



D.Y.Patil International University, Akurdi, Pune

**School of Computer Science Engineering and
Applications (SCSEA)**

Final Year Engineering (B.Tech)

Cloud Computing for Artificial Intelligence CSE 4001

Lab Manual



Vision of the University:

"To Create a vibrant learning environment – fostering innovation and creativity, experiential learning, which is inspired by research, and focuses on regionally, nationally and globally relevant areas."

Mission of the School:

- *To provide a diverse, vibrant and inspirational learning environment.*
- *To establish the university as a leading experiential learning and research oriented center.*
- *To become a responsive university serving the needs of industry and society.*
- *To embed internationalization, employability and value thinking*

Cloud Computing for Artificial Intelligence

Course Objectives:

1. To understand the fundamental concepts of AI/ML and their application in cloud environments.
2. To learn how to build, train, and deploy machine learning models using cloud platforms.
3. To gain practical experience in using cloud services for natural language processing, speech processing, and image/video analysis.
4. To develop skills in designing and implementing intelligent serverless workflows using cloud AI services.
5. To understand the security and ethical considerations in deploying AI/ML applications on the cloud.

Course Outcomes:

On completion of the course, learner will be able to—

| | |
|------------|--|
| CO1 | (Understanding) Explain the differences between AI, Machine Learning, and Deep Learning and describe their various applications across different domains. |
| CO2 | (Applying) Build, train, and deploy machine learning models using Amazon SageMaker, including data preprocessing, hyperparameter tuning, and model evaluation. |
| CO3 | (Analyzing) Analyze and compare the performance of different cloud-based NLP, speech processing, and image/video analysis services for specific tasks. |
| CO4 | (Creating) Design and implement intelligent serverless workflows integrating various AWS services like Lambda, S3, DynamoDB, and API Gateway for an AI application. |
| CO5 | (Evaluating) Critically evaluate the security, ethical, and scalability aspects of deploying AI/ML applications on cloud platform. |

Rules and Regulations for Laboratory:

- Students should be regular and punctual to all the Lab practical
- Lab assignments and practical should be submitted within given time.
- Mobile phones are strictly prohibited in the Lab.
- Please shut down the Computers before leaving the Lab.
- Please switch off the fans and lights and keep the chair in proper position before leaving the Lab
- Maintain proper discipline in Lab

D.Y.Patil International University, Akurdi, Pune
School of Computer Science Engineering and Applications
Index

| Sr. No. | Name of the Practical | Date of Conduction | Page No. | | Sign of Teacher | Remarks* |
|---------|---|--------------------|----------|----|-----------------|----------|
| | | | From | To | | |
| 1. | Setting up AWS Account and IAM Roles : To configure an AWS account and create IAM roles with appropriate permissions for accessing AI/ML services. | 1/09/2025 | | | | |
| 2. | Exploring SageMaker Notebook Instances: To familiarize oneself with the SageMaker environment and create a notebook instance for developing and running ML code. | 8/09/2025 | | | | |
| 3. | Building a Simple Linear Regression Model with SageMaker: To train and deploy a linear regression model using built-in algorithms in SageMaker | 15/09/2025 | | | | |
| 4. | Implementing Data Preprocessing using SageMaker Notebooks: To perform data cleaning, transformation, and feature engineering using SageMaker notebooks | 22/09/25 | | | | |
| 5. | Hyperparameter Tuning for a Classification Model: To optimize the performance of a classification model by tuning its hyperparameters using SageMaker | 29/09/25 | | | | |
| 6. | Sentiment Analysis using Amazon Comprehend: To use Amazon Comprehend to perform sentiment analysis on text data. | 06/10/25 | | | | |
| 7 | Building a Simple Conversational Bot with Amazon Lex: To design and implement a simple conversational bot using Amazon Lex. | 13/10/25 | | | | |
| 8 | Object Detection using Amazon Rekognition: To use Amazon Rekognition to perform object detection in images. | 27/10/2025 | | | | |

*Absent/Attended/Late/Partially Completed/Completed

CERTIFICATE

This is to certify that Mr. /Miss Suhani Deepak Nemade

PRN: 20220802404 of class: _____ has completed practical/term work in
the course of Advanced Artificial Intelligence of Final Year Engineering (B.Tech) within SCSEA, as
prescribed by D.Y.Patil International University, Pune during the academic year 2025 - 2026.

Date: **Teaching Assistant**

Faculty I/C

Director
(SCSEA)

Practical 01

| |
|---|
| Student Name: Suhani Deepak Nemade |
| Date of Experiment: |
| Date of Submission: |
| PRN No: 20220802404 |

Title of lab: Setting up AWS Account and IAM Roles : To configure an AWS account and create IAM roles with appropriate permissions for accessing AI/ML services.

Objective: To Create AWS Account and IAM Roles.

Tools:

- Personal Computer
- Web Browser
- Internet Connection

Software Used:

- Amazon Web Services (AWS) Management Console

Theory / Concept: The AWS Free Tier account provides new users with free access to a range of Amazon Web Services for 6 months, allowing them to explore and test AWS offerings without any upfront cost. The key benefits of AWS Free Tier include the ability to learn and experiment with AWS services, deploy and develop applications at no cost, and explore cloud solutions with minimal risk.

AWS Identity and Access Management (IAM) is a web service that helps you securely control access to AWS resources. You use IAM to control who is authenticated (signed in) and authorized (has permissions) to use resources. This lab involves creating a root account (during sign-up) and then creating individual IAM users with specific, limited permissions (e.g., access to S3, access to EC2). We will also organize these users into groups to manage permissions more efficiently, which is a security best practice.

Output:

1. IAM Dashboard: A screenshot of the main IAM dashboard after logging in.
2. User Creation: Screenshots showing the "Specify user details" and "Set permissions" steps for a new user (e.g., personl-s3 with AmazonS3FullAccess).
3. User List: A screenshot of the "Users" page showing all created users (e.g., personl-s3, personl-ec2, personl-sage).
4. Permission Test (Failure): A screenshot showing the "Access Denied" error when the personl-s3 user attempts to access the EC2 console.
5. Group Creation: A screenshot showing the "Create user group" page with a group name (e.g., s3-developers) and attached permissions.
6. Final Group List: A screenshot of the "User groups" page showing the newly created group.

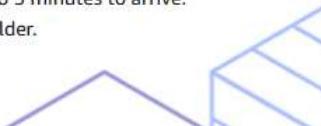
The image shows two screenshots of the AWS website. The top screenshot is the 'Free Cloud Computing Services - AWS Free Tier' page, featuring sections for AWS Free Tier FAQS, Migration, Compute, Serverless, Storage, and Analytics. It also includes a 'GeekforGeeks' link and a 'Amazon Web Services (AWS) - Free Tier Account Set up' button. The bottom screenshot is the main 'Amazon Web Services' homepage, displaying the company's logo, a brief history (founded July 2002), key people (Matt Green, CEO; J. J. Morris, COO), and links for English, Contact us, Support, and My account.

The bottom screenshot also shows a navigation bar with links for Agent AI, Discover AWS, Products, Solutions, Pricing, Resources, and a search bar. Below the navigation bar, there's a breadcrumb trail: AWS > AWS Free Tier. A message box asks if the user created an AWS account before July 15th, 2025, with a link to access the Legacy Free Tier. The main content area features the 'AWS Free Tier' section, which encourages users to gain free, hands-on experience with AWS products and services. It includes a 'Create a Free Account' button and a live chat window offering assistance from an AWS representative. A note at the bottom states that new customers get up to \$200 in credits.



Try AWS at no cost for up to 6 months

Start with USD \$100 in AWS credits, plus earn up to USD \$100 by completing various activities.



Sign up for AWS

Confirm you are you

Making sure you are secure -- it's what we do.

We sent an email with a verification code to
[\(not you?\)](#)

Enter it below to confirm your email.

Verification code

Verify

[Resend Code 55](#)

Didn't get the code?

- Codes can take up to 5 minutes to arrive.
- Check your spam folder.

The image shows a complex web-based sign-up interface for AWS. At the top is the AWS logo. Below it is a section titled "Try AWS at no cost for up to 6 months" with a brief description of the offer. To the right of this is a "Sign up for AWS" section. This section includes a "Create your password" form with fields for "Root user password" and "Confirm root user password". A green success message box says "It's good! Your email address has been successfully verified". Below the password fields is a note: "Your password provides you with sign in access to AWS; so it's important we got it right.". At the bottom of the sign-up form are two options: "Continue step 2 of 5" (in orange) and "Sign in to an existing AWS account" (with a small "OR" between them). The entire interface is framed by a decorative border of stylized server racks and clouds.

Sign up for AWS

Choose your account plan



Free (6 months)

- Learn, experiment, and build prototypes
 - ✓ Receive up to \$200 in credits
 - ✓ Includes free usage of select services
 - ✗ Workloads scale beyond credit thresholds
 - ✗ Access to all AWS services and features
- ⓘ After the 6 month free period or when all credits are used, you can choose to Upgrade to a paid plan. Otherwise, your account closes automatically.

[Choose free plan](#)



Paid

- Develop production-ready workloads
 - ✓ Receive up to \$200 in credits
 - ✓ Includes free usage of select services
 - ✓ Workloads scale beyond credit thresholds
 - ✓ Access to all AWS services and features
- ⓘ After all of your credits are used, you are charged using pay-as-you-go pricing.

[Choose paid plan](#)

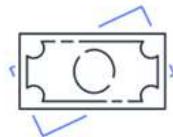
[View additional details](#)



Sign up for AWS

Earn additional AWS credits

Complete various activities to earn up to an additional USD \$100 in credits, such as creating your first AWS budget to monitor cloud costs.



Contact Information

How do you plan to use AWS?

- Business - for your work, school, or organization
- Personal - for your own projects

Who should we contact about this account?

Full Name

Country Code Phone Number

+1 222-333-4444

Country or Region

Address line 1

Address line 2

Apartment, suite, unit, building, floor, etc.

City

State, Province, or Region

Postal Code

I have read and agree to the terms of the [AWS Customer Agreement](#).

[Agree and Continue \(step 2 of 5\)](#)

Sign up for AWS

Billing Information

Why is this required?

Our verification process holds USD \$1 (or equivalent) for 3-5 days to verify your account and prevent fraud.

For the free plan, no charges occur until upgrade to a paid plan. Providing your billing information now enables a seamless upgrade to a paid plan.



Billing country
Your billing country determines the payment methods available to you to pay for AWS services.

India

Credit or Debit card number



AWS accepts most major credit and debit cards. To learn more about payment options, review our [FAQ](#).

Expiration date

Month Year

Security code [?](#)

CVV/CVC

Cardholder's name

Save card and charge automatically for future payments. [Learn more...](#)

[?](#) Automatic payments (e-mandates) currently doesn't support RuPay and AMEX cards.

Billing address:

Use my contact address
Flat No. 1203, Legacy urbana, Panditshwar Wasti
Pune Maharashtra 411033
IN

Use a new address

Do you have a PAN?
Permanent Account Number (PAN) is a ten-digit alphanumeric number issued by the Indian Income Tax Department. This 10-digit number is printed on the front of your PAN card.

Yes
 No

You can go on the Tax Settings Page on Billing and Cost Management Console to update your PAN information.

Verify and continue (step 3 of 5)

Sign up for AWS

Select a support plan

Choose a support plan for your business or personal account. Compare plans and pricing examples

You can change your plan anytime in the AWS Management Console.

| | | |
|---|---|--|
| <input checked="" type="radio"/> Basic support - Free + Recommended for small teams and getting started with AWS. + 24x7 performance monitoring, AWS CloudWatch Metrics. + Full account and billing traces only. + Access to Personal Health Dashboard & Trusted Advisor | <input type="radio"/> Developer support - From \$25/month + Recommended for developers and working with AWS. + Direct access to AWS Support during business hours. + 12 Business-hour response time | <input type="radio"/> Business support - From \$100/month + Recommended for organizations with multiple AWS accounts. + 24x7 technical support via email, phone, and chat. + 2-hour response times. + Full set of Standard Service Level Agreements (SLAs). |
|---|---|--|

Need Enterprise level support?
From \$10,000 a month you will receive 10-minute response times and concierge-style support with an assigned Technical Account Manager. [Learn more](#) 

Complete sign up

Language English

aws

Sign up for AWS

Confirm your identity [? Help](#)

Name [Edit](#)
Choose the name that you want to use for identity verification.

code bear

Primary purpose of account registration
Choose one that best applies to you. If your account is used for a business, select the one that applies to your business.

Select one

Ownership type
Select one

I consent to allowing AWS to use and send the information above to a third-party service for identity verification purposes.

[Continue \(Step 4 of 5\)](#)



English [▼](#)

aws



Congratulations!

Thank you for signing up for AWS.
We are activating your account, which should only take a few minutes. You will receive an email when this is complete.

[Go to the AWS Management Console](#)

Sign up for another account or contact sales.

Console Home [? Help](#)

Recently visited [View all services](#)

No recently visited services

Explore one of these commonly visited AWS services.

ECS | S3 | Amazon VPC | Lambda

Applications [? Help](#)

Angular Example (Standard)

Create application

Select Region [to-reach-1 \(Current Region\)](#) [Find a region](#)

Name Description Region Original

We applications. Get started by creating an application.

Create application

Go to my applications

Welcome to AWS [? Help](#)

Getting started with AWS [Learn the fundamentals and find valuable information to get the most out of AWS.](#)

Training and certificates [Learn from AWS experts and enhance your skills and knowledge.](#)

AWS Health [? Help](#)

Open issues 0 [View 7 days](#)

Scheduled changes 0 [Upcoming and past 7 days](#)

Other notifications 1 [View 7 days](#)

Cost and usage [? Help](#)

Data unavailable

You must have Cost Explorer enabled to view your cost and usage data.

Feedback [? Help](#) © 2023, Amazon Web Services, Inc. or its affiliates. Privacy Terms [Edit preferences](#)

IAM Dashboard

The IAM Dashboard provides an overview of your AWS Account's security posture. It includes sections for Security recommendations, IAM resources, What's new, AWS Account, Quick Links, and Tools.

Security recommendations:

- Add MFA for root user
- Root user has no active access keys

IAM resources:

| User groups | Users | Roles | Policies | Identity providers |
|-------------|-------|-------|----------|--------------------|
| 0 | 0 | 3 | 0 | 0 |

What's new:

- Updates for features in IAM
- AWS Lambda now supports AWS Lambda@Edge for serverless functions. (Preview)
- AWS Service Catalog now supports AWS Lambda@Edge for serverless functions. (Preview)
- AWS Lambda now supports AWS Lambda@Edge for serverless functions. (Preview)

AWS Account:

Account ID: 377235689640
Account Alias: Create
Sign-in URL for IAM users in this account: https://377235689640.signin.aws.amazon.com/home

Quick Links:

No security credentials

Tools:

Policy simulator

Additional information:

Security best practices in IAM

Users

The Users page lists all IAM users in the account. There are currently 0 users.

| User name | Path | Group | Last activity | MFA | Access key age | Console last sign in | Access key ID | Active key age | Access |
|-------------------------|------|-------|---------------|-----|----------------|----------------------|---------------|----------------|--------|
| No resources to display | | | | | | | | | |

Access management:

- User groups
- Users
- Roles
- Policies
- Identity providers
- Access settings
- Root access management

Access reports:

- Access analysis (New)
- Access history
- Access settings
- Credential report
- Organization activity
- Service control policies
- Assume role policies (New)

IAM Metrics Center
AWS Organizations

Create user

The Create user wizard is currently at Step 1: Specify user details.

Specify user details

User details:

User name:

You can use underscores and hyphens in the user name. Valid characters: A-Z, a-z, 0-9, and -_. (Up to 128 characters.)

Provide root access to the IAM Management Console - optional
If you're providing temporary access to a primary AWS account, [to manage their access in IAM Identity Center](#).

If you are granting programmatic access through access keys or service-specific credentials for AWS CodeCommit or Amazon RDS, you can generate them after you create this IAM user.
[Generate keys](#)

Next Step **Cancel**

Review and create

Please plan clients. After you create the user, you can view and download the autogenerated password. If needed.

| | | |
|--|--|---------------------------------------|
| User details | Console password type: Autogenerated | Require password reset: Yes |
| User name: sharon_hartmann_123 | | |
| Permissions summary | | |
| Name | Type | User on |
| sharon_hartmann | Not assigned | Temporary policy |
| Tags · optional | | |
| Tags are key-value pairs you can add to API resources to help identify, organize, or search for resources. Choose any tags you want to associate with this user. | | |
| No tags associated with this resource. | | |
| Add new tag <small>You can add up to 50 tags.</small> | | |
| Cancel Previous Create user | | |

User created successfully.

You can view and download the user's password and sign-in instructions for signing in to the AWS Management Console.

[View user](#) [X](#)

Step 1: Specify user details

Step 2: Set permissions

Step 3: Review and create

Step 4: Retrieve password

Retrieve password

You can view and download the user's password below or email sign-in instructions for signing in to the AWS Management Console. This is the only time you can view and download this password.

[Email sign-in instructions](#)

Console sign-in details

Console sign-in URL:
<https://CT7723M027000.signin.aws.amazon.com/console>

User name:
[admin_johnsmith_111](#)

Console password:
[XXXXXXXXXXXX](#) [Show](#)

[Cancel](#) [Download .CSV file](#) [Retain my password](#)

The screenshots illustrate the process of managing users and groups in the AWS IAM service.

Screenshot 1: IAM Dashboard

- Security recommendations:**
 - Add MFA for root user: Adds MFA for the root user to enable multi-factor authentication (MFA) for the root user and improve security for this account.
 - Root user has no active access keys: Directs users to the IAM User Access Keys page to manage access keys.
- IAM resources:**
 - Resources in this AWS Account: Shows 1 User, 3 Roles, 0 Policies, and 0 Identity providers.
 - What's new: Lists recent changes in IAM, including:
 - Amazon Rekognition introduces API keys for unenrolled fingerprints, 1 month ago
 - AWS Service Reference Information now supports annotations for service details, 1 month ago
 - AWS Lambda now supports custom policies (KDS) to support up to two additional arguments, 4 months ago
 - AWS IAM now enables MFA for root users across all account types, 4 months ago
- AWS Account:**
 - Account ID: 1TTT750000000000
 - Account Alias: Untitled
 - Sign-in URL for IAM users in this account: https://1TTT750000000000.signin.aws.amazon.com/home
- Quick Links:**
 - My security credentials
 - Manage your account via AWS multi-factor authentication (MFA) and other credentials
- Tools:**
 - Policy calculator
 - The structure evaluates the policies that you choose and determines the effective permissions for each of the actions that you specify.
- Additional information:**
 - Security best practices in IAM
 - IAM documentation

Conclusion: We created Multiple users using the IAM Service on AWS Console, and saw that users with no permissions are unable to access features not given to them. We also saw how to create groups and gave permissions to user in specific groups.

Practical 02

| |
|---|
| Student Name: Suhani Deepak Nemade |
| Date of Experiment: |
| Date of Submission: |
| PRN No: 20220802404 |

Title of lab: Exploring SageMaker Notebook Instances: To familiarize oneself with the SageMaker environment and create a notebook instance for developing and running ML code.

Objective: To familiarize oneself with the SageMaker environment and create a notebook instance for developing and running ML code.

Tools:

- Personal Computer
- Web Browser
- Internet Connection

Software Used:

- Amazon Web Services (AWS) Management Console
- Amazon SageMaker
- Jupyter Notebook

Theory / Concept: Amazon SageMaker is a fully managed cloud service from Amazon Web Services (AWS) that enables developers and data scientists to build, train, and deploy machine learning (ML) models at scale. It simplifies the entire ML workflow, from data labeling and preparation to model training, tuning, and deployment for real-time predictions or batch processing.

A core component of this service is the SageMaker Notebook Instance. This is essentially a managed cloud server (an EC2 instance) that comes pre-configured with a Jupyter Notebook or JupyterLab environment.

Output:

1. SageMaker Dashboard: A screenshot of the main Amazon SageMaker service dashboard.
2. Notebook Instances Page: A screenshot of the "Notebook instances" page before creation.
3. Create Notebook Instance: A screenshot of the "Create notebook instance" configuration page, showing the instance name and type.
4. Instance Pending: A screenshot showing the notebook instance with the "Pending" status.
5. Instance InService: A screenshot showing the notebook instance with the "InService" status, with the "Open Jupyter" link visible.
6. Jupyter Environment: A screenshot of the open Jupyter Notebook environment.
7. New Notebook: A screenshot showing the creation of a new "conda_python3" notebook from the Jupyter "New" dropdown.
8. Running Notebook: A screenshot of the blank, untitled notebook, ready for code.

The screenshots illustrate the workflow for creating and running a Jupyter Notebook in Amazon SageMaker. The first screenshot shows the initial state where no notebook instances have been created. The second screenshot shows the final state where a new notebook has been created and is running, ready for code execution.

Signed In: 27255 000-0000-0000-0000-000000000000

Amazon SageMaker AI Notebook Instances

Getting started Deployment Workflows Applications and IOEs Admin configurations Jupyter

Notebook instances

Your notebook instance is being created. Open the notebook instance when status is **Running** and open a Jupyter notebook to get started.

Notebook Instances

Name: LAB0 Instance type: ml.t2.medium Creation time: 10/8/2023, 4:56:22 AM Status: Pending Actions

View details

Create notebook instance

Search

Ajaxstart Foundation models Computer vision models Natural language processing models

Governance

Dashboard Notebook

jupyter

File View GR Settings Help

File Running

New Upload

Name

Modified File Size

Amazon SageMaker AI Notebook Instances Create notebook instance

Amazon SageMaker provides pre-built, fully managed notebook instances that run Jupyter notebooks. The notebook instances include reusable code for common model training and hosting scenarios. [Learn more](#)

Notebook instance settings

Notebook instance name: LAB0

Notebook instance type: ml.t2.medium

Platform identifier: [Create new](#) [LAB0](#)

Amazon Linux 2, Jupyter Lab 4

Additional configuration

Permissions and encryption

MM role: AmazonSageMakerServiceRoleForJupyterNotebook

Create role using the role creation wizard [Create role](#)

Root access - system:

Enable - Give users root access to the notebook

Feedback

Amazon SageMaker AI Notebook Instances

Succeeded Your notebook instance is being created. Open the notebook instance when status is InService and open a template notebook to get started.

Notebook instances [View details](#)

Search notebook instances

| Name | Instance | Creation time | Status | Actions |
|----------|------------|-----------------------|-----------|----------------------|
| Untitled | m1.1xlarge | 9/18/2023, 6:06:16 PM | InService | Edit |

Create notebook instances

Dashboard [Feedback](#)

Amazon SageMaker AI Notebook Instances

Succeeded Your notebook instance is being created. Open the notebook instance when status is InService and open a template notebook to get started.

Notebook instances [View details](#)

Search notebook instances

| Name | Instance | Creation time | Status | Actions |
|----------|------------|-----------------------|-----------|----------------------|
| Untitled | m1.1xlarge | 9/18/2023, 6:06:16 PM | InService | Edit |

Create notebook instances

Dashboard [Feedback](#)

jupyter

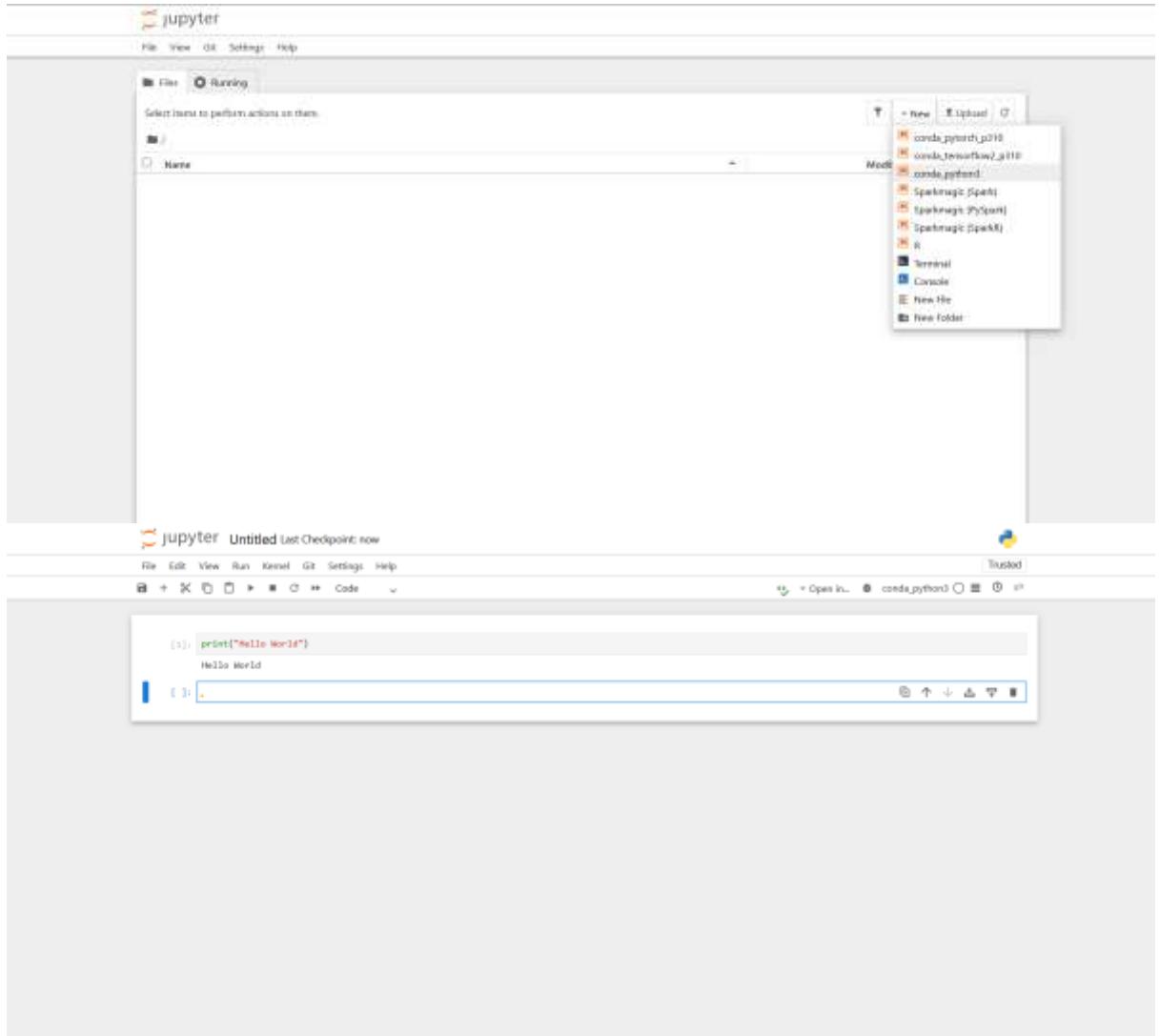
File View GR Settings Help

File Running

Open Download Rename Duplicate Move to S3

New Upload

Name Modified File Size



Conclusion: This lab provided a successful, hands-on introduction to Amazon SageMaker by guiding us through the creation and launch of a Notebook Instance. We experienced how these instances serve as a fully managed and pre-configured development environment, which eliminates infrastructure overhead and allows an immediate focus on machine learning code. By verifying the instance's access to its IAM role and default S3 bucket, we confirmed its seamless integration within the wider AWS ecosystem. Critically, this exercise highlighted the importance of diligent resource management—stopping and deleting instances to control costs—a fundamental best practice that prepares us to now leverage this powerful tool for the complete ML lifecycle, from data preparation to model deployment.

Practical 03

| |
|---|
| Student Name: Suhani Deepak Nemade |
| Date of Experiment: |
| Date of Submission: |
| PRN No: 20220802404 |

Title of lab: Building a Simple Linear Regression Model with SageMaker: To train and deploy a linear regression model using built-in algorithms in SageMaker.

Objective: To use an Amazon SageMaker Notebook to train a simple linear regression model on data stored in S3, then visualize the model's fit and evaluate its performance directly within the notebook.

Tools:

- Personal Computer
- Web Browser
- Internet Connection

Software Used:

- Amazon Web Services (AWS) Management Console
- Amazon SageMaker (Notebook Instance)
- Amazon S3 (Simple Storage Service)
- Jupyter Notebook
- Python (with sagemaker, numpy, scikit-learn, and matplotlib libraries)

Theory / Concept:

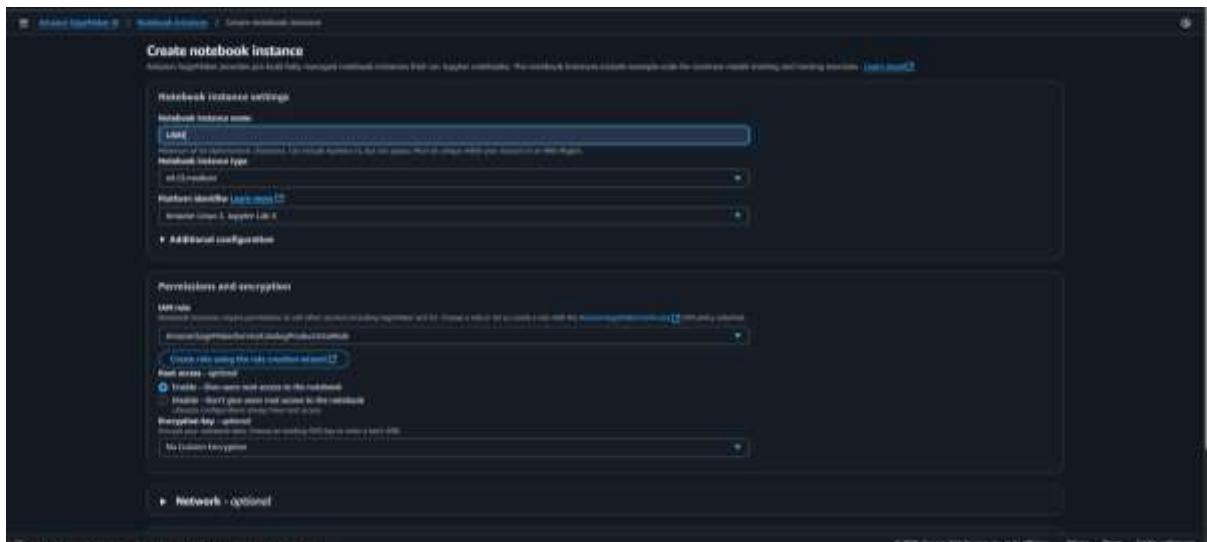
- Amazon SageMaker Notebook: A managed Jupyter notebook instance that provides a powerful, integrated environment for data

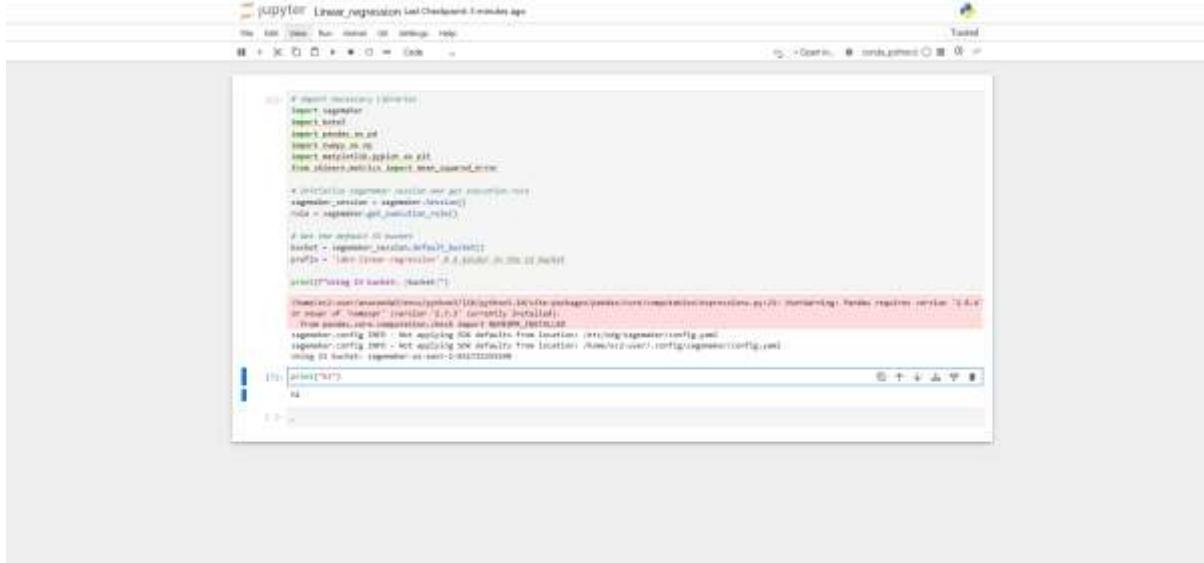
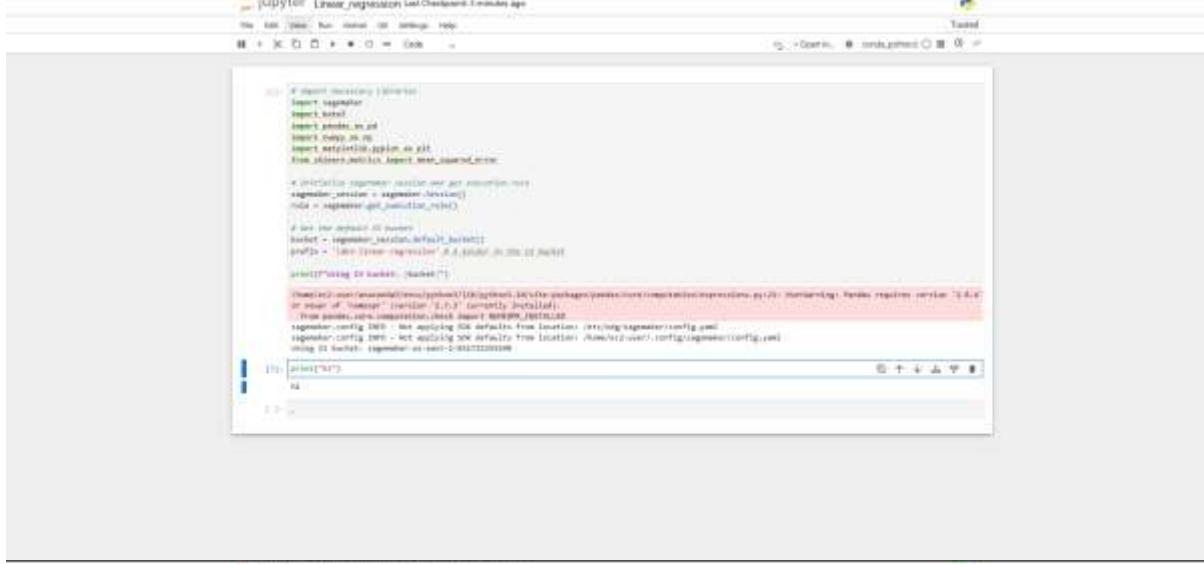
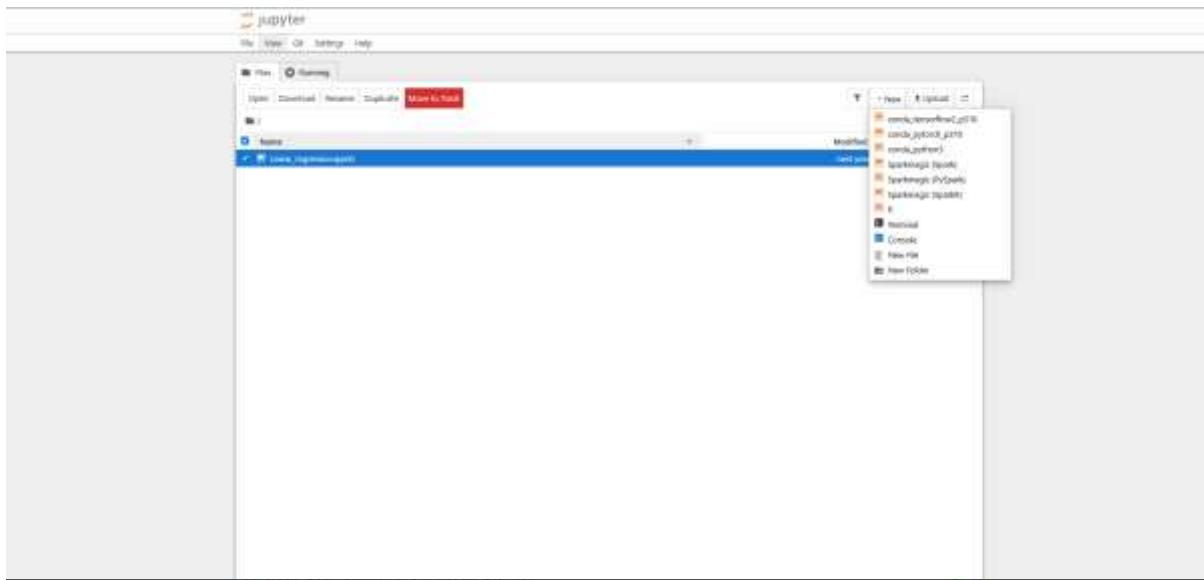
exploration, analysis, and model training. It comes pre-packaged with common data science libraries and AWS SDKs.

- Amazon S3 (Simple Storage Service): A scalable object storage service used here to store the training dataset. SageMaker training jobs are optimized to read data efficiently from S3.
- Linear Regression: A statistical method for modeling the relationship between a dependent variable (target) and an independent variable (feature) by fitting a straight line to the data.
- Data Visualization: The process of creating graphical representations of data. In this lab, we will use a scatter plot for our data points and a line plot for our model's predictions to visually assess how well the model fits the data.
- Mean Squared Error (MSE): A common metric used to measure the performance of a regression model. It calculates the average of the squared differences between the actual and predicted values, providing a measure of the model's error. A lower MSE indicates a better fit.

Output:

1. **Setup:** Screenshot of the Jupyter notebook cell importing libraries (`sagemaker`, `boto3`, `numpy`, etc.) and setting up the SageMaker session.
2. **Data Generation:** Screenshot of the code cell using NumPy to generate the synthetic dataset.
3. **S3 Upload:** Screenshot of the code cell uploading the '`train.csv`' file to S3, showing the output S3 path.
4. **Model Training:** Screenshot of the code cell where the `LinearRegression` model from scikit-learn is initialized, trained (`.fit()`), and used to generate predictions (`.predict()`).
5. **Evaluation (MSE):** Screenshot of the code cell calculating and printing the Mean SquaredError.
6. **Visualization:** Screenshot of the final Matplotlib plot showing the "Linear Regression Model Fit," with the blue "Actual Data" points and the red "Regression Line."







The screenshot shows a Jupyter Notebook interface with the following code:

```
# Import the linear regression module
from sklearn.linear_model import LinearRegression

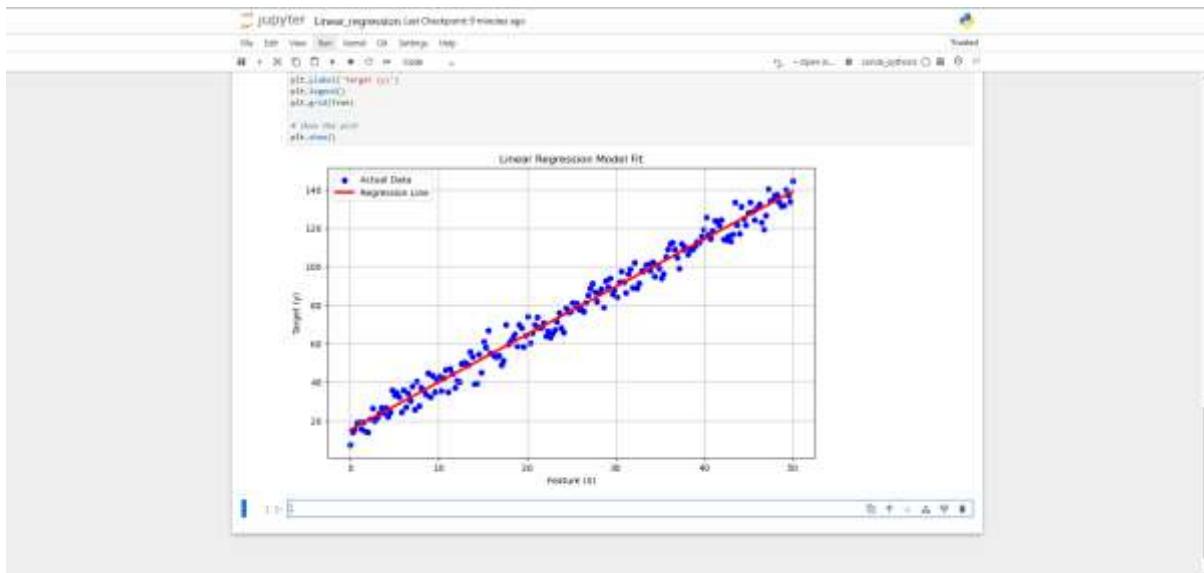
# Create a new instance of the model
model = LinearRegression()

# Fit the model to the data
model.fit(X_train, y_train)

# Predict values from the model for our training data
y_pred = model.predict(X_train)

print("Model training complete.\n")
print("Model intercept (b0):", model.intercept_, "\n")
print("Model coefficient (b1):", model.coef_[0], "\n")

print("Training score (r^2):", r2_score(y_train, y_pred))
print("Model intercept (b0):", b0)
print("Model coefficient (b1):", b1)
```



Conclusion: In this lab, we successfully utilized an Amazon SageMaker notebook to perform an end-to-end machine learning task. We began by generating a dataset and storing it in Amazon S3, a best practice for cloud-based ML workflows. We then trained a simple linear regression model directly within the notebook, evaluated its performance by calculating the Mean Squared Error, and created a clear visualization showing the relationship between the actual data and the model's predictions. This exercise demonstrates the power and convenience of using SageMaker as an integrated environment for data analysis and model building.

Practical 04

| |
|---|
| Student Name: Suhani Deepak Nemade |
| Date of Experiment: |
| Date of Submission: |
| PRN No: 20220802404 |

Title of lab: Implementing Data Preprocessing using SageMaker Notebooks: To perform data cleaning, transformation, and feature engineering using SageMaker notebooks.

Objective: The objective of this lab is to perform data cleaning, transformation, and feature engineering on the Titanic dataset using an Amazon SageMaker Notebook environment. The goal is to prepare the raw data into a clean, structured format suitable for training a machine learning model.

Tools:

- Personal Computer
- Web Browser
- Internet Connection

Software Used:

- Amazon Web Services (AWS) Management Console
- Amazon SageMaker (Notebook Instance)
- Jupyter Notebook
- Python (with pandas, numpy, and seaborn libraries)

Theory / Concept: Data preprocessing is a critical step in any machine learning pipeline. Raw datasets are often incomplete, inconsistent, and contain many features that may not be in a format suitable for modeling. This lab demonstrates the essential preprocessing techniques within Amazon SageMaker, a fully managed service that enables developers to build, train, and deploy machine learning models at scale.

We will use the well-known Titanic dataset, which contains information about passengers aboard the Titanic. Our task is to clean this data by handling missing values (imputation or removal), transform categorical data into a numerical format (one-hot encoding, label mapping), and engineer new features that might improve model performance.

Output:

1. Setup and Data Load: Screenshot of the Jupyter notebook cell importing pandas, numpy, and seaborn, and then loading the 'titanic' dataset.
2. Initial Exploration (head): Screenshot of the df_titanic.head() output showing the first few rows of the raw data.
3. Exploration (info): Screenshot of the df_titanic.info() output, highlighting the columns with missing values (e.g., age, deck).
4. Exploration (describe): Screenshot of the df_titanic.describe() output showing statistics for numerical columns.
5. Handling Missing Values: Screenshots of the code cells used to:
 - o Fill missing age values (e.g., with the median).
 - o Drop the deck column.
 - o Fill missing embarked values (e.g., with the mode).
6. Data Transformation: Screenshots of the code cells for:
 - o Mapping sex to numerical values (0 and 1).
 - o Applying one-hot encoding to embarked and pclass.
7. Feature Engineering: Screenshot of the code cell creating the family_size feature from sibsp and parch.
8. Final Data: Screenshot of the df_titanic.head() output after all preprocessing, showing the new, clean, and transformed columns.

The screenshot shows the Jupyter Notebook interface. At the top, there is a 'Create notebook instance' dialog box. In the main area, there are two code cells. The first cell contains the following code:`import numpy as np
import pandas as pd
import seaborn as sns`The second cell contains the following code and its output:`# Seaborn has a built-in function to load the Titanic dataset
It downloads it from an online source and returns a pandas DataFrame
df_titanic = sns.load_dataset('titanic')

That's all! The data is in your variable
df_titanic.head()`

The output of the second cell is a Pandas DataFrame with 8 rows and 12 columns. The columns are: survived, pclass, sex, age, sibsp, parch, fare, embarked, class, who, adult_male, deck, embark_town, alive, alone. The data includes information about passengers' survival status, class, gender, age, and travel details.

```

[14]: # Import the necessary libraries
# It download it from an online source and return a pandas DataFrame
df_titanic = sns.load_dataset("titanic")

# There is 1378 data in your variable.
df_titanic.head(5)

[14]:
   survived pclass sex age sibsp parch fare embarked class who alive alone
0         0     1   male 22.0    1    0  72.3833   S  Third  man   True  NaN  Southampton no  False
1         1     1   female 38.0    1    0  71.2833   C  First  woman  False  C   Cherbourg yes  False
2         1     1   female 28.0    0    0  75.5000   S  Third  woman  False  NaN  Southampton yes  True
3         1     1   female 35.0    1    0  32.3833   S  First  woman  False  C   Southampton yes  False
4         0     1   male 25.0    0    0  80.0000   S  Third  man   True  NaN  Southampton no  True

```



```

[15]: # Display the data types and count of non-null values for each column
print(df_titanic.info())

'pandas.core.frame.DataFrame' object>
RangeIndex: 891 entries, 0 to 890
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype  
--- 
 0   survived        891 non-null   int64  
 1   pclass          891 non-null   int64  
 2   sex             891 non-null   object  
 3   age             871 non-null   float64 
 4   sibsp           891 non-null   int64  
 5   parch           891 non-null   int64  
 6   fare             891 non-null   float64 
 7   embarked        891 non-null   object  
 8   class            891 non-null   category 
 9   who              891 non-null   object  
 10  adult_male      891 non-null   bool    
 11  deck             891 non-null   category 
 12  embark_town     891 non-null   object  
 13  alive            891 non-null   bool    
dtypes: bool(2), category(1), float64(3), int64(4), object(5)
memory usage: 88.7+ KB
None

```



```

[16]: # Let's do a statistical summary of the numerical columns
print(df_titanic.describe())

```

| | survived | pclass | age | sibsp | parch | fare |
|-------|------------|------------|------------|------------|------------|------------|
| count | 891.000000 | 891.000000 | 714.000000 | 891.000000 | 891.000000 | 891.000000 |
| mean | 0.313833 | 2.389542 | 31.239890 | 0.318394 | 0.318394 | 32.294288 |
| std | 0.488891 | 0.330872 | 13.326497 | 1.182757 | 0.388087 | 49.071129 |
| min | 0.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 25% | 0.000000 | 2.000000 | 28.125000 | 0.000000 | 0.000000 | 7.910400 |
| 50% | 0.000000 | 3.000000 | 32.000000 | 0.000000 | 0.000000 | 14.514280 |
| 75% | 1.000000 | 3.000000 | 38.000000 | 1.000000 | 0.000000 | 63.166660 |
| max | 1.000000 | 3.000000 | 80.000000 | 0.000000 | 0.000000 | 512.315000 |


```

[17]: # Check the dimensions of the dataset (rows, columns)
print("The dataset has {} rows and {} columns.".format(len(df_titanic), len(df_titanic.columns)))
The dataset has 891 rows and 13 columns.

```



```

[18]: # Drop all the null values
print(df_titanic.dropna())

```

| survived | pclass | sex | age | sibsp | parch | fare |
|----------|--------|--------|------|-------|-------|---------|
| 0 | 1 | male | 22.0 | 1 | 0 | 72.3833 |
| 1 | 1 | female | 38.0 | 1 | 0 | 71.2833 |
| 2 | 1 | female | 28.0 | 0 | 0 | 75.5 |
| 3 | 1 | female | 35.0 | 1 | 0 | 32.3833 |
| 4 | 0 | male | 25.0 | 0 | 0 | 80.0 |


```

[19]: # Calculate the median age
median_age = df_titanic['age'].median()
print("Median age: ", median_age)

# Fill missing "age" values with the median
df_titanic['age'].fillna(median_age, inplace=True)

```

Median age: 29.0

jupyter LAB4 Last Checkpoint 5 days ago

File Edit View Run Kernel Help Settings Help Trusted

In [1]:

```
# Calculate the median age
median_age = df_titanic['age'].median()
print("Median age: " + str(median_age))

# Fill missing 'age' values with the median
df_titanic['age'] = df_titanic['age'].fillna(median_age)
median_age = 28.0

In [2]:
```

Drop the 'deck' column
df_titanic.drop('deck', axis=1, inplace=True)

In [3]:

```
# Find the mode of the 'embarked' and 'embark_town' column

mode_embarked = df_titanic['embarked'].mode()[0]
print("Most common embarked port: " + mode_embarked)

mode_embark_town = df_titanic['embark_town'].mode()[0]
print("Most common embarked port: " + mode_embark_town)

# Fill missing values in 'embarked' and 'embark_town'
df_titanic['embarked'] = df_titanic['embarked'].fillna(mode_embarked)
df_titanic['embark_town'] = df_titanic['embark_town'].fillna(mode_embark_town)

most_common_embarked_port = 8
most_common_embarked_port = Southampton

# Drop 'age' column to numerical values
df_titanic['age'] = df_titanic['age'].apply(lambda x: 0 if np.isnan(x) else x)

In [4]:
```

at embryo header

jupyter LAB4 Last Checkpoint 5 days ago

File Edit View Run Kernel Help Settings Help Trusted

In [1]:

```
print("Most common embarked port: " + mode_embarked)

mode_embark_town = df_titanic['embark_town'].mode()[0]
print("Most common embarked port: " + mode_embark_town)

# Fill missing values in 'embarked' and 'embark_town'
df_titanic['embarked'] = df_titanic['embarked'].fillna(mode_embarked)
df_titanic['embark_town'] = df_titanic['embark_town'].fillna(mode_embark_town)

most_common_embarked_port = 8
most_common_embarked_port = Southampton

In [2]:
```

Drop 'age' column to numerical values
df_titanic['age'] = df_titanic['age'].apply(lambda x: 0 if np.isnan(x) else x)

In [3]:

```
# Perform one-hot encoding on 'embarked' and 'pclass'
df_titanic = pd.get_dummies(df_titanic, columns=['embarked', 'pclass'], drop_first=True)

# We can now drop columns that are redundant or not useful for modeling
df_titanic.drop(['who', 'adult_male', 'class', 'alive', 'embark_town'], axis=1, inplace=True)

In [4]:
```

Create a new feature for family size
df_titanic['family_size'] = df_titanic['sibsp'] + df_titanic['parch'] + 1

We can also create a binary feature 'is_alone'
df_titanic['is_alone'] = df_titanic['SibSp'] == 0 & df_titanic['Parch'] == 0, dtype=bool

Now we can drop the original 'sibsp' and 'parch' columns
df_titanic.drop(['sibsp', 'parch'], axis=1, inplace=True)

In [5]:

at titanic head()

jupyter LAB4 Last Checkpoint 5 days ago

File Edit View Run Kernel Help Settings Help Trusted

In [1]:

```
# Drop 'age' column to numerical values
df_titanic['age'] = df_titanic['age'].apply(lambda x: 0 if np.isnan(x) else x)

In [2]:
```

Perform one-hot encoding on 'embarked' and 'pclass'
df_titanic = pd.get_dummies(df_titanic, columns=['embarked', 'pclass'], drop_first=True)

We can now drop columns that are redundant or not useful for modeling
df_titanic.drop(['who', 'adult_male', 'class', 'alive', 'embark_town'], axis=1, inplace=True)

In [3]:

Create a new feature for family size
df_titanic['family_size'] = df_titanic['sibsp'] + df_titanic['parch'] + 1

We can also create a binary feature 'is_alone'
df_titanic['is_alone'] = df_titanic['SibSp'] == 0 & df_titanic['Parch'] == 0, dtype=bool

Now we can drop the original 'sibsp' and 'parch' columns
df_titanic.drop(['sibsp', 'parch'], axis=1, inplace=True)

In [4]:

df_titanic.head()

Conclusion: In this lab, we successfully utilized a SageMaker Notebook to perform essential data preprocessing on the Titanic dataset. We systematically addressed missing values, transformed categorical data into a machine-readable format, and engineered new features. This process highlights the importance of data preparation as a fundamental

step in the machine learning workflow. The Amazon SageMaker environment provided an interactive and powerful platform for executing these tasks efficiently. The resulting clean dataset is now ready for the next stage: model training and evaluation.

Practical 05

| |
|---|
| Student Name: Suhani Deepak Nemade |
| Date of Experiment: |
| Date of Submission: |
| PRN No: 20220802404 |

Title of lab: Hyperparameter Tuning for a Classification Model: To optimize the performance of a classification model by tuning its hyperparameters using SageMaker.

Objective: The objective of this lab is to implement end-to-end model training, hyperparameter optimization, and deployment using Amazon SageMaker. The experiment demonstrates how to preprocess data, train a TensorFlow model, automatically tune hyperparameters using SageMaker's Hyperparameter Tuner, and deploy the best-performing model as a real-time prediction endpoint.

Tools:

- Personal Computer
- Web Browser
- Internet Connection

Software Used:

- Amazon Web Services (AWS) Management Console
- Amazon SageMaker (Notebook Instance)
- Jupyter Notebook
- Python (with pandas, numpy, tensorflow, and sagemaker libraries)

Theory / Concept: Amazon SageMaker is a fully managed machine learning service that provides the tools and infrastructure needed to build, train, and deploy machine learning models efficiently. In this lab, we explored the complete lifecycle of developing a model using the Iris dataset, from data preprocessing to model deployment. The Iris dataset is a classic example in machine learning, containing measurements of flower features such as sepal length, sepal width, petal length, and petal width, used to classify iris flowers into three species. The process began with data preprocessing, which included cleaning column names, scaling numerical features, and encoding categorical labels. After preparing the data, we used TensorFlow within SageMaker to define a neural network

model, and then applied hyperparameter tuning using SageMaker's Hyperparameter Tuner to automatically search for the best model configuration by varying learning rate and hidden layer size. Once the optimal model was identified, it was deployed as an endpoint using SageMaker's hosting service, allowing real-time inference through API calls. This lab demonstrates how SageMaker streamlines the workflow of data preparation, model training, optimization, and deployment in a single, integrated environment, making it an essential tool for modern machine learning development.

Output:

- SageMaker session and S3 bucket setup.
 - Preprocessing and train/validation data upload.
 - Training script (train.py) content.
 - Hyperparameter tuner configuration.
 - Tuning job completion output with results table.
 - Deployment success message showing endpoint name.
 - Test prediction output (showing Iris-versicolor).
 - Endpoint deletion confirmation.


```

jupyter Untitled Last Checkpoint: 27 minutes ago
File Edit View Run Kernel Help Settings Trusted
B + X C O Code
various layers, Dense, Softmax, activation='softmax')
]

# Optimizer. The learning rate is a hyperparameter
optimizer = keras.optimizers.Adam(learning_rate=learning_rate)

# Compile the model
model.compile(
    optimizer=optimizer,
    loss='categorical_crossentropy', # Use this loss for integer labels.
    metrics=['accuracy'])
)
return model

# --- 3. Data Cleaning and Parsing logic ---
if __name__ == '__main__':
    # --- 4. Parse arguments ---
    # Hyperparameter parser hyperparameters and paths as command-line arguments
    parser = argparse.ArgumentParser()

    # Hyperparameters
    parser.add_argument('--hidden-units', type=int, default=10)
    parser.add_argument('--learning-rate', type=float, default=0.01)
    parser.add_argument('--epochs', type=int, default=10)
    parser.add_argument('--batch-size', type=int, default=10)

    # Logfile-like environment variables
    parser.add_argument('--model-dir', type=str, default=os.environ.get('SM_MODEL_DIR'))
    parser.add_argument('--train', type=str, default=os.environ.get('SM_CHANNEL_TRAIN'))
    parser.add_argument('--validation', type=str, default=os.environ.get('SM_CHANNEL_VALIDATION'))

# --- 5. Load Data ---
X_train, y_train = load_data(args.train)
X_val, y_val = load_data(args.validation)

# --- 6. Build Model ---
model = build_model(args.hidden_units, args.learning_rate)

# --- 7. Train Model ---
print("Starting model training...")
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=args.epochs,
    batch_size=args.batch_size,
    verbose=1 # Show logs
)

# --- 8. EVALUATE! Print the objective metric ---
# The launcher API job will read this line from the logs to find the score.
# It WILL match the "regress" as defined later.
# We use the final validation accuracy from the training history.
final_val_accuracy = history.history['val_accuracy'][-1]
print(f"val_accuracy: {final_val_accuracy}")

# --- 9. Save Model ---
# Save the model to TensorFlow's standardized format
model.save(experiment_dir, args.model_dir, '1')
print("Model saved!")

# Writing training

```

(11): `from sagemaker.tensorflow import TensorFlow`

```

jupyter Untitled Last Checkpoint: 28 minutes ago
File Edit View Run Kernel Help Settings Trusted
B + X C O Code
model.save(experiment_dir, args.model_dir, '1')
print("Model saved!")

Writing train.py

```

(11): `from sagemaker.tensorflow import TensorFlow
from sagemaker.tuner import IntegerParameter,
 ContinuousParameter,
 HyperparameterTuner,`

```

# --- 10. Define the Metric ---
# This tells Sagemaker what to look for in the logs.
# It MUST match the metric defined in training.
metric_definitions = [
    {
        'Name': 'val_accuracy',
        'Regex': 'val_accuracy: ([0-9.]+)'
    }
]

# --- 11. Create the TensorFlow Estimator ---
# This is the blueprint for a single training job.
estimator = TensorFlow(
    entry_point='train.py', # Our script.
    role=sagemaker.get_execution_role(),
    instance_count=1, # Use 1 instance per job.
    instance_type='ml.m4.xlarge',
    framework_version='2.3.1',
    py_version='py36',
    # ... THIS IS THE CRITICAL PART ...
)
```

```

jupyter Untitled Last Checkpoint: 28 minutes ago
File Edit View Run Kernel Cell Settings Help
B + X D * Code
initial_instance_count=1,
instance_type='ml.m4.large',
framework_version='1.12',
py_version='py36',
)
# *** NOTE TO THE ESTIMATOR: FIX ***
# We are passing the metric definition to the estimator.
metric_definitions=metric_definitions,
)
# --- d. Define hyperparameter ranges to search
hyperparameter_ranges = {
    'learning_rate': ContinuousParameter(0.0001, 0.01),
    'hidden_units': IntegerParameter(10, 50),
}
# --- e. Create the HyperparameterTuner object
# This tuner will now inherit the metric_definitions from the estimator
tuner = HyperparameterTuner(
    estimator_fn=estimator,
    objective_metric_name='val_accuracy', # This name was defined in defied metric
    objective_type='Maximize',
    hyperparameter_ranges=hyperparameter_ranges,
    metric_definitions=metric_definitions,
    max_jobs=10,
    max_parallel_jobs=2
)
print("Tuner object created successfully with metrics. Ready for Step 4!")
Tuner object created successfully with metrics. Ready for Step 4.

# --- f. Launch the Tuning Job ---
jupyter Untitled Last Checkpoint: 28 minutes ago
File Edit View Run Kernel Cell Settings Help
B + X D * Code
tuner.object_createdSuccessfullyWithMetrics=True
# Create all input objects:
et_train = BigQueryInputs.TrainingInput(et_train, content_type='text/csv'),
et_validation = BigQueryInputs.ValidationInput(et_validation, content_type='text/csv'),
)
print("Starting Hyperparameter Tuning Job.... This will take 10-15 minutes...")
tuner.fit(['train']: et_train, 'validation': et_validation, validation=True)
print("Tuning Job complete.")

# Get the name of the best job
best_job_name = tuner.best_training_job()
print(f"\n\nBest Training Job Name: {best_job_name}\n")
# --- g. Analyze results
# Get a summary of all tuning jobs
results = tuner.analytics().getframe()
print("All Tuning Job Results:")
print(results.sort_values('FinalObjectValue', ascending=False))

# Get the name of the best job
best_job_name = tuner.best_training_job()
print(f"\n\nBest Training Job Name: {best_job_name}\n")
jupyter Untitled Last Checkpoint: 29 minutes ago
File Edit View Run Kernel Cell Settings Help
B + X D * Code
initial_instance_count=1,
instance_type='ml.m4.large', # --> changed from ml.t2.medium
)
print(f"\n\nTraining Job '{estimator.endpoint_name}' deployed successfully!")

# --- h. Actual: Test the Endpoint ---
import tensorflow as tf
# Prepare test input (object shape based on your model)
test_data = np.random.rand(5, 4) # e.g., 4 features if your model expects that
print("Creating test prediction request...")
prediction = predictor.predict(test_data)
print("Prediction result:", prediction)

all_tuning_job_results:
  hidden_units learning_rate TrainingJobStatus
0 44.0 0.005794 tensorflow-training-252095-1237-000-00043094 Completed
1 57.0 0.000448 tensorflow-training-252095-1237-000-796e4294 Completed
2 32.0 0.000139 tensorflow-training-252095-1237-000-15009711 Completed
3 45.0 0.000019 tensorflow-training-252095-1237-000-46842895 Completed
4 31.0 0.000148 tensorflow-training-252095-1237-000-0007241 Completed
5 27.0 0.000034 tensorflow-training-252095-1237-000-4fe05526 Completed

  TrainingJobStatus FinalObjectValue TrainingStartTime
0 Completed 1.000000 2025-11-05 22:42:13+00:00
1 Completed 1.000000 2025-11-05 22:49:57+00:00
2 Completed 8.000000 2025-11-05 22:42:09+00:00
3 Completed 8.000000 2025-11-05 22:48:50+00:00
4 Completed 8.444447 2025-11-05 22:37:59+00:00
5 Completed 8.266667 2025-11-05 22:37:57+00:00

```

The image shows two vertically stacked Jupyter Notebook interfaces. Both notebooks have the title 'Untitled' and a 'Last Checkpoint: 10 minutes ago' message.

Top Notebook:

```
# --- 1. Load the dataset ...
# Let's take one sample from our collection, set [1, len()]
# Note: we must use the data in the same scaled format
sample = X_val[0]
print("Test sample (scaled):", sample)

# The TensorFlow endpoint expects a specific JSON format
payload = {'instances': [sample.tolist()]}

# Make the prediction
response = predictor.predict(payload)

# The output 'predictions' is a list of probabilities for each class {0, 1, 2}
predictions = response['predictions'][0]
predicted_class = np.argmax(predictions)

# Insert the class index back to the original list
predicted_label = encoder.inverse_transform([predicted_class])[0]

print("Predictions probabilities: ", predictions)
print("Predicted class: ", predicted_class)
print("Predicted species: ", predicted_label)

Test sample (scaled): [ 0.31099975 -0.58778353  0.55538983  0.88175297]
Prediction probabilities: [0.00009488618, 0.941276705, 0.0142397987]
Predicted class: 1
Predicted species: Iris-versicolor
# --- 4. 42208 IP
```

Bottom Notebook:

```
print("Predict class: ", predicted_class)
print("Predicted species: ", predicted_label)

Test sample (scaled): [ 0.31099975 -0.58778353  0.55538983  0.88175297]
Prediction probabilities: [0.00009488618, 0.941276705, 0.0142397987]
Predicted class: 1
Predicted species: Iris-versicolor
# --- 4. 42208 IP

print("Deleting endpoint: ", predictor.endpoint_name)
predictor.delete_endpoint()
print("Endpoint deleted.")

# --- Now, you must manually stop the notebook ...
# 1. Go to the top-left corner.
# 2. Click on "Notebook" -> "Notebook Instances".
# 3. Select your notebook.
# 4. Click "Actions" -> "Stop".
# 5. Once it's stopped, go over click "Actions" -> "Delete" to remove it.
Deleting endpoint: tensorflow-training-211085-12717-000-798e4385...
Endpoint deleted.
```

Conclusion: In this lab, we successfully implemented automatic hyperparameter tuning and model deployment using AWS SageMaker on the Iris dataset. We performed complete preprocessing, trained a TensorFlow model, and used SageMaker's Hyperparameter Tuner to find the optimal values of learning-rate and hidden-units for maximum validation accuracy. Finally, the best model was deployed as a real-time inference endpoint and successfully predicted the species of a flower sample. This experiment demonstrates the power of SageMaker for automating the model development cycle — from data preprocessing to deployment — in a scalable and managed environment.

Practical 06

| |
|---|
| Student Name: Suhani Deepak Nemade |
| Date of Experiment: |
| Date of Submission: |
| PRN No: 20220802404 |

Title of lab: Sentiment Analysis using Amazon Comprehend: To use Amazon Comprehend to perform sentiment analysis on text data.

Objective: To use the Amazon Comprehend NLP service to perform real-time sentiment analysis on provided text data and interpret the results.

Tools:

- Personal Computer
- Web Browser
- Internet Connection

Software Used:

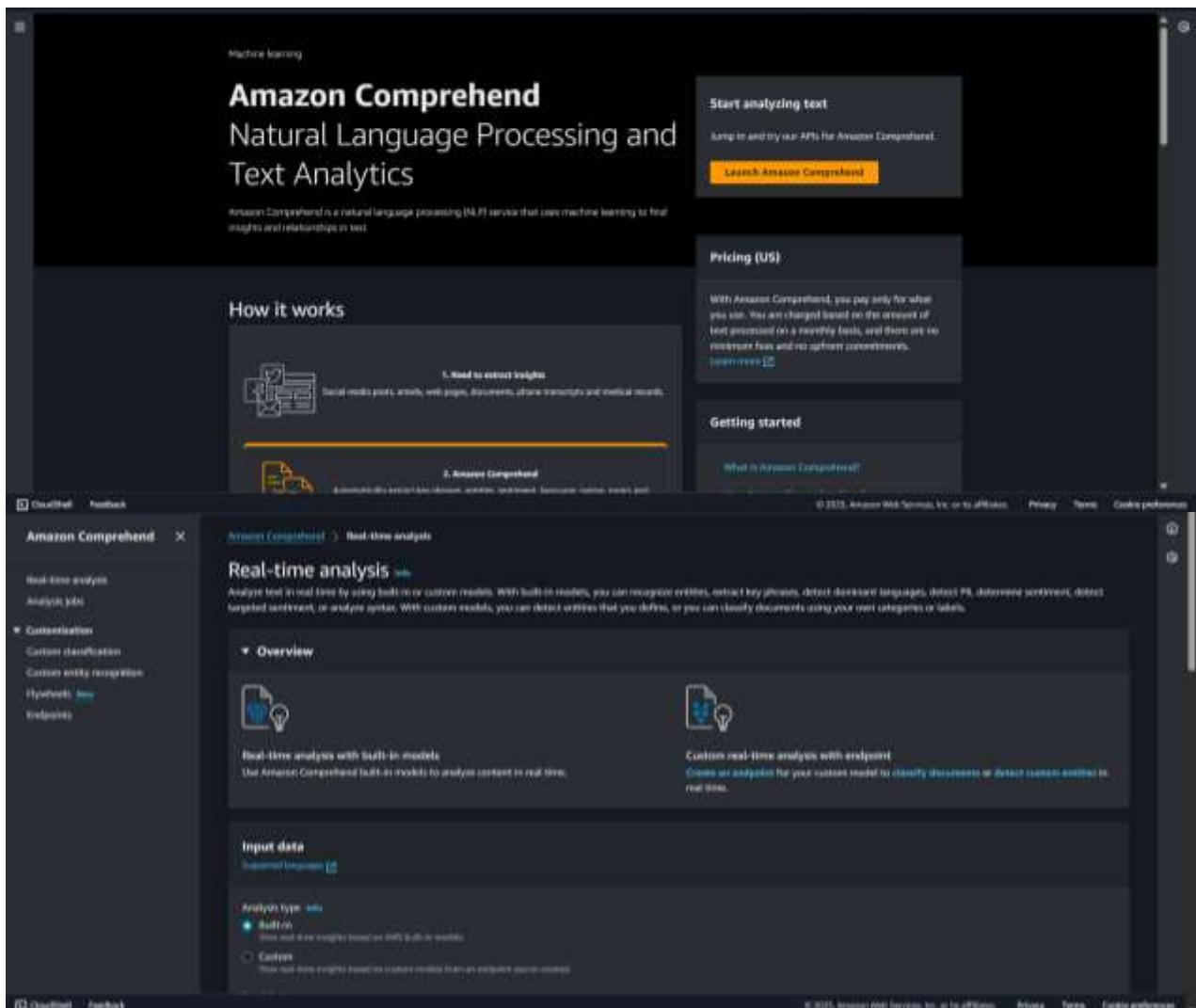
- Amazon Web Services (AWS) Management Console
- Amazon Comprehend

Theory / Concept: Amazon Comprehend is a natural language processing (NLP) service from Amazon Web Services that uses machine learning to uncover valuable insights and relationships in text. It is a fully managed service, meaning you don't need to train your own models to use it.

One of its key features is **Sentiment Analysis**, which is the process of identifying and categorizing the emotional tone expressed in a piece of text. Amazon Comprehend analyzes text and returns the dominant sentiment as POSITIVE, NEGATIVE, NEUTRAL, or MIXED, along with confidence scores for each. This is extremely useful for understanding customer feedback, social media comments, or any other text-based data.

Output:

1. Amazon Comprehend Console: A screenshot of the main Amazon Comprehend service page in the AWS console.
 2. Input Text: A screenshot of the "Built-in analysis" section, showing the default text provided in the "Input text" box.
 3. Analysis Results: A screenshot of the output after clicking "Analyze." This should clearly show the "Sentiment" tab, with the resulting sentiment (e.g., "MIXED") and the confidence scores for each category (Positive, Negative, Neutral, Mixed).



Input text:

Hello Zhang Wei, I am John. Your AnyCompany Financial Services, LLC credit card account 1111-0000-1111-0000 has a minimum payment of \$24.53 that is due by July 31st. Based on your display settings, we will withdraw your payment on the due date from your bank account number 0000001111 with the routing number 0000000000.

Customer feedback for Sunshine Spa, 123 Main St, Anywhere. Send comments to Alice at sunsp@sunsp.com.

I enjoyed visiting the spa. It was very comfortable but it was also very expensive. The amenities were ok but the service made the spa a great experience.

Insights

Analyzed text

Your entity settings, we will withdraw your payment on the due date from your bank account number 0000001111 with the routing number 0000000000.

Customer feedback for Sunshine Spa, 123 Main St, Anywhere. Send comments to Alice at sunsp@sunsp.com.

I enjoyed visiting the spa. It was very comfortable but it was also very expensive. The amenities were ok but the service made the spa a great experience.

Results

| Entity | Type | Confidence |
|------------------------------------|--------------|------------|
| Zhang Wei | Person | 0.99+ |
| John | Person | 0.99+ |
| AnyCompany Financial Services, LLC | Organization | 0.88+ |
| 1111-0000-1111-0000 | Other | 0.99+ |
| \$24.53 | Quantity | 0.99+ |
| July 31st | Date | 0.99+ |
| 0000001111 | Other | 0.98 |
| 0000000000 | Other | 0.97 |
| Sunshine Spa | Organization | 0.88 |

Conclusion: In this lab, we successfully utilized a SageMaker Notebook to perform essential data preprocessing on the Titanic dataset. We systematically addressed missing values, transformed categorical data into a machine-readable format, and engineered new features. This process highlights the importance of data preparation as a fundamental step in the machine learning workflow. The Amazon SageMaker environment provided an interactive and powerful platform for executing these tasks efficiently. The resulting clean dataset is now ready for the next stage: model training and evaluation.

Practical 07

| |
|---|
| Student Name: Suhani Deepak Nemade |
| Date of Experiment: |
| Date of Submission: |
| PRN No: 20220802404 |

Title of lab: Building a Simple Conversational Bot with Amazon Lex: To design and implement a simple conversational bot using Amazon Lex.

Objective: The objective of this lab is to gain hands-on experience with Amazon Web Services (AWS) Amazon Lex. Students will learn to design, build, and test a simple, "traditional" (intent-based) conversational bot.

Tools:

- Personal Computer
- Web Browser
- Internet Connection

Software Used:

- Amazon Web Services (AWS) Management Console
- Amazon LEX

Theory / Concept: Amazon Lex is an AWS service for building conversational interfaces (chatbots) using both voice and text. This lab uses the **Traditional** creation method, which provides direct, manual control over the bot's logic.

The core components we built are:

1. **Intent:** An intent represents the user's primary goal. For this lab, a single intent, **BookTrip**, was created to handle a user's request to book a hotel.
2. **Sample Utterances:** These are training phrases that the user might type to trigger an intent (e.g., "I want to book a trip"). The bot's Natural Language Understanding (NLU) model is trained on these phrases to recognize the user's goal.
3. **Slots:** Slots are the individual pieces of information the bot must collect from the user to fulfill the intent, like variables in a form. We configured three required slots:
 - **Location** (Type: AMAZON.City)

- **CheckInDate** (Type: AMAZON.Date)
 - **Nights** (Type: AMAZON.Number)
4. **Slot Prompts:** For each required slot, a prompt (a question) is written. The bot uses this question to "elicit" (ask for) the information. For example, the prompt for the Location slot was, "What city are you booking in?"
 5. **Fulfillment:** This is the final action taken after all required slots are successfully collected. For this simple lab, no external code (like AWS Lambda) was used. Instead, the bot was configured to provide a simple "**Success Response**" message, which dynamically inserted the collected slot values:
 - OK, I am booking your {Nights} night trip to {Location} starting {CheckInDate}.
 6. **Build & Test:** The "Build" step trains the AI model. The "Test" step opens a chat window to interact with the bot and verify its logic.

Output:

1. Bot Creation: Screenshot of the "Create bot" page, showing the "Traditional" and "Create a blank bot" options selected, with the Bot name set to BookTrip. (*Insert your screenshot of the bot creation settings here.*)
2. Intent & Utterances: Screenshot of the BookTrip intent page, showing the list of "Sample utterances" ("I want to book a trip", "Book a trip", "Can I reserve a room?", etc.). (*Insert your screenshot of the "Sample Utterances" section here.*)
3. Slots Configuration: Screenshot of the "Slots" section *after* all three slots (Location, CheckInDate, Nights) have been added, showing their names and slot types. (*Insert your screenshot of the completed "Slots" section here.*)
4. Fulfillment Configuration: Screenshot of the "Fulfillment" section, showing the "On successful fulfillment" message box with the final text: OK, I am booking your {Nights} night trip to {Location} starting {CheckInDate}. (*Insert your screenshot of the "Fulfillment" message box here.*)
5. Build & Test: Screenshot of the successful "Test Draft version" window showing the complete conversation, from the initial utterance to the final fulfillment response. (*Insert your screenshot from the Test Console here - image_1a92ff.png*)
6. Bot Deletion: Screenshot of the main "Bots" list page, with the BookTrip bot selected and the "Delete" action visible. (*Insert your screenshot of the bot list - image_0dd85d.png*)

7. IAM Role Deletion: Screenshot of the IAM "Roles" page, showing the search for the bot's role and the "0 matches" result, confirming the role (or search) is gone. (*Insert your screenshot of the IAM Roles page - image_1a9a82.png*)

Bots - [0] [Info](#)

No bots found [Create bot](#)

Configure bot settings [Info](#)

Creation method

Create a blank bot Create a blank bot with no preexisting code, language, intents, and slot types.

Start with an example An example bot that processes common messages, intents, and slot types. You can change these at any time.

Start with transcripts Automatically generates a bot from common transcripts and slot types. This will capture any Amazon Alexa or custom intent you say.

Bot configuration

Bot name: BotWith

Description (optional): A simple bot for my AI test

IAM permissions [Info](#)

IAM roles are used to define which services your bot can access.

Role: awsLambdaBasicExecutionRole

Select a role that defines permission for your bot. To create a custom role, use the AWS console.

Create a role with basic Amazon Lex permissions

Use an existing role

Creating a role takes a few minutes. Don't delete the role or edit the trust or permissions policies in this role until we've finished creating it.

New role: Amazon Lex creates a custom role with permissions to update to Amazon CloudWatch Log.

Bot error logging [Info](#)

Output unstructured errors on Lex logs.

Error logs: Enabled Disabled [Learn more about error logs](#)

Bot error logging [Info](#)
Billing unexpected issues on bot logs.

Error log:

- Disabled
- Enabled

[Learn more about error logs](#) [Edit](#)

[Go to error logs](#) [Logs](#)

Children's Online Privacy Protection Act (COPPA) [Info](#)

Is use of your bot subject to the Children's Online Privacy Protection Act (COPPA)?

- Yes
- No

[Learn more about COPPA](#) [Edit](#)

[Go to COPPA logs](#) [Logs](#)

Idle session timeout

You can configure how long a session is maintained when the user does not provide any input and the session is idle. Amazon Lex retains context information until a session ends.

Session timeout:

6 minutes

By default, session timeout is 5 minutes, but you can specify any duration between 1 and 1,440 minutes (24 hours).

[Edit](#)

Children's Online Privacy Protection Act (COPPA) [Info](#)

Is use of your bot subject to the Children's Online Privacy Protection Act (COPPA)?

- Yes
- No

[Learn more about COPPA](#) [Edit](#)

[Go to COPPA logs](#) [Logs](#)

Idle session timeout

You can configure how long a session is maintained when the user does not provide any input and the session is idle. Amazon Lex retains context information until a session ends.

Session timeout:

6 minutes

By default, session timeout is 5 minutes, but you can specify any duration between 1 and 1,440 minutes (24 hours).

[Edit](#)

+ Advanced settings - optional [Info](#)

[Cancel](#) [Next](#)

Add language to bot [Info](#)

Step 1: Configure bot settings

Step 2: Add languages

▼ Language: English (US)

Select language:

English (US)

Description - optional

Maximum 250 characters

Voice interaction:

The text to speech voice that your bot uses to interact with users.

Dora the Explorer

Voice sample:

Hello, my name is Dora! Let me know how I can assist you.

Play

Intent classification confidence score threshold:

0.40

Min 0.00, max 1.00

[Cancel](#) [Add another language](#) [Done](#)

© 2018, Amazon Web Services, Inc. or its affiliates. [Privacy](#) [Terms](#) [Cookie preferences](#)

The screenshot shows the Amazon Lex console interface with three tabs open:

- Intent details**: Shows the intent name "BookTrip" and its description "Intent to book a hotel trip".
- Contexts - optional**: Shows input contexts "Chinese contexts" and output contexts "Chinese contexts".
- Sample utterances**: Displays sample utterances such as "I want to book a flight to New York" and "I need to book a flight".

Amazon Lex X

Draft version English (US) Next

Initial response (optional)
You can provide messages to acknowledge the user's initial request. You can also configure the next step in the conversation and branch based on conditions.

► Response to a flight booking (the user's request)
Message

► Starts (1) optional
Information that you want to send first. This can include a greeting or a reminder for the user to respond before the bot continues. It's typically hidden from users.

Greet

Add start

► Prompts for slot: Location Slot type: `OPTIONAL`
Message: What city are you departing from?

► Prompts for slot: CheckInDate Slot type: `OPTIONAL`
Message: When does your flight leave?

► Prompts for slot: FlightRights Slot type: `OPTIONAL`
Message: What are your flight rights?

Add start

Edit Visual builder Test Save intent

Amazon Lex X

Draft version English (US) Next

Confirmation (optional)
Prompts the user to clarify whether the user wants to book the order or cancel it.

► Wants to confirm the intent Response sent when the user declines the intent
Message: Decline

Fulfillment (optional)
Run a Lambda function to fulfill the user's slot value in case of an error when its resolved.

► On successful fulfillment Message: All set! Let me book your flight right now! Re-type of failure: Message: ...

► On unsuccessful fulfillment Message: OK, I am booking your flight right now to [Location] starting ...

In case of failure Message: Something went wrong

Advanced options Configure metrics, reduce and increase responses.

Edit Visual builder Test Save intent

Amazon Lex X

Draft version English (US) In fact: Round Trip. If your language contains several source slot types, the build might take longer to complete. Next

Advanced options Configure metrics, reduce and increase responses.

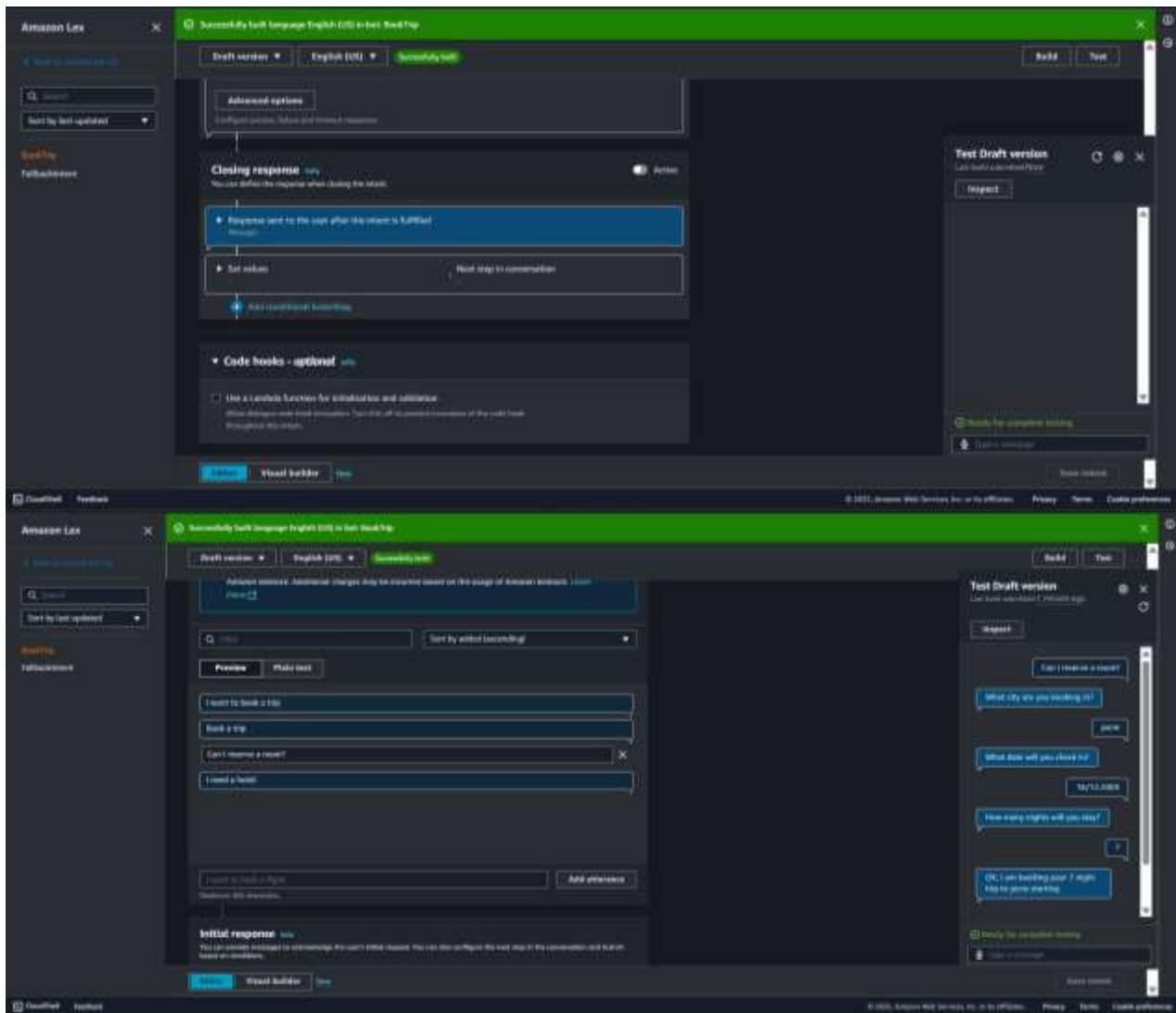
Closing response (optional)
You can define the message when closing the intent.

► Prompts user to the user after the intent is fulfilled Message: Thank you!

► End session Next step in conversation
Message: End of conversation

► Code hooks - optional Use a Lambda function for initialization and validation.
More information: Code hook: Turn this off if you don't want to use any of the code hooks throughout this intent.

Edit Visual builder Test Save intent



Conclusion: In this lab, we successfully utilized the Amazon Lex V2 console to design, build, and test a functional conversational bot. We learned that the core logic of a bot is a flow of **Intents** (goals) and **SLOTS** (data), connected by **Sample Utterances** and **Prompts**. This exercise shows that an intelligent chatbot can be created rapidly without any custom code.

Finally, the lab reinforced the importance of proper AWS resource management by performing a full cleanup (deleting the Lex Bot and its associated IAM Role) to ensure no future costs are incurred. The resulting bot correctly handled the user's request and fulfilled the intent as designed.

Practical 08

| |
|---|
| Student Name: Suhani Deepak Nemade |
| Date of Experiment: |
| Date of Submission: |
| PRN No: 20220802404 |

Title of lab: Object Detection using Amazon Rekognition: To use Amazon Rekognition to perform object detection in images.

Objective: The objective of this lab is to use the Amazon Rekognition service to analyze an image and identify objects, concepts, and scenes within it. The goal is to understand how Rekognition processes visual data and to interpret the resulting labels and confidence scores returned by the service.

Tools:

- Personal Computer
- Web Browser
- Internet Connection
- A sample image for uploading (e.g., a photo from your phone or a file from the internet)

Software Used:

- Amazon Web Services (AWS) Management Console
- Amazon Rekognition

Theory / Concept: Object detection is a computer vision task that identifies and locates objects within an image or video. Amazon Rekognition is a fully managed, deep-learning-based visual analysis service that automates this process. It can detect objects, scenes, activities, faces, text, and more.

Instead of requiring users to build and train their own complex machine learning models, Rekognition provides a simple API and a web-based demo console to get results quickly.

In this lab, we will use the '**Label detection**' demo feature. This feature analyzes an image and returns a list of 'labels' (e.g., 'Car,' 'Building,' 'Person') along with a 'confidence score' (e.g., 99.8%) indicating how certain the model is about that label. For labels that are physical objects, it can also provide 'bounding boxes' to show the exact location of the detected object in the image. Labels that describe the whole scene (e.g., 'Indoors,' 'Architecture') do not receive bounding boxes.

Output:

1. Rekognition Service Home: Screenshot of the main Amazon Rekognition service page after logging into the AWS console.
2. Label Detection Demo Interface: Screenshot of the "Label detection" demo page, showing the sample image and the "Use your own image" upload box.
3. Image Upload and Analysis Results: Screenshot after uploading your custom image. This screenshot must clearly show your image on the left and the full list of Results (all labels and their confidence scores) on the right.
4. Bounding Box Visualization: Screenshot showing the same uploaded image, but with a specific *object label* (like "Furniture," "Chair," or "Person") selected from the results list, and the corresponding blue bounding box clearly visible on the image.

The screenshots illustrate the Amazon Rekognition service home page and its label detection demo interface.

Main Service Home (Top Screenshot):

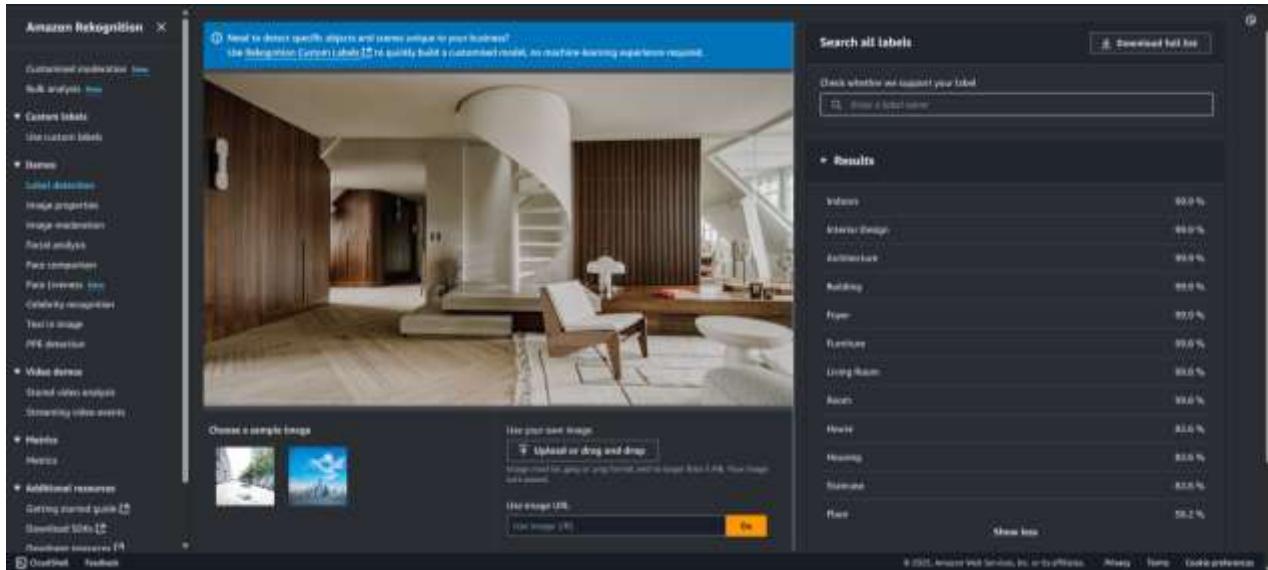
- Leverage proven image and video analysis:** Describes how Rekognition's high-quality image and video analysis can help build your own customized ML model to detect objects and scenes unique to your business, using your own training data.
- Detect objects unique to your business:** Details how Amazon Rekognition Custom Labels can help you quickly build your own customized ML model to detect objects and scenes unique to your business, directly by introducing your own training data. It mentions a guided experience, no machine learning experience required.
- Integrated with AWS Services:** Notes that Amazon Rekognition is designed to work seamlessly with other AWS services. It integrates directly with Amazon S3 and AWS Lambda so you can build scalable, affordable, and reliable visual analysis applications. It also mentions supporting image and video stored in Amazon S3 without AWS Lambda or glue. It includes a link to the Amazon Rekognition Video Streamer.

Label Detection Demo Interface (Bottom Screenshot):

- Feature Spotlight:** Analyze Video and Extract Rich Metadata: Learn how to use the video analysis features in Amazon Rekognition. [Learn more](#)
- Deep Dive on Amazon Rekognition:** Study and investigate image and video analysis in applications. [Learn more](#)
- Learning content:**
 - Worker Safety with AWS:** Build a worker safety application using AWS DeepRacer and Amazon Rekognition. [Learn more](#)
 - Amazon Rekognition Developer Guide:** Detailed instructions on how to use Amazon Rekognition. [Learn more](#)

The demo interface includes a sidebar with navigation links for various Rekognition services like Custom Labels, Detect Labels, Detect Objects, and Detect Scenes. The main area shows a sample image of a person on a skateboard, an "Upload or drag and drop" button, and a "Use image URL" input field. On the right, a results table lists detected labels with their confidence scores:

| Label | Confidence Score |
|--------------|------------------|
| Neighborhood | 99.9% |
| City | 99.9% |
| Road | 99.9% |
| Skateboard | 99.9% |
| Person | 99.7% |



Conclusion: In this lab, we successfully utilized the Amazon Rekognition service to perform object and scene detection on a custom image. We explored the 'Label detection' demo feature to instantly analyze visual content without writing any code or provisioning any infrastructure. This process highlights the power and accessibility of pre-trained AI models for complex computer vision tasks. The Amazon Rekognition console provided an interactive and intuitive platform for understanding how AI can interpret images. We were able to observe the detected labels, their confidence scores, and the bounding boxes that locate objects, successfully demonstrating the core capabilities of the service.