'key': 'value'})`).

5. Build the IPL MLOps Pipeline (2 hrs)

Mentor Goal: Bring all the individual service setups together to form a cohesive, end-to-end MLOps pipeline.

Real-life project: “Predict if an IPL Team Can Win”

We will use our simple dataset and trained model to predict if an IPL team, given its `ipl_wins` and `ipl_team_score`, is likely to "can win" (based on our simple definition: `ipl_wins >= 3` and `ipl_team_score >= 2800`).

Full pipeline steps (covered in previous setup, now demonstrated end-to-end):

1. **Upload CSV to S3:** We already uploaded `ipl_data.csv` to our S3 bucket.
2. **Trigger Lambda for Preprocessing:** We will manually trigger `IPLDataPreProcessor` to simulate a data update event.
3. **Train model in SageMaker:** We manually ran the SageMaker notebook to train and save the model.
4. **Save model to S3:** The SageMaker notebook saved the `ipl_winner_predictor.joblib` model to S3.
5. **Deploy Lambda for inference:** We created and deployed `IPLPredictionInference` Lambda.
6. **Connect API Gateway:** We set up API Gateway to invoke `IPLPredictionInference`.
7. **Monitor logs via CloudWatch:** We know how to check CloudWatch for logs.

✅ **Task: Build and test the full pipeline**

- **Mentor Instructions:** Guide students through the entire flow, showing how each component interacts.

- Go to Lambda console.
- Select `IPLDataPreProcessor`.
- Go to "Test" tab and click "Test" with your configured test event.
- Verify in S3 that `processed_ipl_data.csv` is updated (check its last modified timestamp).

2. **Confirm Model in S3:**
   - Go to S3 console.
   - Navigate to your bucket.
   - Confirm `ipl_winner_predictor.joblib` is present.
3. **Test the Inference API Gateway Endpoint:**
   - Open a terminal or use an online `curl` tool.
   - Use the Invoke URL you copied from API Gateway.
   - **JSON example for API call:**

     JSON

     ```
     {
         "ipl_team": "KKR",
         "ipl_wins": 3,
         "ipl_team_score": 2800
     }
     ```

   - **Curl command example:** (Replace with your actual Invoke URL)

     Bash

     ```
     curl -X POST -H "Content-Type: application/json" -d '{ "ipl_wins": 3, "ipl_team_score": 28
     ```

     - **Explanation:**
       - `-X POST`: Specifies the HTTP method as POST.
       - `-H "Content-Type: application/json"`: Sets the request header to indicate that the body is JSON.
       - `-d '{ "ipl_wins": 3, "ipl_team_score": 2800 }'`: Provides the JSON data for the request body. **Note: `ipl_team` is not needed for the model inference, only `ipl_wins` and `ipl_team_score`.**

`1.amazonaws.com/prod/predict`: Your API Gateway Invoke URL.

- **Expected Output:**

  JSON

  ```
  {"prediction": 1, "message": "Team can win"}
  ```

  or

  JSON

  ```
  {"prediction": 0, "message": "Team cannot win"}
  ```

  depending on the input values.

4. **Monitor Logs in CloudWatch (for inference):**
   - Immediately after making the API call, go to CloudWatch Logs -> Log groups.
   - Click on `/aws/lambda/IPLPredictionInference`.
   - Check the latest log stream for details of your API call, including print statements and any errors.

- **Common error solutions during pipeline testing:**
  - **`curl` command errors:** Double-check the URL, ensure proper JSON formatting (`"` around keys and string values), and correct single/double quotes in the terminal.
  - **Lambda returning "Internal Server Error" (500):** This is the most common. Go to CloudWatch logs for the *inference* Lambda. Look for Python tracebacks. Common issues:
    - **Model not found:** Check if `ipl_winner_predictor.joblib` exists in the correct S3 bucket and path.
    - **Incorrect input format to model:** Ensure the `pd.DataFrame` creation in the Lambda matches the feature names and order used during training (`ipl_wins`, `ipl_team_score`).
    - **Missing `pandas` or `scikit-learn` in Lambda:** If you did not create a Lambda Layer for these, the inference Lambda will fail. For this workshop, if you hit this, you would need to:
      - **Mentor Tip:** Briefly explain Lambda Layers for Python dependencies. For this workshop, if time is a constraint, we might suggest skipping the Pandas DataFrame for inference input and converting inputs to a simple list/array directly if the model allows it (most scikit-learn models prefer 2D arrays). Or, if feasible, quickly guide them to create a simple Lambda layer with pandas and scikit-learn (this is outside Free Tier for time/complexity but good to mention).

inference Lambda completely for simplicity and Free Tier, you would change `input_data = pd.DataFrame(...)` to `input_data = [[ipl_wins, ipl_team_score]]` and ensure your model `predict` method can accept a list of lists/array directly, which LogisticRegression can.

Python

```
# ... (previous code) ...
    # Line 8: Prepare input for the model (must match training features order)
    # No pandas DataFrame needed if model accepts list of lists/numpy array directly
    input_data = [[ipl_wins, ipl_team_score]] # Just a list of lists

    # Line 9: Make prediction
    try:
        prediction = current_model.predict(input_data)[0]
# ... (rest of code) ...
```

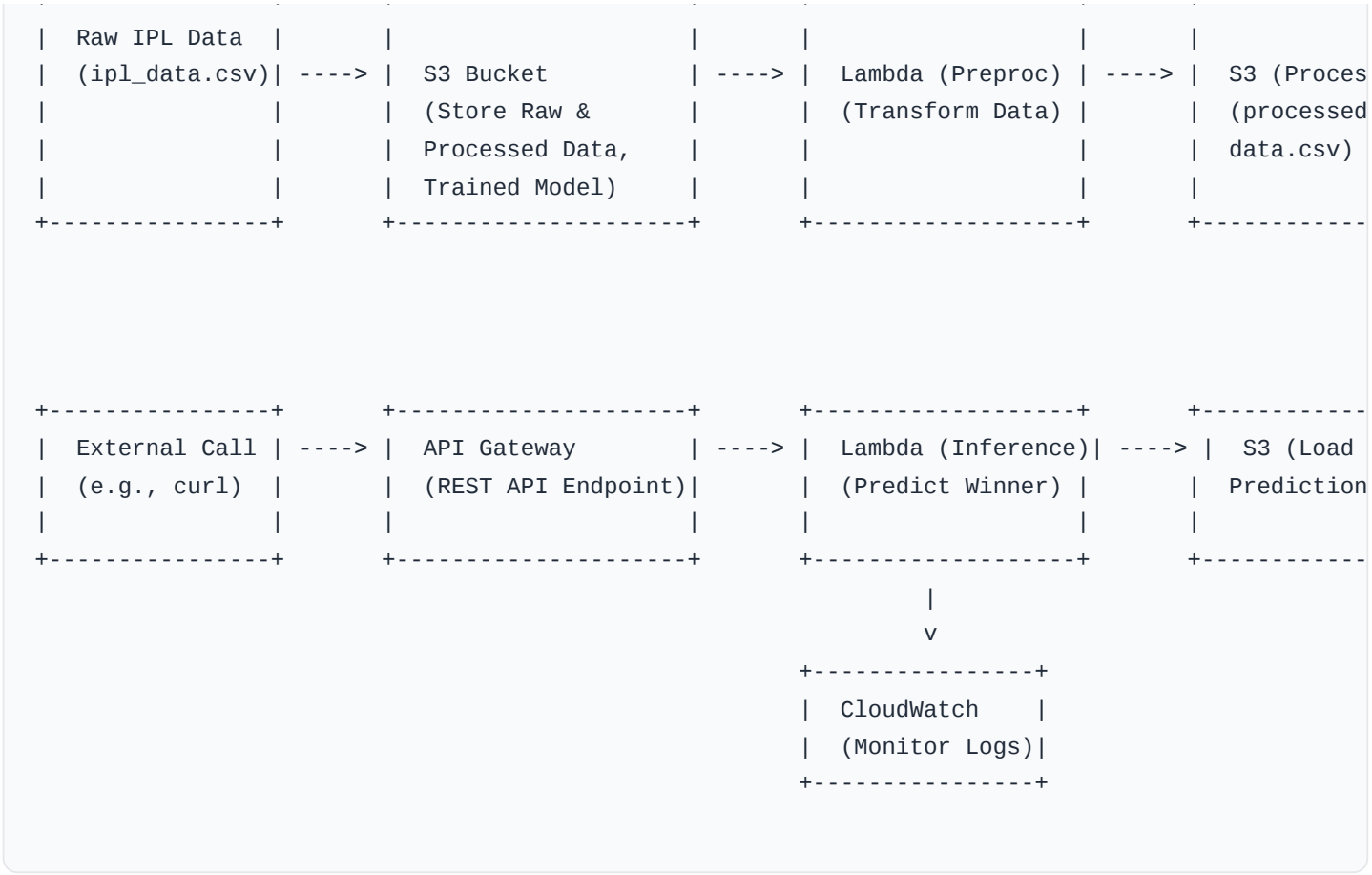
- **Re-deploy this simplified inference Lambda if you make changes.**

6. Visual Recap & Architecture Diagram (15 mins)

Mentor Goal: Solidify the understanding of the pipeline's components and their interactions with a visual representation.

Provide full pipeline diagram (describe services & flow):

Concept: Imagine a flow where data enters, gets processed, a model is trained, and then that model is available to make predictions via an API.



Explanation of Flow:

1. **Raw IPL Data to S3:** Our initial `ipl_data.csv` is uploaded to an S3 bucket. S3 acts as our data lake.
2. **S3 to Lambda (Preprocessing):** When new raw data arrives (simulated by manually running the Lambda), our `IPLDataPreProcessor` Lambda function is triggered. It reads the raw data from S3.
3. **Lambda (Preprocessing) to S3 (Processed Data):** The Lambda processes the data (adds the `can_win` column) and saves the `processed_ipl_data.csv` back to the same S3 bucket.
4. **S3 (Processed Data) to SageMaker:** A SageMaker Notebook Instance accesses this `processed_ipl_data.csv` from S3.
5. **SageMaker (Train Model) to S3 (Trained Model):** The SageMaker notebook trains a Logistic Regression model and then saves the `ipl_winner_predictor.joblib` (the trained model artifact) back to S3.
6. **External Call to API Gateway:** An external application or user makes an HTTP POST request to our deployed API Gateway endpoint.
7. **API Gateway to Lambda (Inference):** API Gateway receives the request and invokes our `IPLPredictionInference` Lambda function.
8. **Lambda (Inference) to S3 (Load Model):** The Inference Lambda, on its first invocation (or if not warm), downloads the trained model (`ipl_winner_predictor.joblib`) from S3 into its `/tmp`

9. **Lambda (Inference) Prediction & Response:** The Lambda uses the loaded model to make a prediction based on the input data from API Gateway and returns the prediction result back through API Gateway to the caller.
10. **CloudWatch:** Both Lambda functions send their logs and metrics to CloudWatch, allowing us to monitor their execution, debug issues, and see print statements.

✅ **Task: Students redraw pipeline architecture from memory**

- **Mentor Instructions:** Give students a few minutes to sketch the diagram. Emphasize understanding the data flow and which service does what.
- **Student Task:** On a blank sheet or digital canvas, draw the end-to-end MLOps pipeline, including all the AWS services and the data flow between them.

7. Free Tier Optimization & Cleanup (15 mins)

Mentor Goal: Teach participants how to manage their AWS resources to stay within Free Tier limits and avoid accidental charges. This is extremely important!

Tips to avoid charges:

- **Delete Resources When Done:** The single most important rule. If you're not using a resource, delete it.
- **Stop/Terminate Instances:** For services like SageMaker Notebook Instances or EC2 instances, "stopping" an instance typically stops billing for compute time, but you might still pay for storage. "Terminating" completely deletes it. For SageMaker, "stopping" is usually sufficient for short breaks, but for a workshop, ensure it's stopped.
- **S3 Lifecycle Rules:** Configure rules to automatically transition objects to cheaper storage classes (e.g., Glacier) or delete them after a certain period if they are no longer needed.
- **CloudWatch Alarms:** Set up billing alarms as discussed earlier to get notified if your estimated charges exceed a threshold.
- **Understand Data Transfer Costs:** Data transfer *out* of AWS regions can incur costs. Keep resources in the same region.
- **Lambda Concurrent Executions:** Keep Lambda invocations low. For a workshop, this is naturally controlled.

✅ **Task: Set lifecycle rule in S3 and remove unused Lambdas**

- **Mentor Instructions:** Guide students through setting an S3 lifecycle rule and deleting a Lambda function (if they created any test ones beyond the two required).

1. **Set S3 Lifecycle Rule:**

- Click on your workshop bucket.
- Go to the **"Management"** tab.
- Click **"Create lifecycle rule"**.
- **Lifecycle rule name:** `DeleteOldObjects`
- **Choose a rule scope:** Select **"Apply to all objects in the bucket"**. Acknowledge the warning.
- **Lifecycle rule actions:**
  - Check **"Transition current versions of objects between storage classes"**.
  - Click **"Add transition"**:
    - **Days after creation:** `30`
    - **Choose storage class:** `Glacier Instant Retrieval` (or `Glacier Deep Archive` for cheapest, but retrieval costs vary).
  - Check **"Expire current versions of objects"**.
  - **Days after creation:** `60` (This means objects will be permanently deleted after 60 days).
- Click **"Create rule"**.
- **Explanation:** This rule will automatically move objects to cheaper storage after 30 days and then delete them entirely after 60 days. This is a good practice for cleaning up old data you might not need forever.

2. **Remove Unused Lambdas (if any):**
   - Go to the Lambda console.
   - Identify any functions you created that are not `IPLDataPreProcessor` or `IPLPredictionInference` (e.g., if you made test functions).
   - Select the function(s) you want to delete.
   - Click **"Actions"** -> **"Delete"**. Confirm the deletion.
3. **Delete API Gateway:**
   - Go to the API Gateway console.
   - Select your `IPLPredictionAPI`.
   - Click **"Actions"** -> **"Delete"**. Confirm by typing the API name.
   - **Mentor Tip:** Deleting the API Gateway will also remove the associated stage and method.

- **Crucial:** Remind students again to ensure their SageMaker Notebook Instance is **STOPPED**.
- If they are completely done and do not plan to resume, they can also **Delete** it (Actions -> Delete). Deleting is safer for avoiding future charges related to underlying EBS volumes.

- **Student Task:** Implement an S3 lifecycle rule for their bucket to expire objects after 60 days. Ensure their SageMaker Notebook Instance is stopped (or deleted). Delete their API Gateway.

8. Practice Exercises (45 mins)

Mentor Goal: Provide hands-on challenges to reinforce learning and build problem-solving skills.

Exercise 1: Add a new team to CSV → re-run pipeline

- **Task:**
  1. Download `ipl_data.csv` from your S3 bucket.
  2. Add a new line for another IPL team (e.g., `LSG,6,3000`).
  3. Upload the modified `ipl_data.csv` back to S3 (overwrite the existing one).
  4. Manually trigger the `IPLDataPreProcessor` Lambda function.
  5. Verify `processed_ipl_data.csv` in S3 has the new team and its `can_win` value.
  6. Re-run the SageMaker notebook (start the instance, open Jupyter, run all cells, stop instance).
  7. Test the API Gateway endpoint with the new team's `ipl_wins` and `ipl_team_score` to see the prediction.
- **Learning Focus:** Understanding data updates, re-triggering preprocessing, and re-training implications.

Exercise 2: Break Lambda → Debug using CloudWatch

- **Task:**
  1. Go to your `IPLPredictionInference` Lambda function.
  2. Intentionally introduce an error in the Python code (e.g., change `json.loads(event['body'])` to `json.loads(event['body_typo'])`).
  3. Deploy the Lambda.
  4. Make an API Gateway call to trigger this broken Lambda.
  5. Observe the "Internal Server Error" from API Gateway.
  6. Go to CloudWatch Logs for `IPLPredictionInference`.

8. Fix the Lambda code, deploy, and re-test until it works.

- **Learning Focus:** Practical debugging using CloudWatch logs.

Exercise 3: Modify IAM to restrict access → test

- **Task:**
  1. Go to IAM console.
  2. Find the role for your `IPLDataPreProcessor` Lambda (e.g., `IPLDataPreProcessor-role-xxxxxx`).
  3. Edit the attached `AmazonS3FullAccess` policy. Remove the `s3:PutObject` permission specifically for your bucket, or change the resource to `arn:aws:s3:::some-other-bucket/*` to deny access to your actual bucket.
  4. Go back to the Lambda console and try to run the `IPLDataPreProcessor` function.
  5. Observe the "Access Denied" error in the CloudWatch logs for this Lambda.
  6. Revert the IAM policy change to restore access.
- **Learning Focus:** Understanding the importance of IAM permissions and how to troubleshoot `Access Denied` errors.

Exercise 4 (optional): Clone pipeline for another sport

- **Task:**
  1. Create a new S3 bucket (e.g., `football-mlops-workshop-yourname`).
  2. Create a new CSV for a different sport (e.g., `football_data.csv` with columns like `team,games_played,goals_scored` and a `can_win` logic).
  3. Create new Lambda functions (e.g., `FootballDataPreProcessor`, `FootballPredictionInference`) and modify their code to use the new dataset and your custom prediction logic.
  4. Create a new SageMaker notebook and train a model for the football data.
  5. Create a new API Gateway endpoint for football predictions.
  6. Test the new end-to-end pipeline.
- **Learning Focus:** Reinforces the entire pipeline creation process, showing reusability and adaptability. (This is a significant time commitment, hence "optional").

Workshop Conclusion:

Prediction Use Case" workshop! You've successfully built a foundational MLOps pipeline using core AWS services, staying within the Free Tier. Remember the key takeaways: automate, monitor, and clean up your resources!

Further Learning:

- Explore AWS Step Functions for orchestrating your MLOps pipeline steps.
- Investigate SageMaker Pipelines for native MLOps pipeline building.
- Learn about AWS CodeCommit, CodeBuild, CodePipeline for CI/CD with ML.
- Dive deeper into SageMaker training jobs and inference endpoints.