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

D Y PATIL
INTERNATIONAL
UNIVERSITY
AKURDI PUNE

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 environ ments.
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 Learn ing 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 platfor

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 beforeleaving 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. From	Page No. To	Sign of Teacher	Remarks*
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.

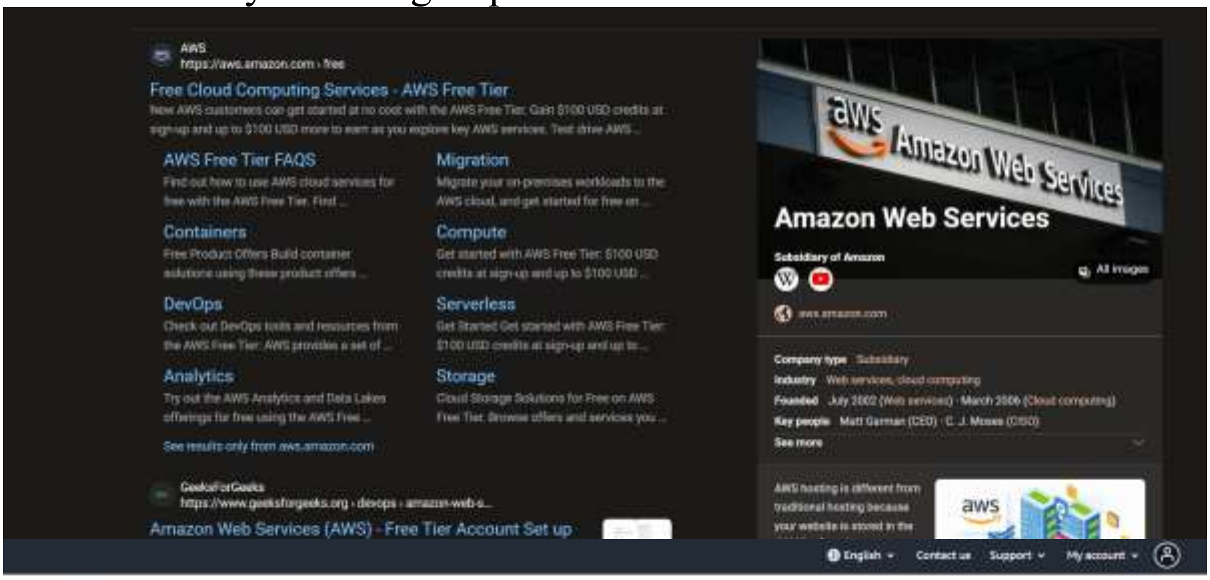
Tools:
- Personal Computer
- Web Browser
- Internet Connection

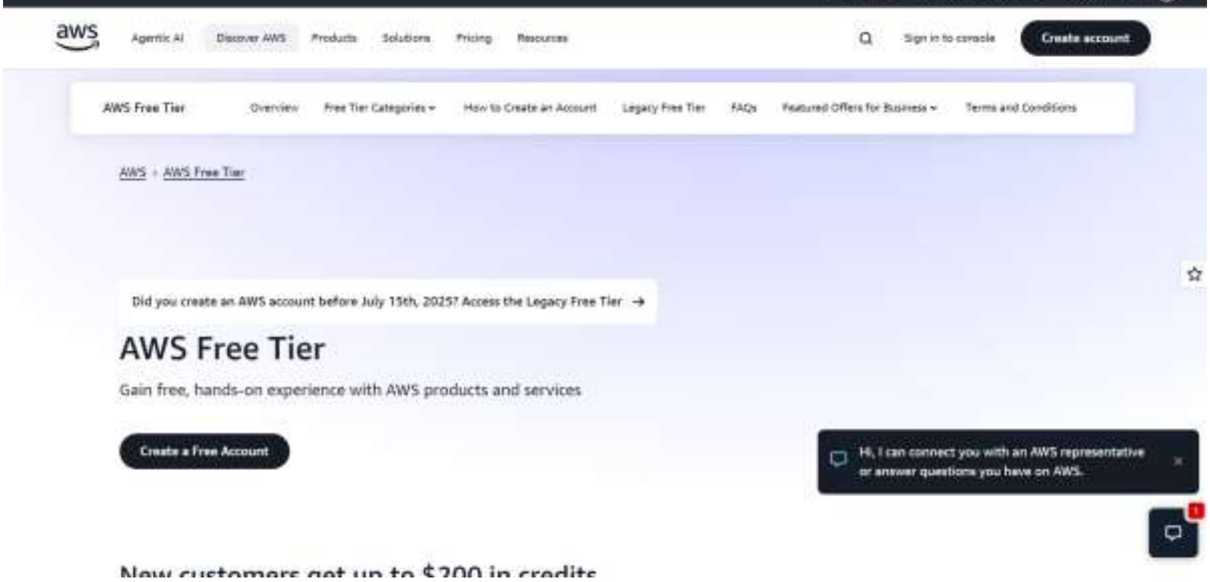
Software Used:
- Amazon Web Services (AWS) Management Console

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.





Sign up for AWS

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.

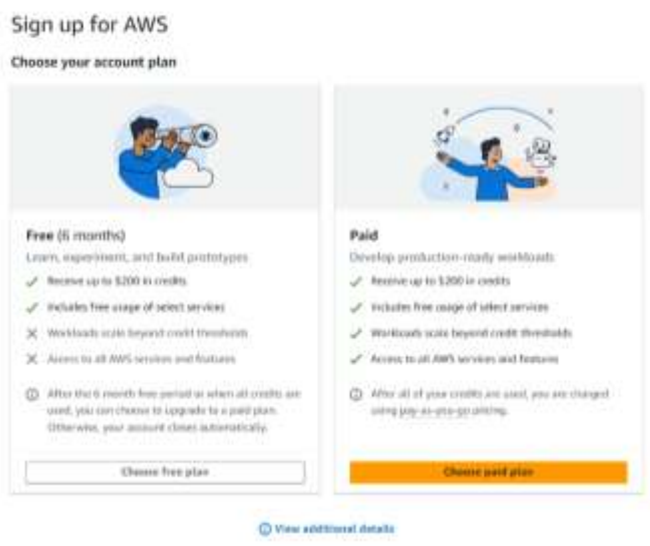
Verification code

Verify

Resend Code 55

Didn't get the code?

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



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

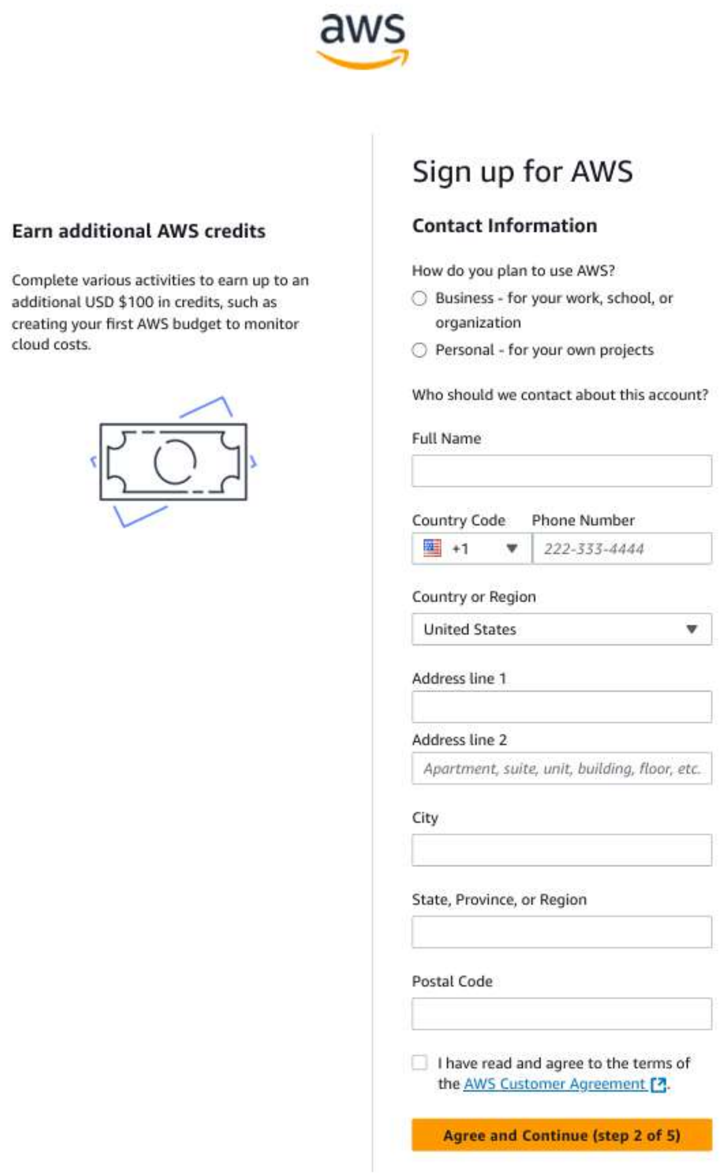
Develop production-ready workloads

- ✓ 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



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.

Sign up for AWS

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: +1 | Phone Number: 222-333-4444

Country or Region: United States

Address line 1

Address line 2: Apartment, suite, unit, building, floor, etc.

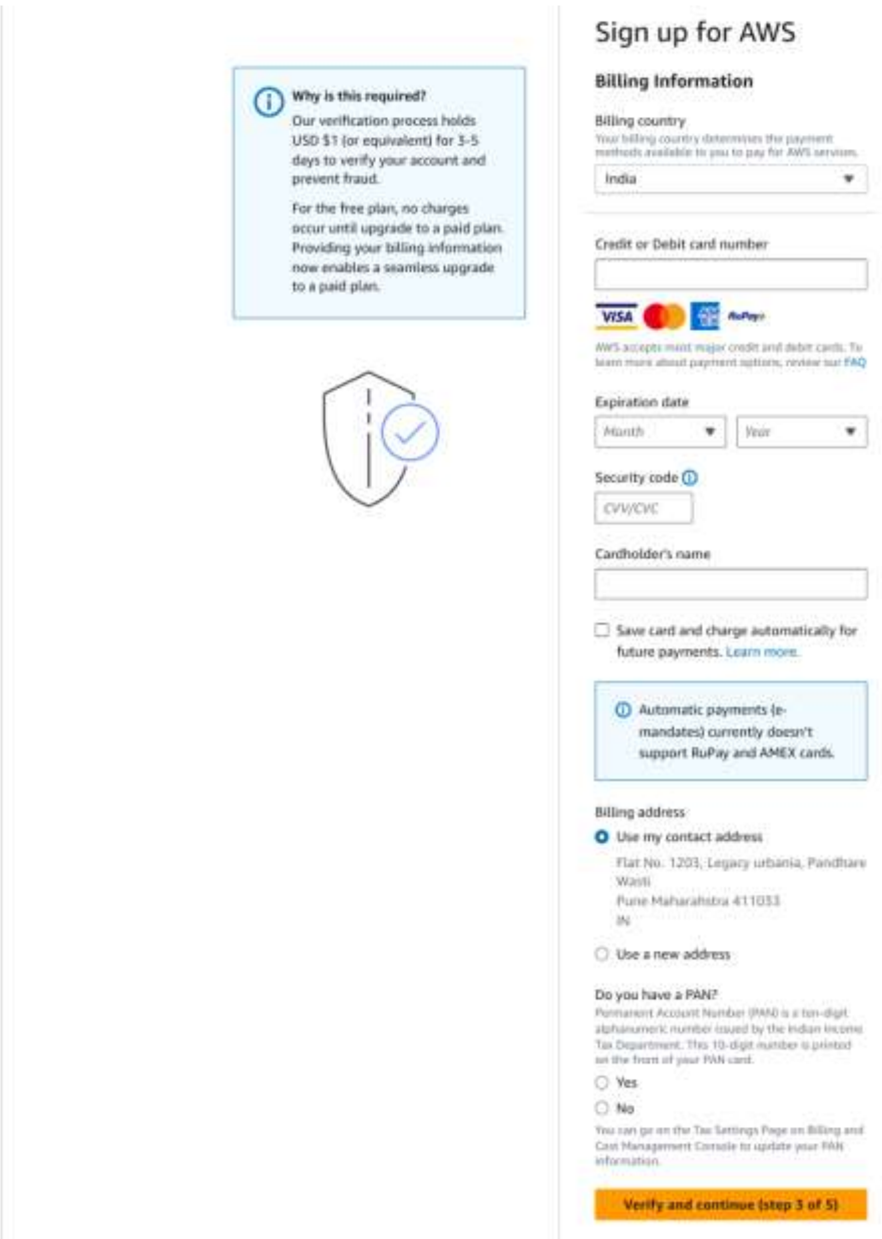
City

State, Province, or Region

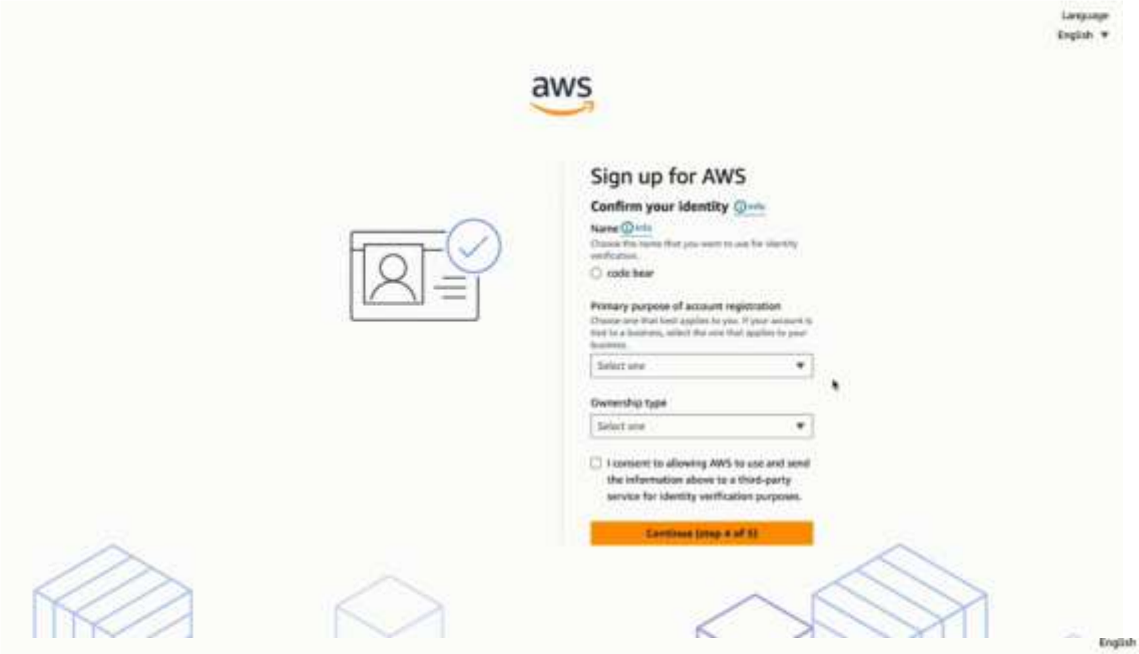
Postal Code

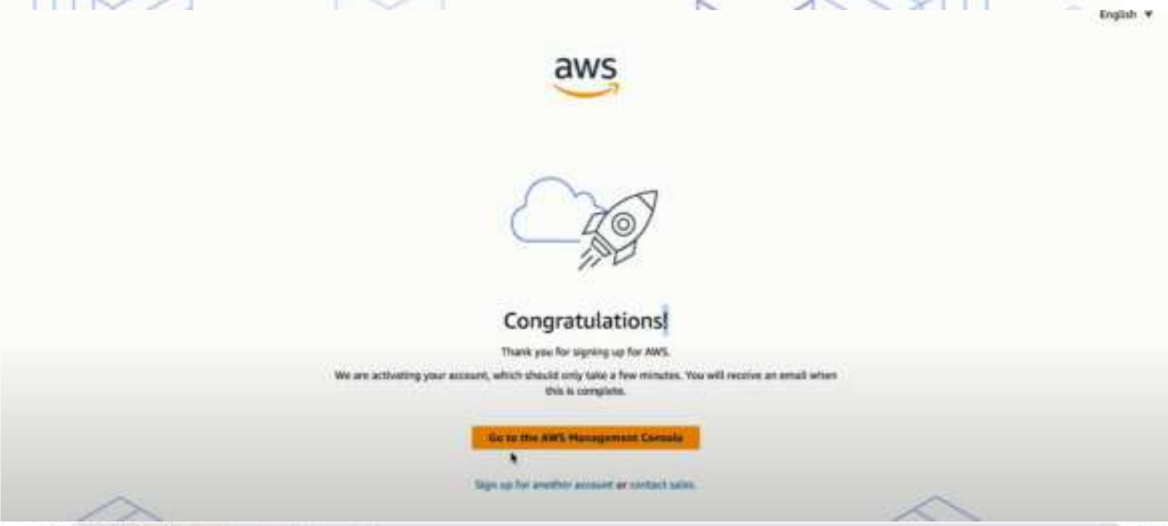
☐ I have read and agree to the terms of the AWS Customer Agreement.

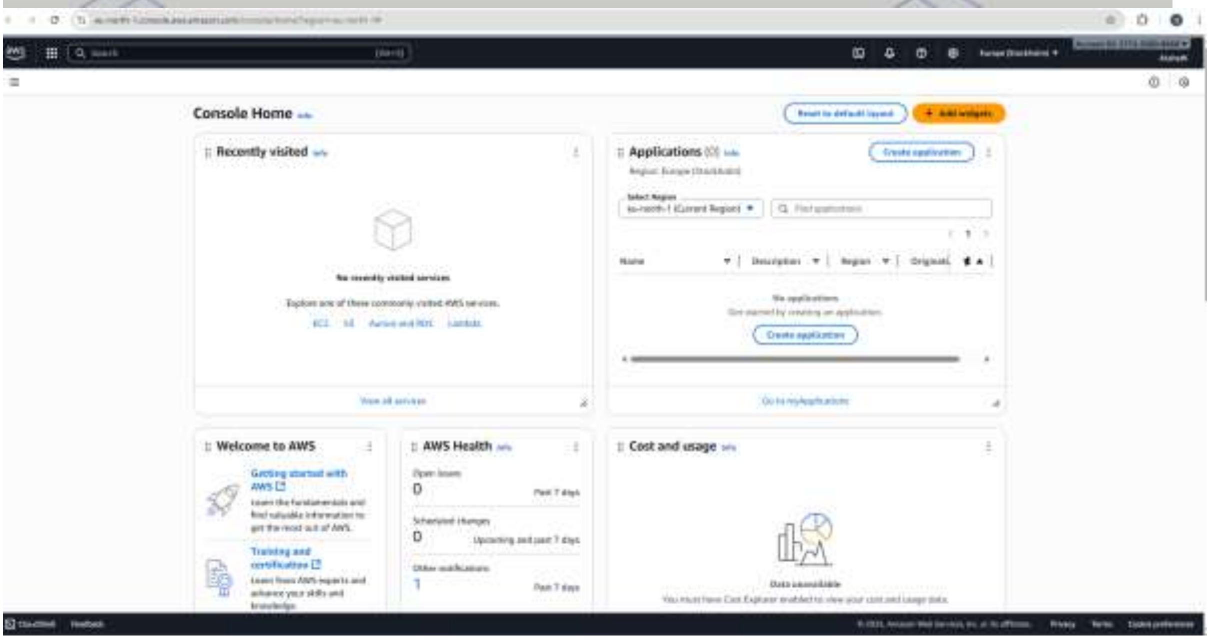
Agree and Continue (step 2 of 5)

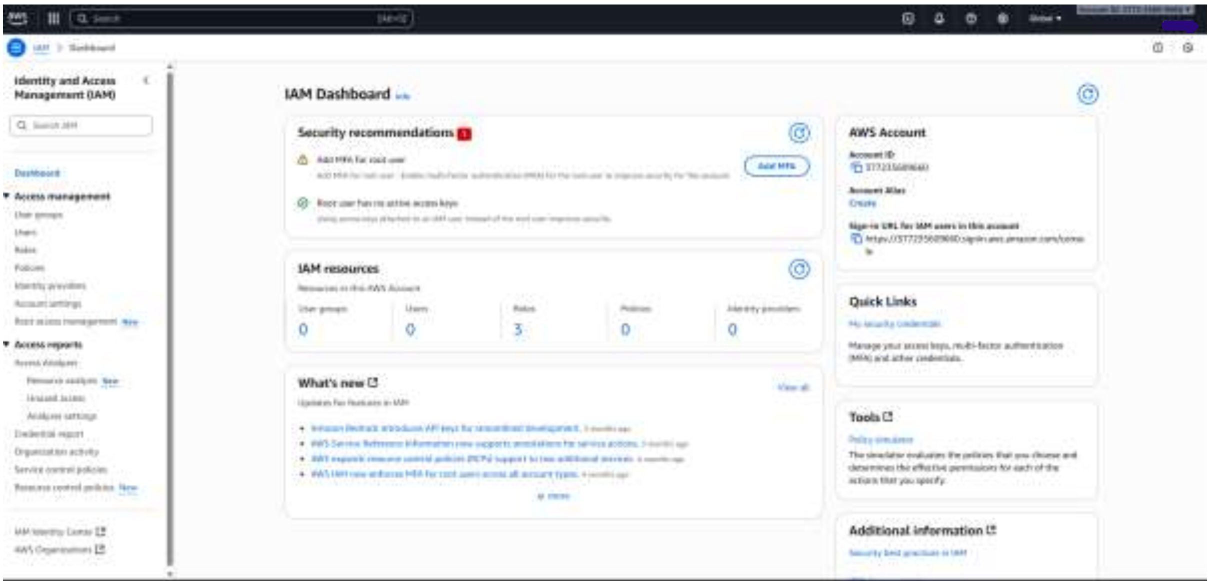

Why is this required?
Our verification process holds USD $1 (or equivalent) for 3-5 days to verify your account and prevent fraud.
For the free plan, no charges occur until upgrade to a paid plan. Providing your billing information now enables a seamless upgrade to a paid plan.
Sign up for AWS
Billing Information
Billing country
Your billing country determines the payment methods available to you to pay for AWS services.
India
Credit or Debit card number
VISA
RuPay
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 urbania, Pandhare Wasti
Pune Maharahstra 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
Basic support - Free
Developer support - From $29/month
Business support - From $100/month
Need Enterprise level support?
Complete sign up

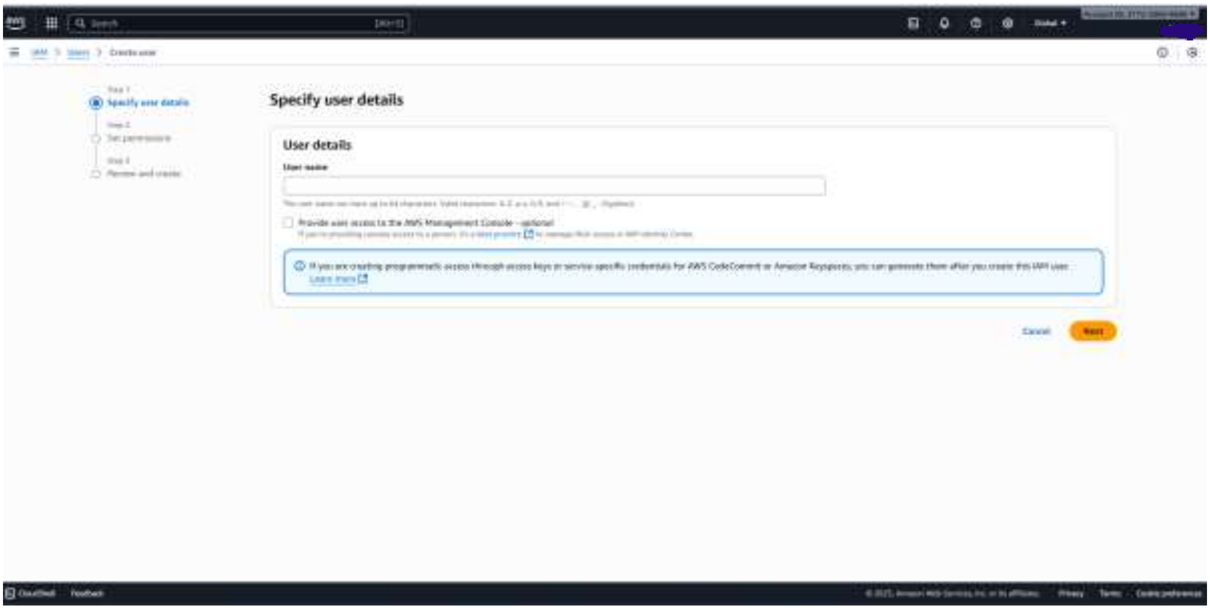

Language
English
aws
Sign up for AWS
Confirm your identity
Name
code bear
Primary purpose of account registration
Select one
Ownership type
Select one
I consent to allowing AWS to use and send the information above to a third-party service for identity verification purposes.

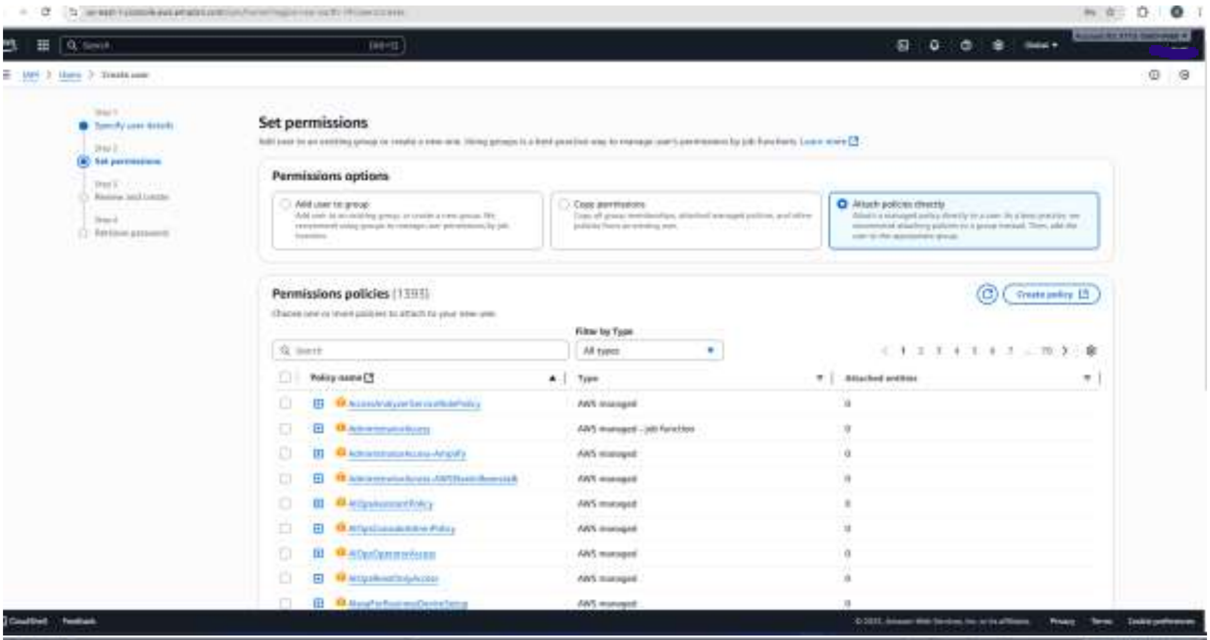
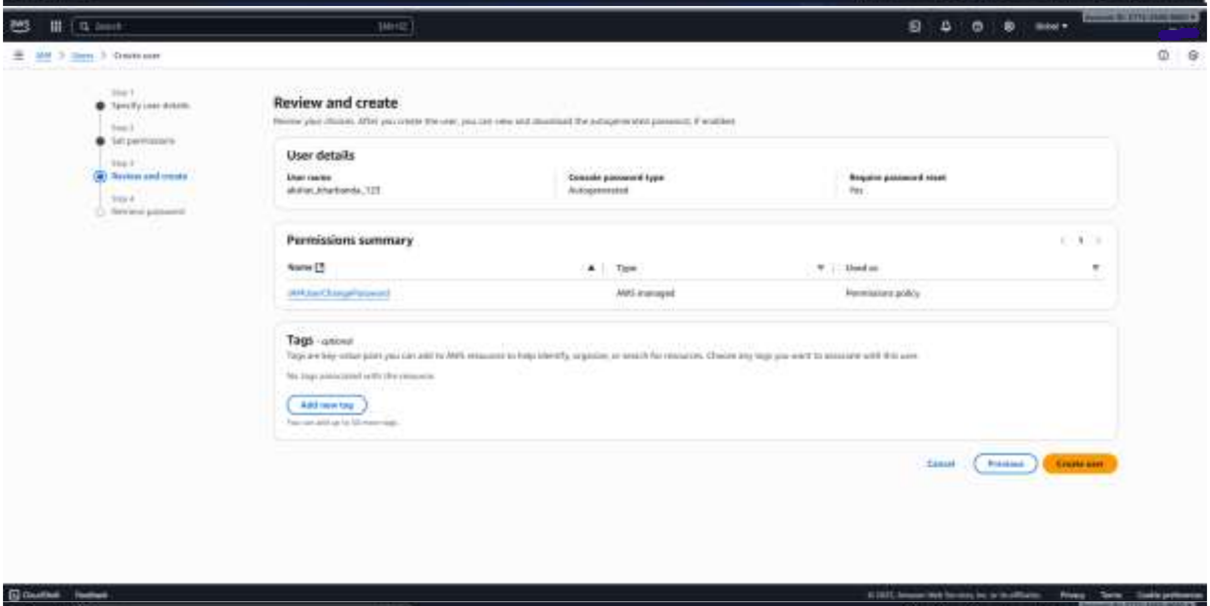
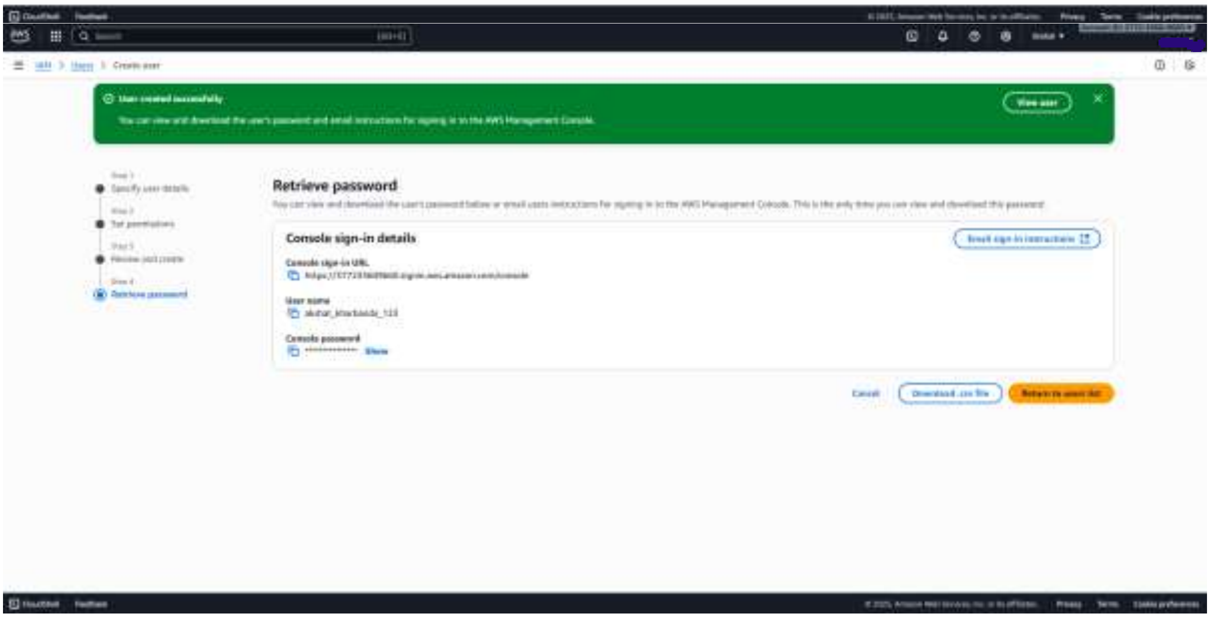

English
aws
Congratulations!
Thank you for signing up for AWS.
We are activating your account, which should only take a few minutes. You will receive an email when this is complete.
Go to the AWS Management Console
Sign up for another account or contact sales.

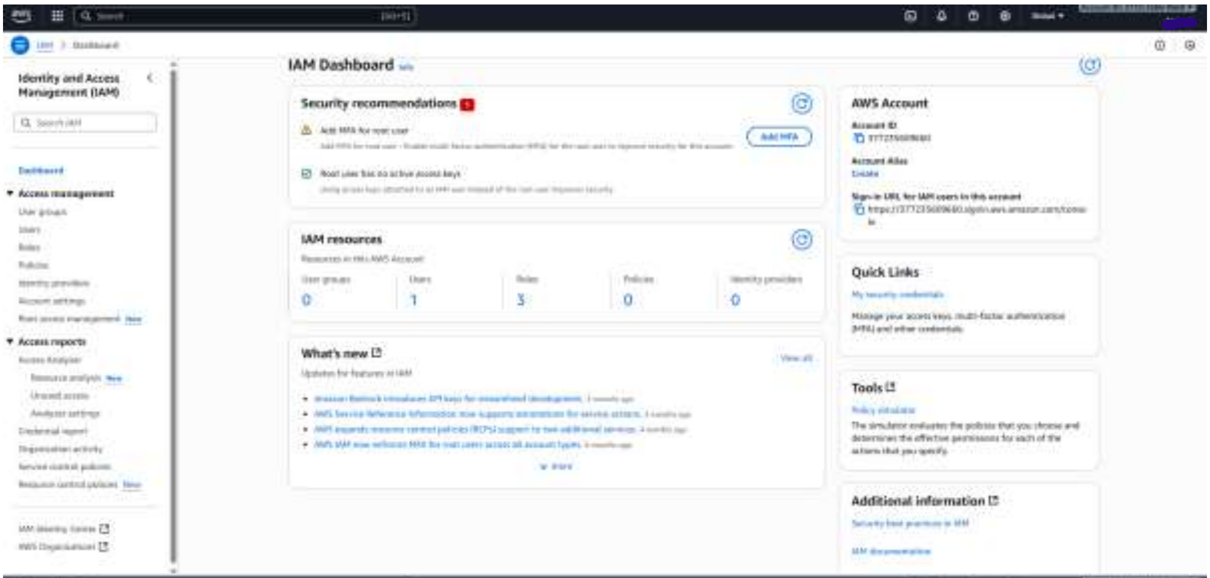

Console Home
Reset to default layout
+ Add widgets
Recently visited
No recently visited services
Applications (0)
Create application
Create application
Welcome to AWS
Getting started with AWS
Training and certification
AWS Health
Open issues
0
Scheduled changes
0
Other notifications
1
Cost and usage
Data unavailable

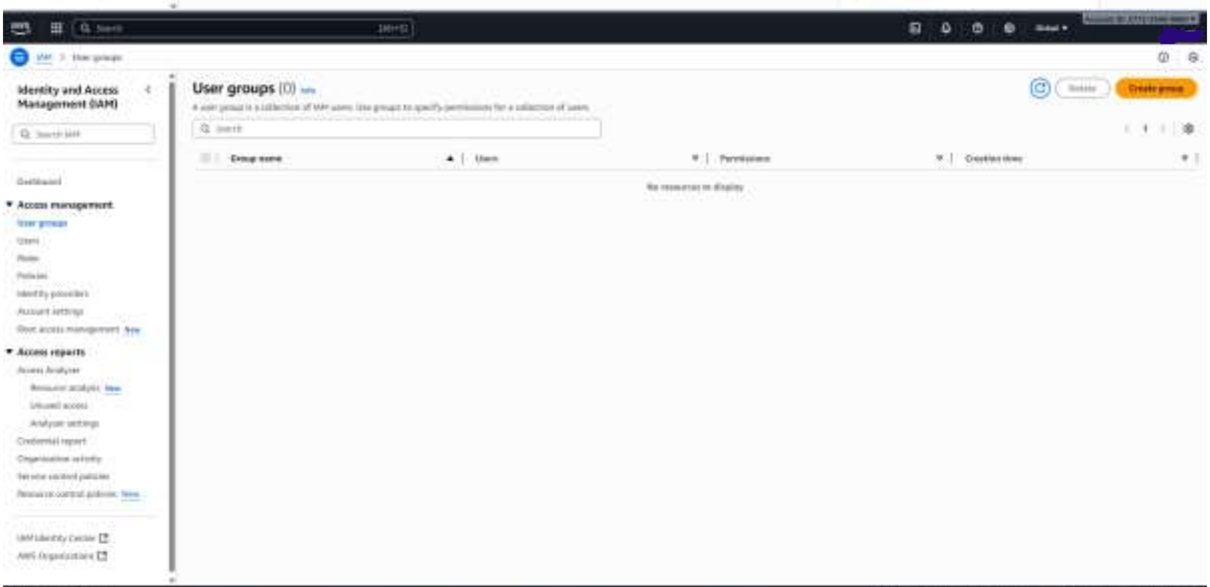

Identity and Access Management (IAM)
Dashboard
Access management
Access reports
IAM Dashboard
Security recommendations
Add MFA for root user
Root user has no active access keys
IAM resources
User groups
Users
Roles
Policies
Identity providers
What's new
AWS Account
Account ID
Account Alias
Quick Links
Tools
Additional information

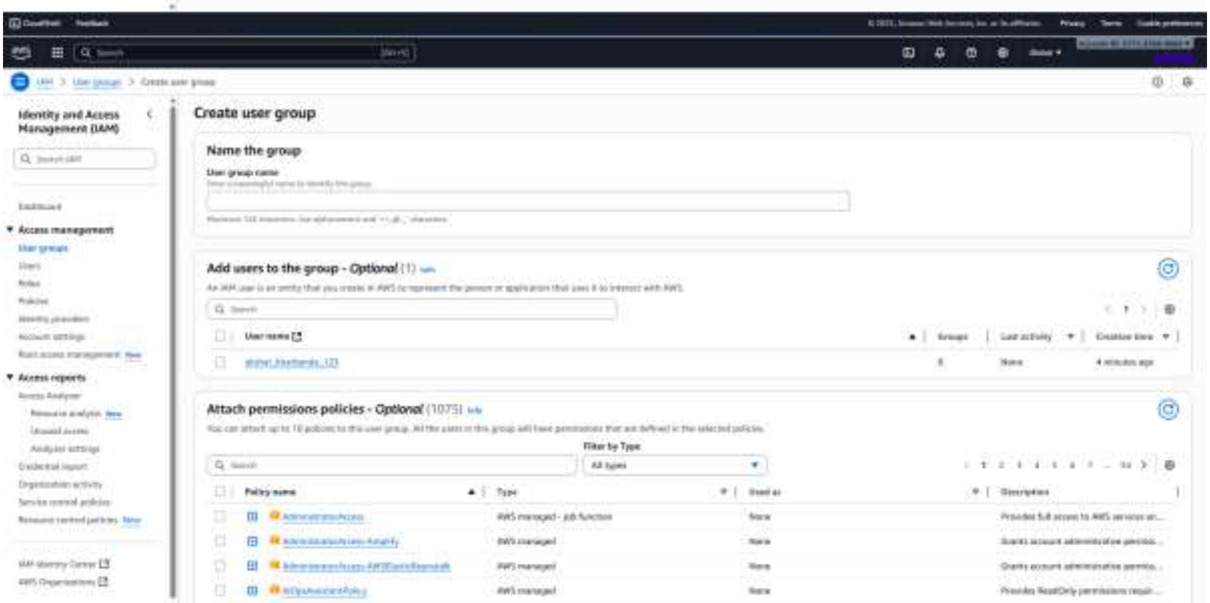

Identity and Access Management (IAM)
Access management
Access reports
Users (0)
User name
Path
Group
Last activity
MFA
Password age
Console last sign-in
Access key ID
Active key age
Create user


Specify user details
Set permissions
Review and create
Specify user details
User details
User name
Provide user access to the AWS Management Console - optional
Cancel
Next


Set permissions
Permissions options
Add user to group
Copy permissions
Attach policies directly
Permissions policies
Filter by Type
All types
Policy name
Type
Attached entities
AWS managed

Review and create
User details
User name
Console password type
Require password reset
Permissions summary
Name
Type
Used as
AWS managed
Permissions policy
Tags - optional
Add new tag
Cancel
Previous
Create user

User created successfully
View user
Retrieve password
Console sign-in details
Email sign-in instructions
Console sign-in URL
User name
Console password
Show
Cancel
Download .csv file
Return to users list







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.

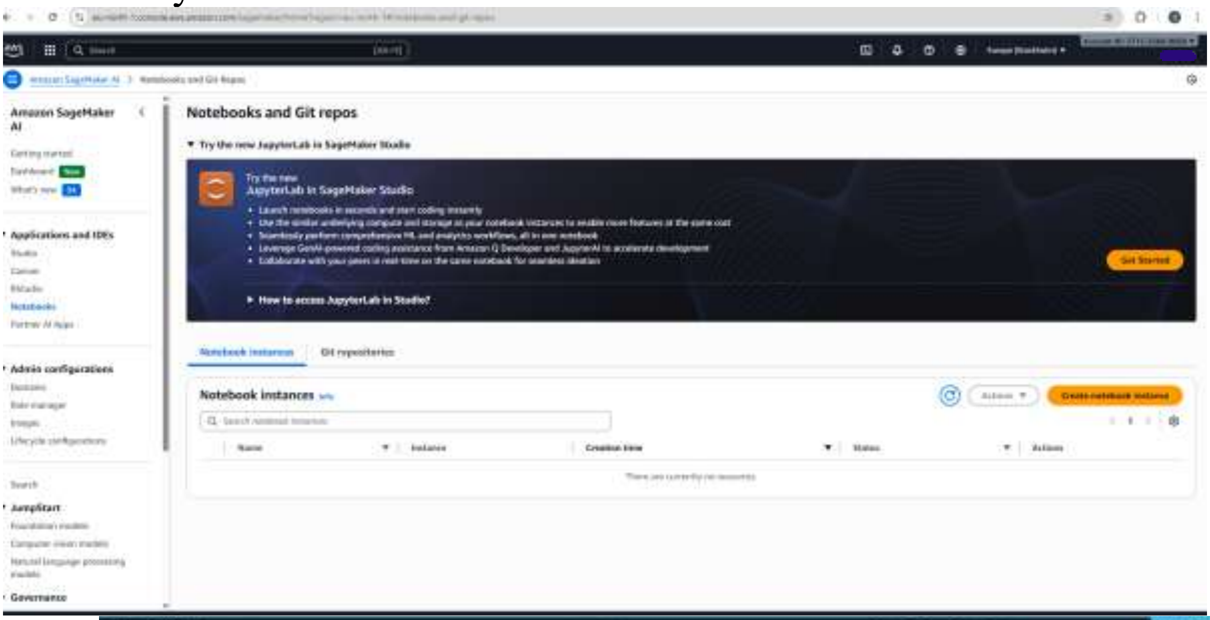
Tools:
- Personal Computer
- Web Browser
- Internet Connection

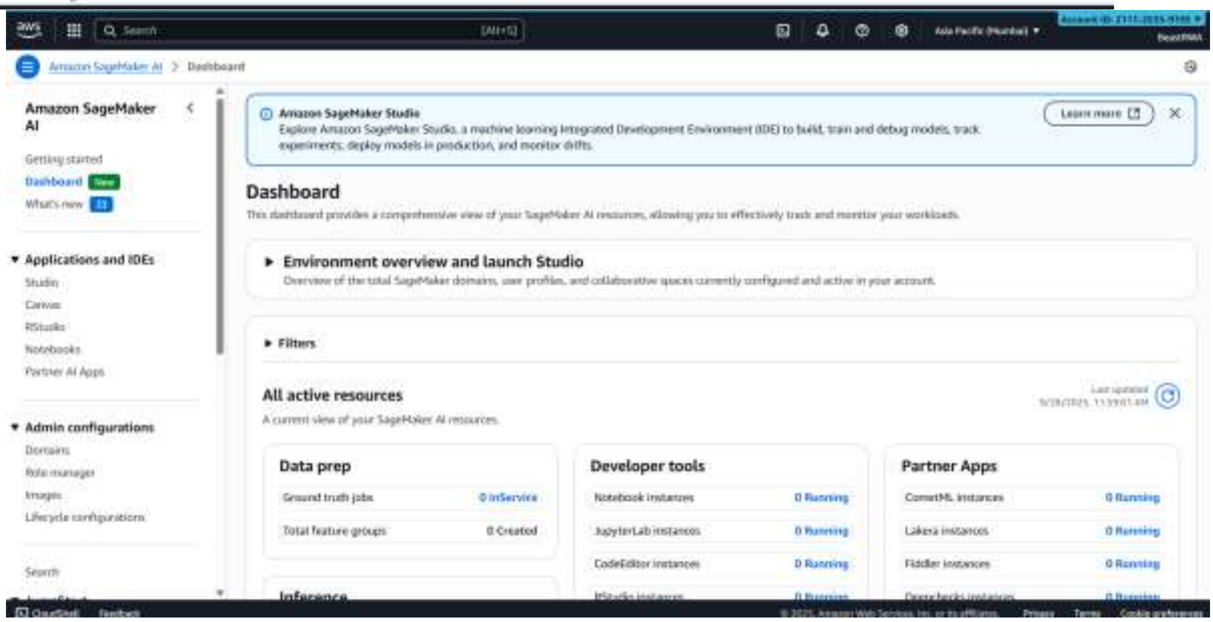
Software Used:
- Amazon Web Services (AWS) Management Console
- Amazon SageMaker
- Jupyter Notebook

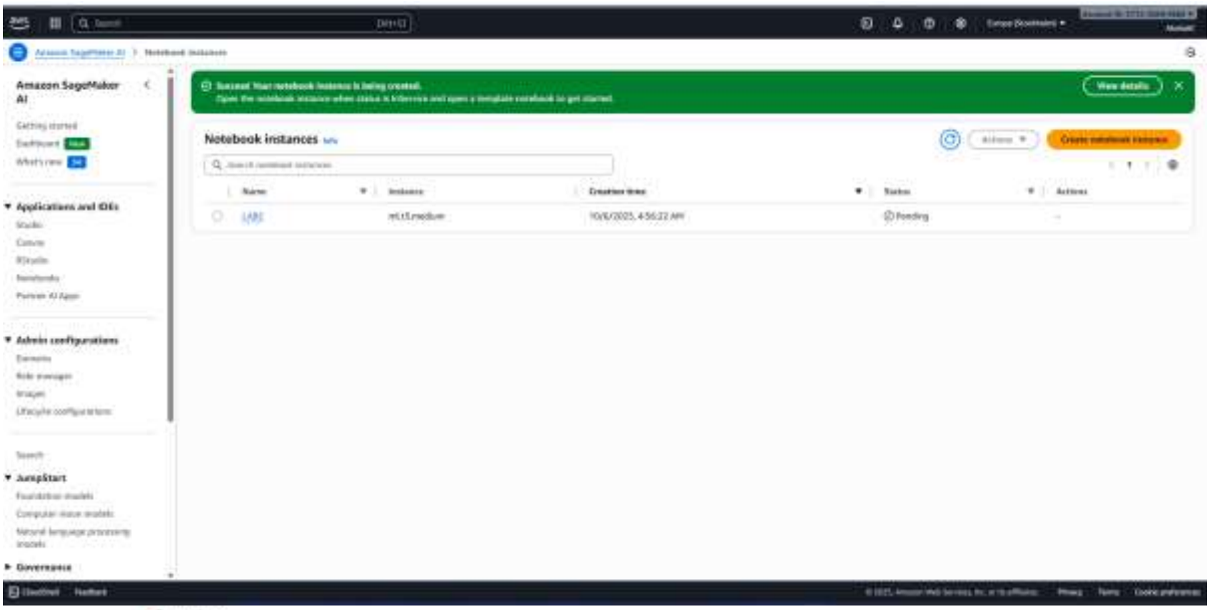
Theory / Concept: Amazon SageMaker is a fully managed cloud service from Amazon Web Services (AWS) that enables developers and data scientists to build, train, and deploy machine learning (ML) models at scale. It simplifies the entire ML workflow, from data labeling and preparation to model training,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

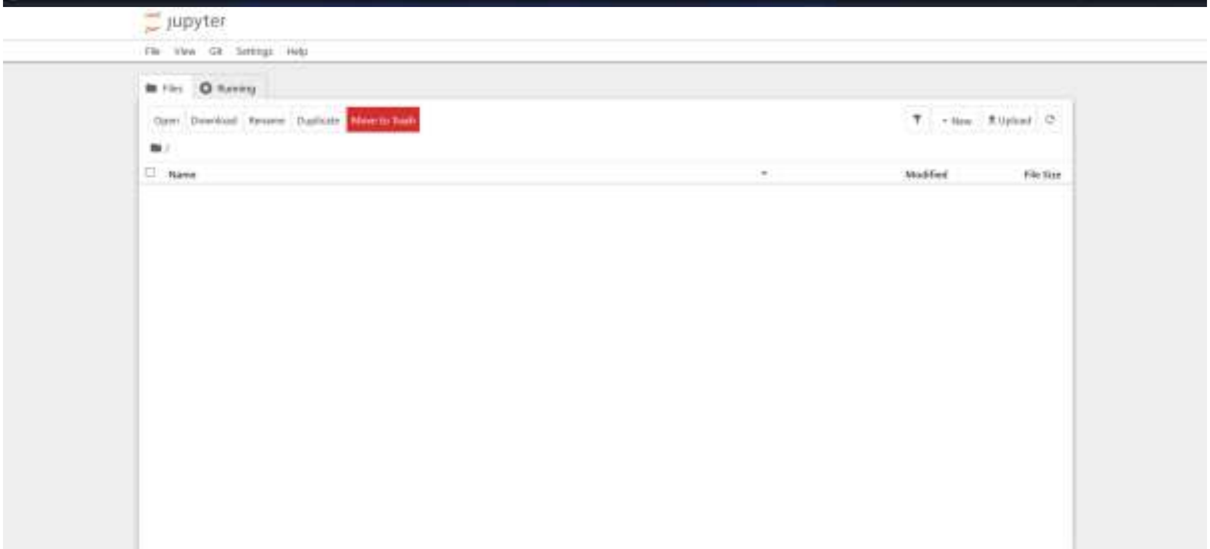
Output:

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.

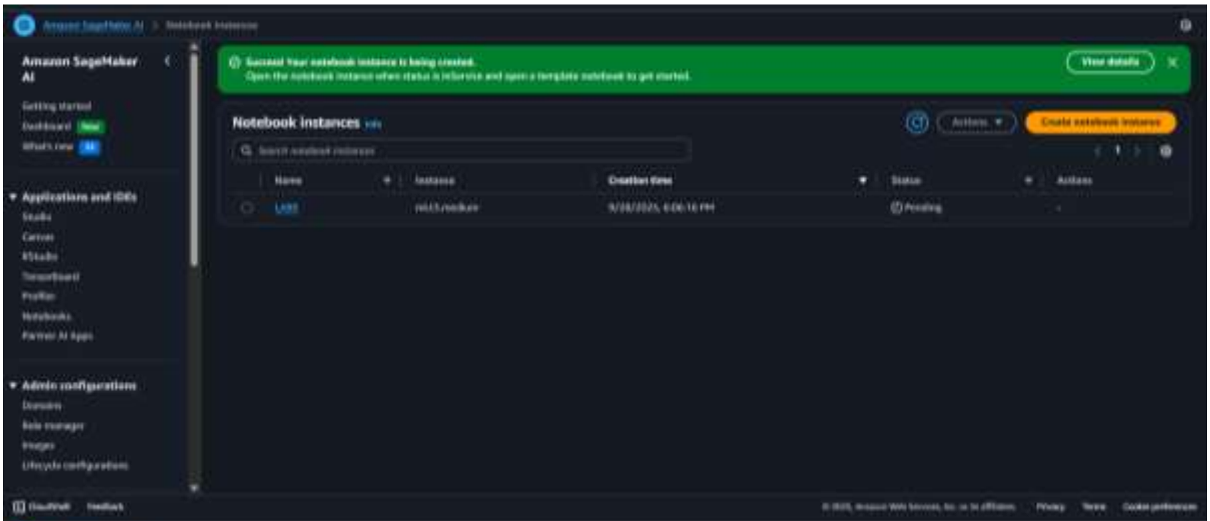


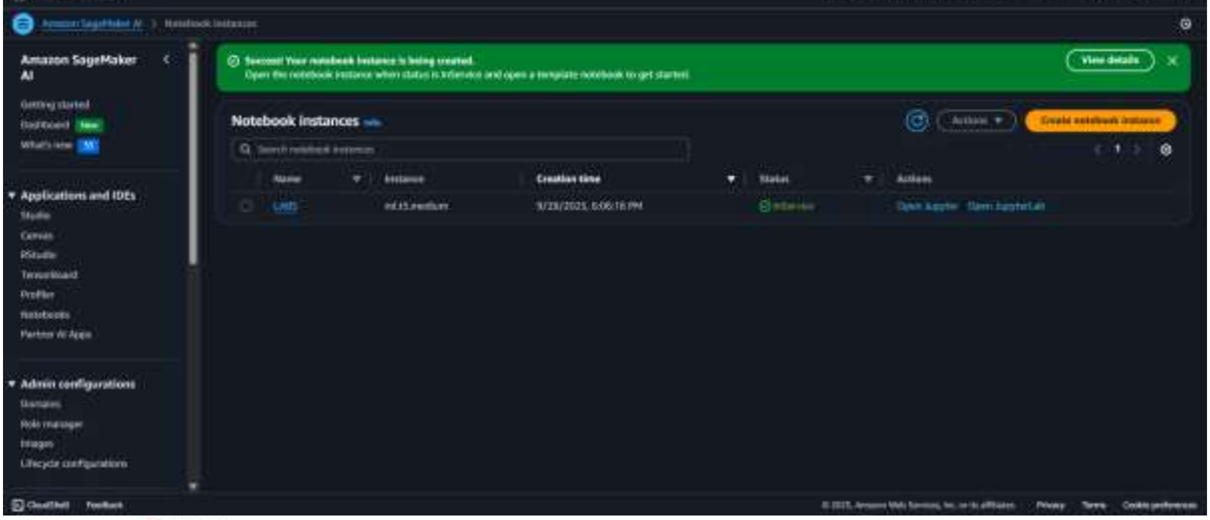


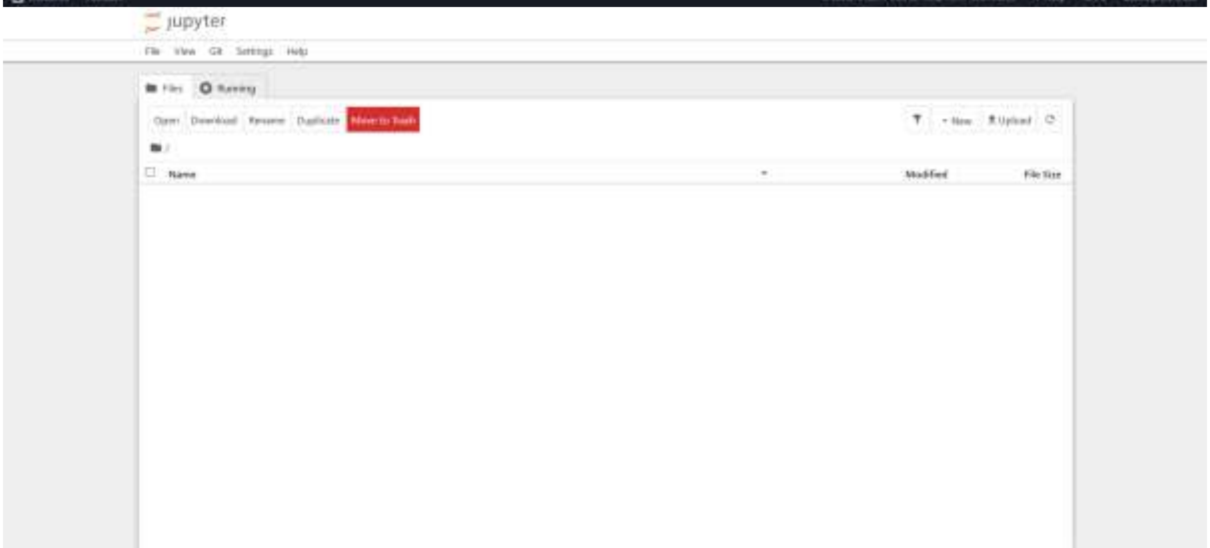

Amazon SageMaker AI
Notebook instances
View details
Create notebook instance
Applications and IDEs
Admin configurations
JumpStart
Governance
ml.t3.medium
Pending

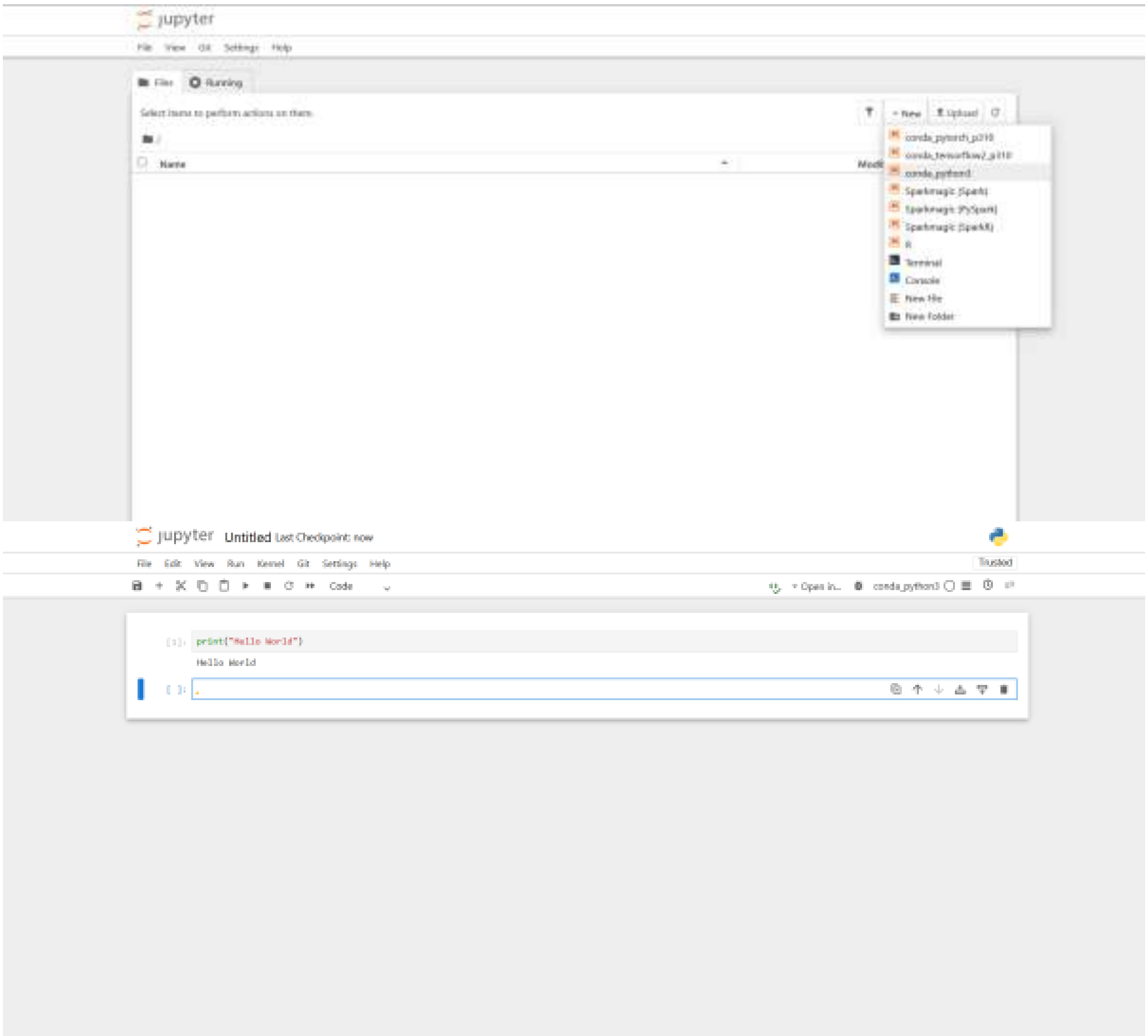

jupyter
File View Git Settings Help
Files
Running
Open
Download
Rename
Duplicate
Move to Trash
New
Upload
Name
Modified
File Size


Create notebook instance
Notebook instance settings
Notebook instance name
Notebook instance type
Platform identifier
Additional configuration
Permissions and encryption
IAM role
Root access - optional


Amazon SageMaker AI
Notebook instances
View details
Applications and IDEs
Admin configurations


Amazon SageMaker AI
Notebook instances
View details
Applications and IDEs
Admin configurations


jupyter
File View Git Settings Help
Files Running
Open Download Rename Duplicate Move to Trash
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)

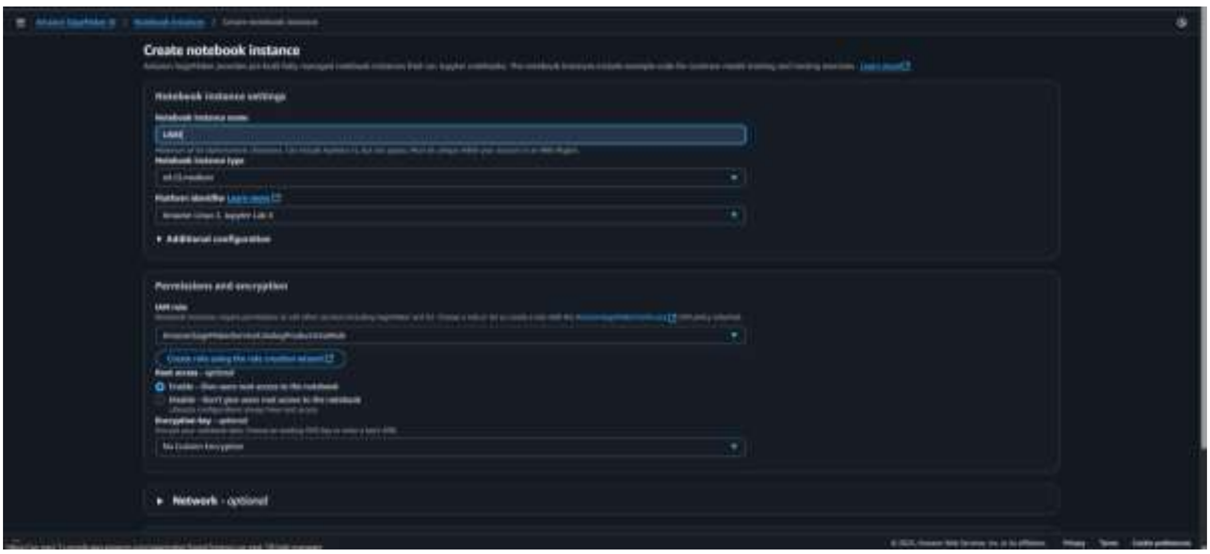
Theory / Concept:

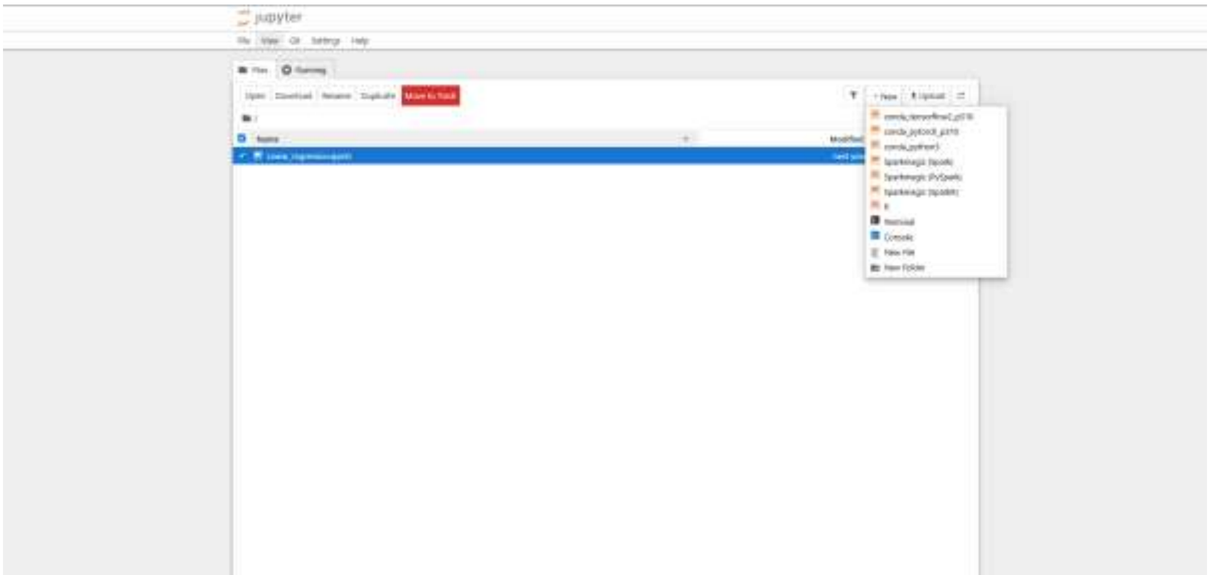
- Amazon SageMaker Notebook: A managed Jupyter notebook instance that provides a powerful, integrated environment for data exploration, analysis, and model training. It comes pre-packaged with common data science libraries and AWS SDKs.

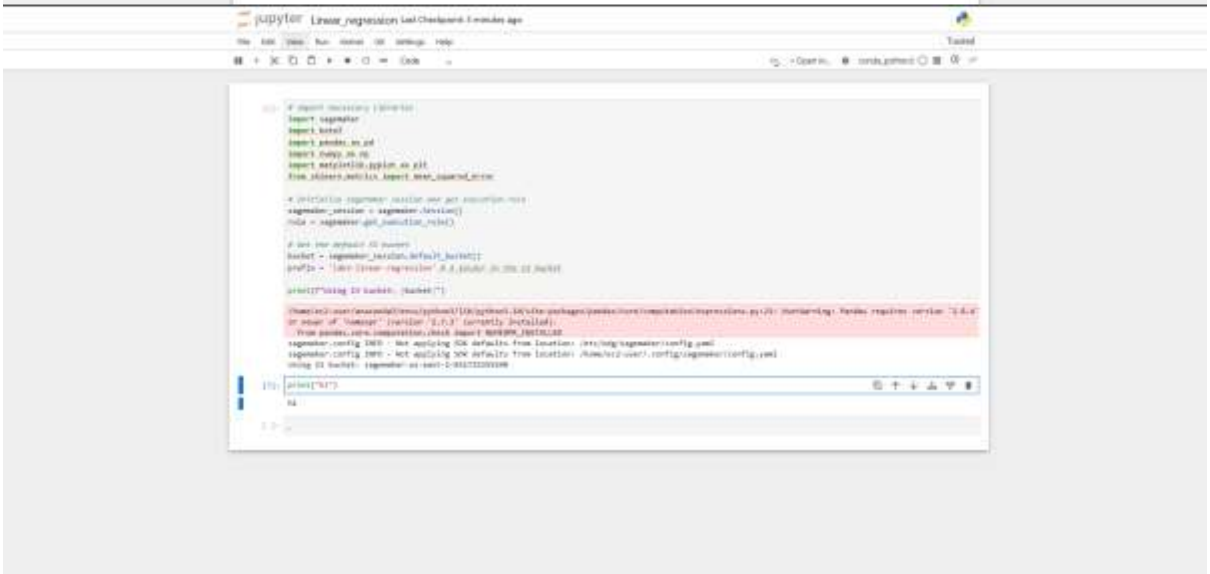
- Amazon S3 (Simple Storage Service): A scalable object storage service used here to store the training dataset. SageMaker training jobs are optimized to read data efficiently from S3.
- Linear Regression: A statistical method for modeling the relationship between a dependent variable (target) and an 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.

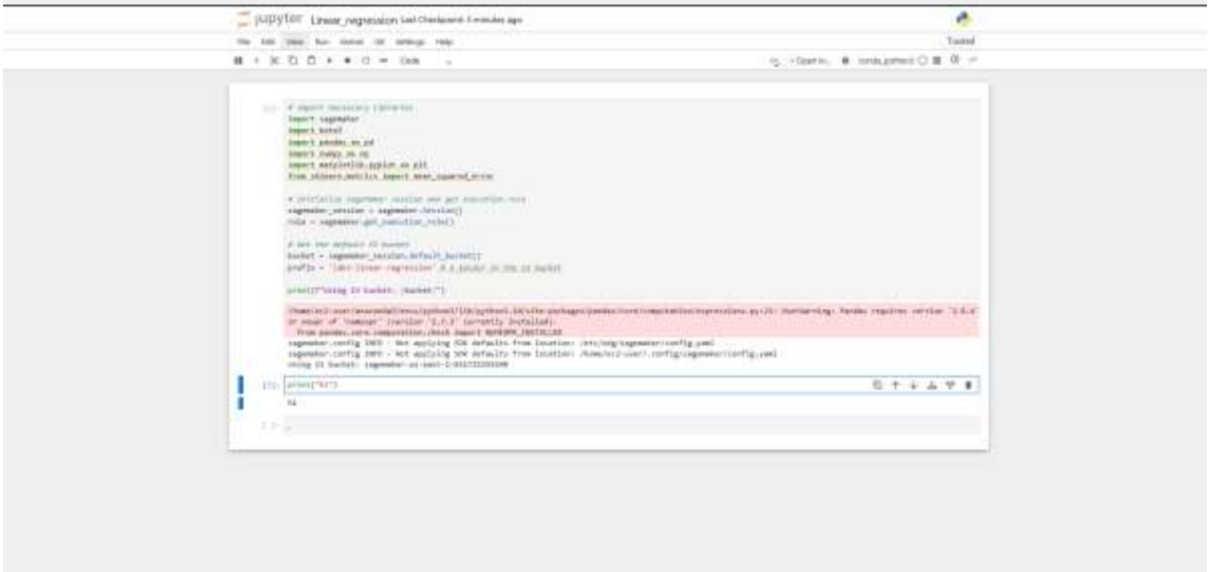
Output:

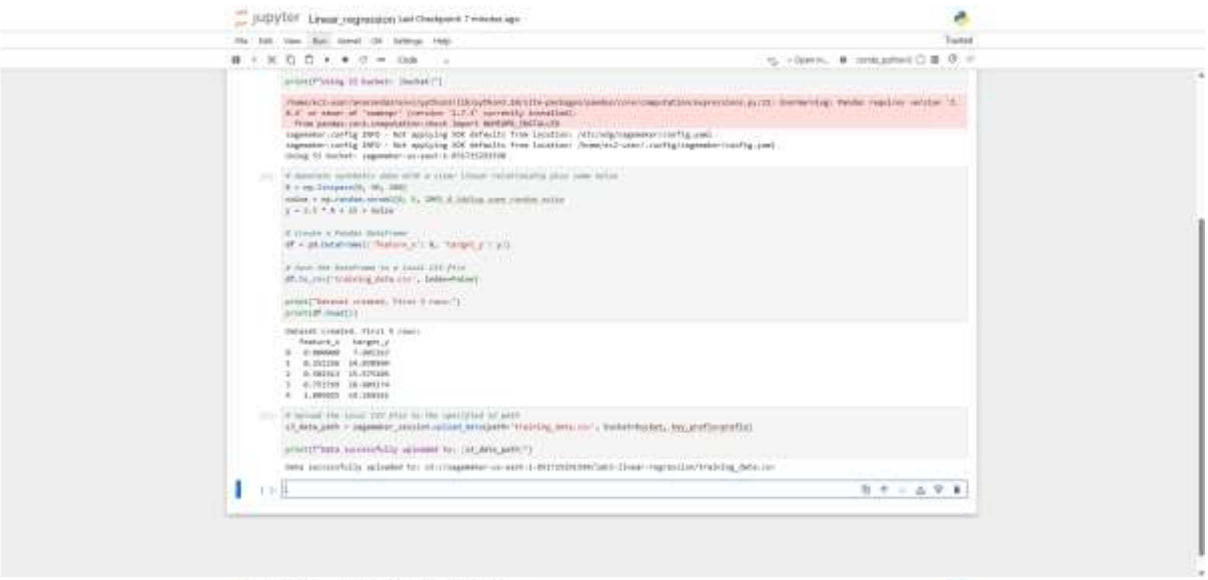
1. **Setup:** Screenshot of the Jupyter notebook cell importing libraries (sagemaker, boto3, numpy, etc.) and setting up the SageMaker session.
2. **Data Generation:** Screenshot of the code cell using NumPy to generate the synthetic dataset.
3. **S3 Upload:** Screenshot of the code cell uploading the '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."

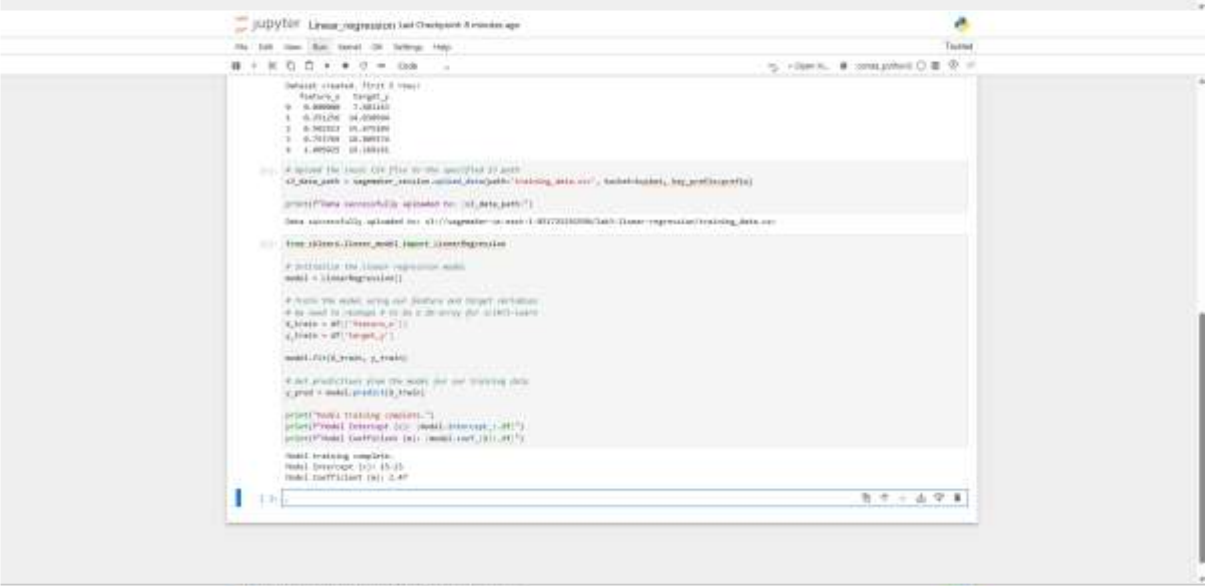


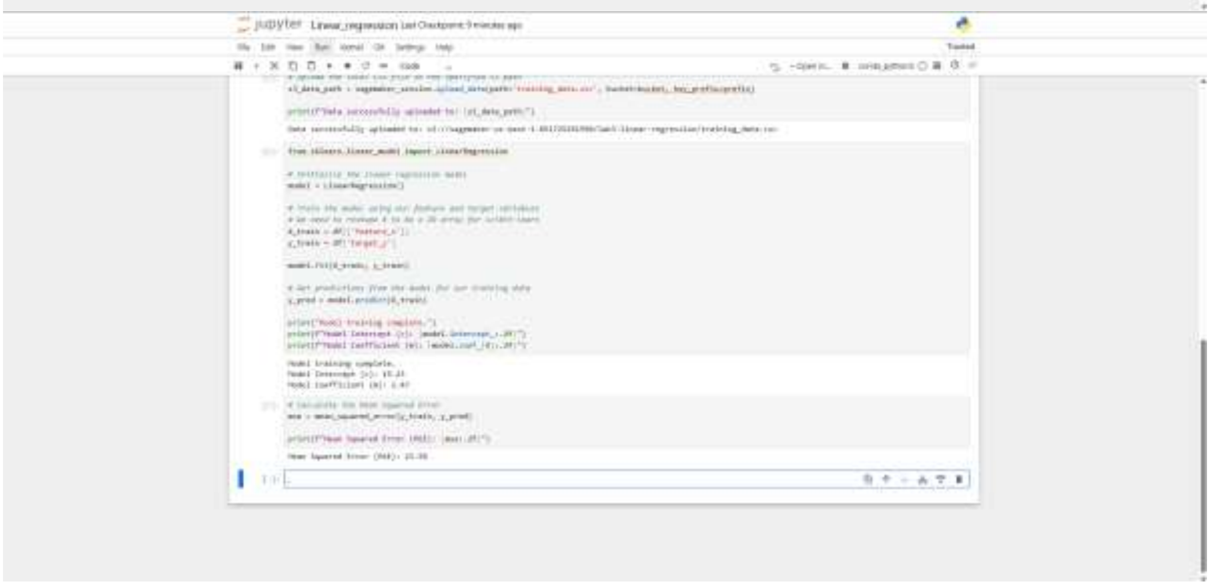

jupyter
Move to Trash

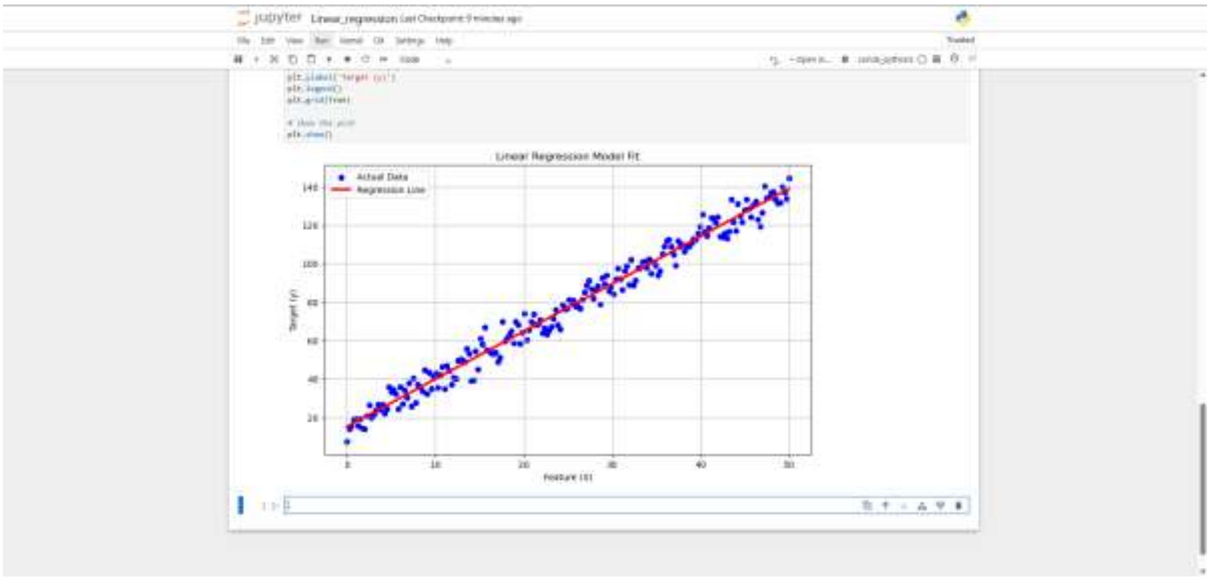

jupyter
Linear_regression


jupyter
Linear_regression


jupyter


jupyter


jupyter



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

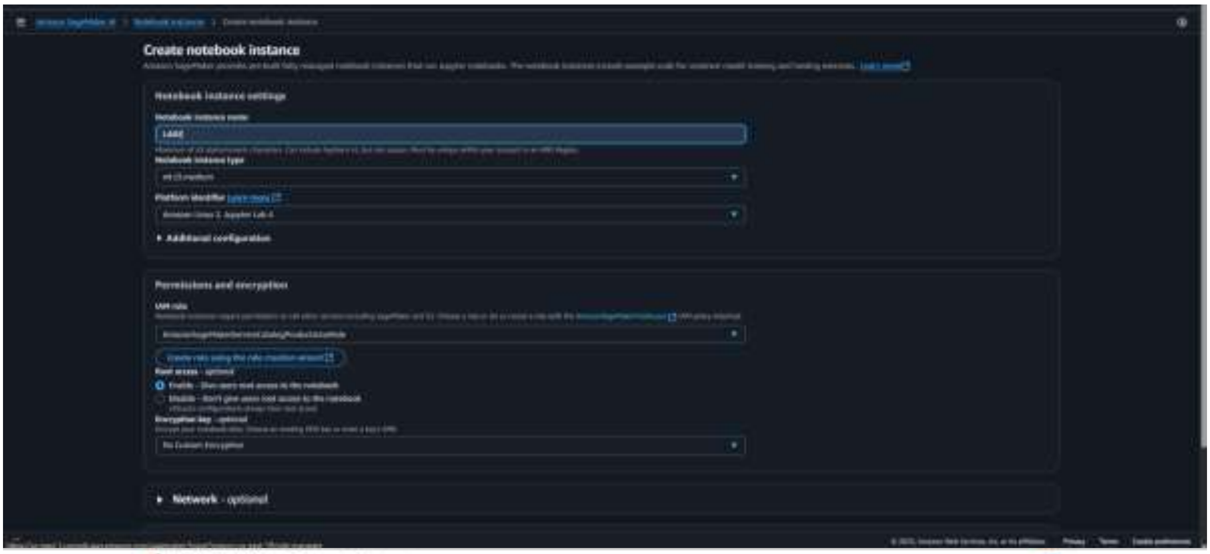
Software Used:

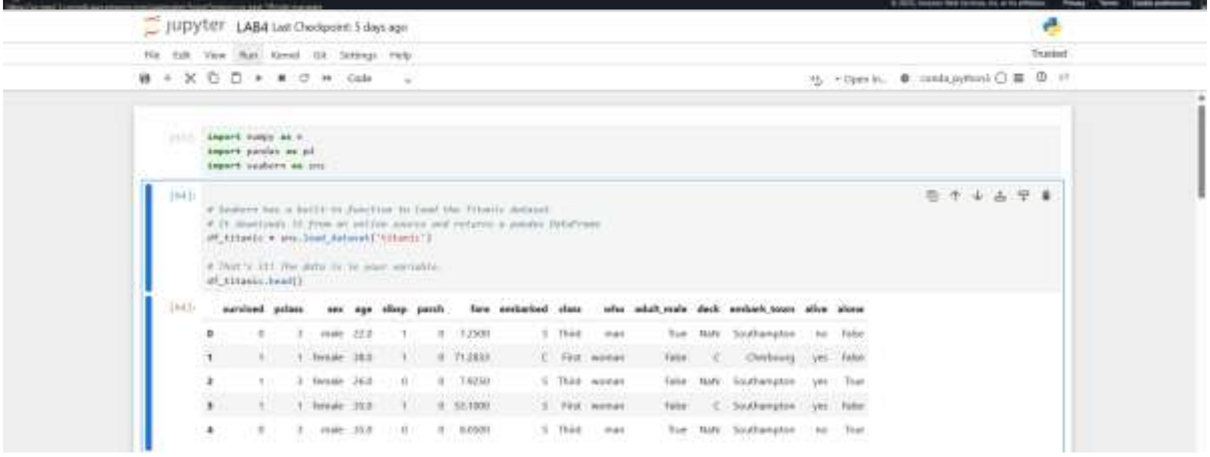
- Amazon Web Services (AWS) Management Console
- Amazon SageMaker (Notebook Instance)
- Jupyter Notebook
- Python (with pandas, numpy, and seaborn libraries)

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:
   - Fill missing age values (e.g., with the median).
   - Drop the deck column.
   - Fill missing embarked values (e.g., with the mode).
6. Data Transformation: Screenshots of the code cells for:
   - Mapping sex to numerical values (0 and 1).
   - 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.





jupyter LAB4 Last Checkpoint: 5 days ago

File Edit View Run Kernel Git Settings Help

Trusted

Code

Open in... conda_python3

[64]:
```python
# 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 it! The data is in your variable.
df_titanic.head()
```

[64]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	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	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

jupyter LAB4 Last Checkpoint: 5 days ago

File Edit View Run Kernel Git Settings Help

Trusted

Code

Open in... conda_python3

| 4 | 0 | 3 | male | 35.0 | 0 | 0 | 8.0500 | S | Third | man | True | NaN | Southampton | no | True |

```python
# Display the data types and count of non-null values for each column
print(df_titanic.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 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          714 non-null    float64
 4   sibsp        891 non-null    int64
 5   parch        891 non-null    int64
 6   fare         891 non-null    float64
 7   embarked     889 non-null    object
 8   class        891 non-null    category
 9   who          891 non-null    object
 10  adult_male   891 non-null    bool
 11  deck         203 non-null    category
 12  embark_town  889 non-null    object
 13  alive        891 non-null    object
 14  alone        891 non-null    bool
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB
None
```

jupyter LAB4 Last Checkpoint: 5 days ago

File Edit View Run Kernel Git Settings Help

Trusted

Code

Open in... conda_python3

```
 12  embark_town  889 non-null    object
 13  alive        891 non-null    object
 14  alone        891 non-null    bool
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB
None
```


```python
# Get 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     0.383838    2.308642   29.699118    0.523008    0.381594   32.204208
std      0.486592    0.836071   14.526497    1.102743    0.806057   49.693429
min      0.000000    1.000000    0.420000    0.000000    0.000000    0.000000
25%      0.000000    2.000000   20.125000    0.000000    0.000000    7.910400
50%      0.000000    3.000000   28.000000    0.000000    0.000000   14.454200
75%      1.000000    3.000000   38.000000    1.000000    0.000000   31.000000
max      1.000000    3.000000   80.000000    8.000000    6.000000  512.329200
```

```python
# Check the dimensions of the dataset (rows, columns)
print(f"The dataset has {df_titanic.shape[0]} rows and {df_titanic.shape[1]} columns.")
```

The dataset has 891 rows and 15 columns.

```python
# Check all the null values
print(df_titanic.isnull().sum())
```

```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
```

jupyter LAB4 Last Checkpoint: 5 days ago

File Edit View Run Kernel Git Settings Help

Trusted

Code

Open in... conda_python3

```
max      1.000000    3.000000   80.000000    8.000000    6.000000  512.329200
```

```python
# Check the dimensions of the dataset (rows, columns)
print(f"The dataset has {df_titanic.shape[0]} rows and {df_titanic.shape[1]} columns.")
```

The dataset has 891 rows and 15 columns.

```python
# Check all the null values
print(df_titanic.isnull().sum())
```

```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
```


```
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

```python
# Calculate the median age
median_age = df_titanic['age'].median()
print(f"Median age: {median_age}")

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

Median age: 28.0

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

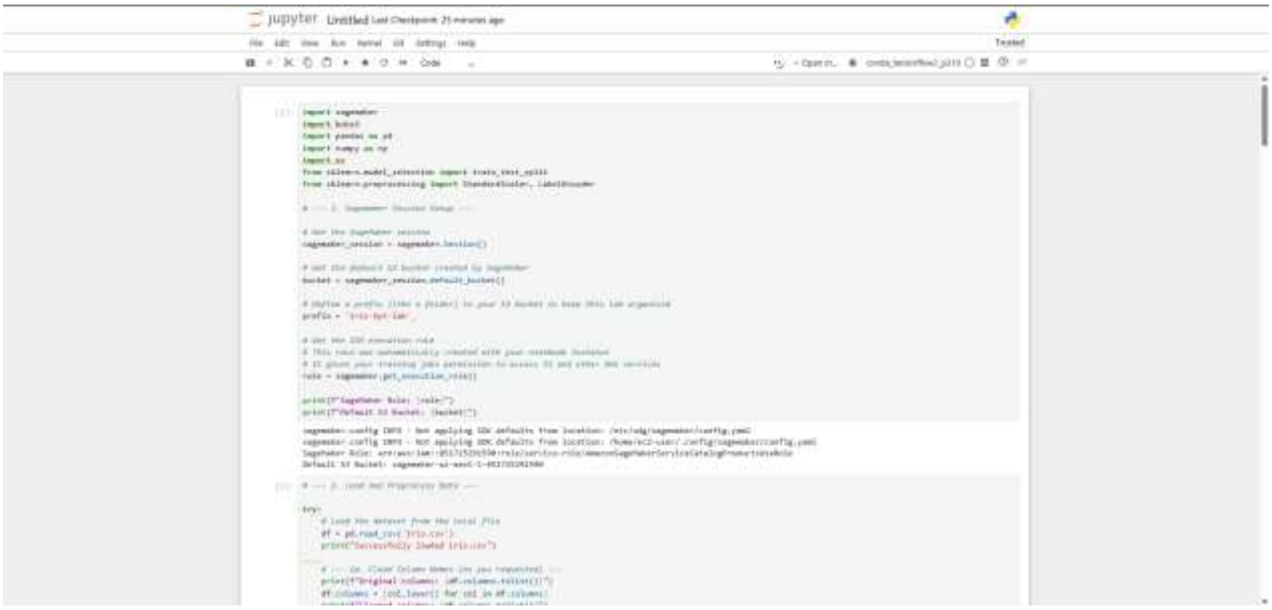
Software Used:

- 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: 25 minutes ago

File Edit View Run Kernel Git Settings Help

Trusted

```
# --- 2. Load and Preprocess Data ---

try:
    # Load the dataset from the local file
    df = pd.read_csv("Iris.csv")
    print("Successfully loaded Iris.csv")

    # --- 2a. Clean Column Names (As you requested) ---
    print(f"Original columns: {df.columns.tolist()}")
    df.columns = [col.lower() for col in df.columns]
    print(f"Cleaned columns: {df.columns.tolist()}")

    # --- 2b. Define Features (X) and Target (y) ---
    # Assuming standard Iris dataset column names
    feature_columns = ['sepallengthcm', 'sepalwidthcm', 'petallengthcm', 'petalwidthcm']
    target_column = 'species'

    # --- 2c. Preprocess Features ---
    scaler = StandardScaler()
    X = scaler.fit_transform(df[feature_columns])

    # --- 2d. Preprocess Target ---
    encoder = LabelEncoder()
    y = encoder.fit_transform(df[target_column])

    # --- 2e. Split Data ---
    # We split into 80% train and 20% validation
    # The HPT job will use the validation set to score the models
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

    # --- 2f. Format Data for Training ---
    # Our script will want the target variable as the first column
    train_data = np.concatenate([y_train.reshape(-1, 1), X_train], axis=1)
    val_data = np.concatenate([y_val.reshape(-1, 1), X_val], axis=1)

    print(f"Training data shape: {train_data.shape}")
    print(f"Validation data shape: {val_data.shape}")

    # --- 2g. Save and Upload to S3 ---
    # Save processed data locally (SageMaker requires header=False, index=False)
    pd.DataFrame(train_data).to_csv('train.csv', header=False, index=False)
    pd.DataFrame(val_data).to_csv('validation.csv', header=False, index=False)

    # Upload to S3
```


jupyter Untitled Last Checkpoint: 25 minutes ago

File Edit View Run Kernel Git Settings Help

Trusted

```
    # The HPT job will use the validation set to score the models
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

    # --- 2f. Format Data for Training ---
    # Our script will want the target variable as the first column
    train_data = np.concatenate([y_train.reshape(-1, 1), X_train], axis=1)
    val_data = np.concatenate([y_val.reshape(-1, 1), X_val], axis=1)

    print(f"Training data shape: {train_data.shape}")
    print(f"Validation data shape: {val_data.shape}")

    # --- 2g. Save and Upload to S3 ---
    # Save processed data locally (SageMaker requires header=False, index=False)
    pd.DataFrame(train_data).to_csv('train.csv', header=False, index=False)
    pd.DataFrame(val_data).to_csv('validation.csv', header=False, index=False)
```


```
    # Upload to S3
    s3_input_train = sagemaker_session.upload_data(path='train.csv', bucket=bucket, key_prefix=f"{prefix}/data/train")
    s3_input_validation = sagemaker_session.upload_data(path='validation.csv', bucket=bucket, key_prefix=f"{prefix}/data/validation")

    print(f"Training data uploaded to: {s3_input_train}")
    print(f"Validation data uploaded to: {s3_input_validation}")

except FileNotFoundError:
    print("Error: 'Iris.csv' not found.")
    print("Please upload 'Iris.csv' to your SageMaker notebook instance directory.")
```

```
Successfully loaded Iris.csv
Original columns: ['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm', 'Species']
Cleaned columns: ['id', 'sepallengthcm', 'sepalwidthcm', 'petallengthcm', 'petalwidthcm', 'species']
Training data shape: (120, 5)
Validation data shape: (30, 5)
Training data uploaded to: s3://sagemaker-us-east-1-[illegible]/iris-hpt-lab/data/train/train.csv
Validation data uploaded to: s3://sagemaker-us-east-1-[illegible]/iris-hpt-lab/data/validation/validation.csv
```

```
%%writefile train.py

import argparse
import os
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras

# 1. --- Define Helper Function to Load Data ---
```

jupyter Untitled Last Checkpoint: 26 minutes ago

File Edit View Run Kernel Git Settings Help

Trusted


```
%%writefile train.py

import argparse
import os
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
```


```
# 1. --- Define Helper Function to Load Data ---
def load_data(channel_path):
    """Load data from the CSV file in the specified channel."""
    # SageMaker channels are directories, so we find the file inside
    input_file = os.path.join(channel_path, os.listdir(channel_path)[0])
    df = pd.read_csv(input_file, header=None)

    # First column is the target (y), rest are features (X)
    y = df.iloc[:, 0].astype(np.float32)
    X = df.iloc[:, 1:].astype(np.float32)
    return X, y

# 2. --- Define Function to Build the Model ---
def build_model(hidden_units, learning_rate):
    """Build a simple Keras Sequential model."""
    model = keras.Sequential([
        # Input layer: 4 features (sepal/petal length/width)
        keras.layers.Input(shape=(4,)),

        # Hidden layer: The number of units is a hyperparameter
        keras.layers.Dense(units=hidden_units, activation='relu'),

        # Output layer: 3 classes (setosa, versicolor, virginica)
        keras.layers.Dense(units=3, 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='sparse_categorical_crossentropy', # Use this loss for integer labels
        metrics=['accuracy']
```

jupyter Untitled Last Checkpoint: 27 minutes ago

File Edit View Run Kernel Git Settings Help

Trusted

```python
        keras.layers.Dense(units=3, 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='sparse_categorical_crossentropy', # Use this loss for integer labels
        metrics=['accuracy']
    )
    return model

# 3. --- Main Training and Parsing Logic ---
if __name__ == '__main__':

    # --- 3a. Parse Arguments ---
    # SageMaker passes 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=16)

    # SageMaker 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'))
```

jupyter Untitled Last Checkpoint: 28 minutes ago

File Edit View Run Kernel Git Settings Help

Trusted

```python
    # --- 3b. Load Data ---
    X_train, y_train = load_data(args.train)
    X_val, y_val = load_data(args.validation)

    # --- 3c. Build Model ---
    model = build_model(args.hidden_units, args.learning_rate)

    # --- 3d. 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=2 # Show logs
    )

    # --- 3e. CRITICAL: Print the Objective Metric ---
    # The SageMaker HPT job will read this line from the logs to find the score.
    # It MUST match the "regex" we define 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};")

    # --- 3f. Save Model ---
    # Save the model in TensorFlow's SavedModel format
    model.save(os.path.join(args.model_dir, '1'))
    print("Model saved.")
```

Writing train.py

[11]: from sagemaker.tensorflow import TensorFlow

jupyter Untitled Last Checkpoint: 28 minutes ago

File Edit View Run Kernel Git Settings Help

Trusted

```python
    model.save(os.path.join(args.model_dir, '1'))
    print("Model saved.")
```

Writing train.py

[11]:

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

# --- 4a. Define the Metric ---
# This tells SageMaker what to look for in the logs.
# It MUST match the print() statement in train.py
metric_definitions = [
    {
        'Name': 'val_accuracy',
        'Regex': 'val_accuracy: ([0-9\\.]+)'
    }
]

# --- 4b. Create the TensorFlow Estimator ---
# This is the blueprint for a single training job.
estimator = TensorFlow(
    entry_point='train.py',            # Our script
    role=role,
    instance_count=1,                  # Use 1 instance per job
    instance_type='ml.m5.large',
    framework_version='2.13',
    py_version='py310',

    # *** THIS IS THE CRITICAL FIX ***
```

jupyter Untitled Last Checkpoint: 28 minutes ago

File Edit View Run Kernel Git Settings Help — Trusted

```
    instance_type='ml.m5.large',
    framework_version='2.12',
    py_version='py310',

    # *** THIS IS THE CRITICAL FIX ***
    # We are passing the metric definition to the estimator
    metric_definitions=metric_definitions
)

# --- 4c. Define Hyperparameter Ranges to Search ---
hyperparameter_ranges = {
    'learning-rate': ContinuousParameter(0.0001, 0.01),
    'hidden-units': IntegerParameter(16, 50)
}

# --- 4d. Create the HyperparameterTuner Object ---
# This tuner will now inherit the metric_definitions from the estimator
tuner = HyperparameterTuner(
    estimator=estimator,
    objective_metric_name='val_accuracy', # This name now matches a defined metric
    objective_type='Maximize',
    hyperparameter_ranges=hyperparameter_ranges,
    metric_definitions=metric_definitions,
    max_jobs=6,
    max_parallel_jobs=2
)

print("Tuner object created successfully with metrics. Ready for Step 5.")
```

```
Tuner object created successfully with metrics. Ready for Step 5.
```

```
[11]: # --- 5. Launch the Tuning Job ---
```

jupyter Untitled Last Checkpoint: 28 minutes ago

File Edit View Run Kernel Git Settings Help — Trusted

conda_tensorflow2_p310

Tuner object created successfully with metrics. Ready for Step 5.

```
[11]: # --- 5. Launch the Tuning Job ---

# Create S3 input objects
s3_train = sagemaker.inputs.TrainingInput(s3_input_train, content_type='text/csv')
s3_validation = sagemaker.inputs.TrainingInput(s3_input_validation, content_type='text/csv')

print("Starting Hyperparameter Tuning Job... This will take 10-15 minutes.")

tuner.fit({'train': s3_train, 'validation': s3_validation}, wait=True)

print("Tuning Job complete.")
```

```
Starting Hyperparameter Tuning Job... This will take 10-15 minutes.
No finished training job found associated with this estimator. Please make sure this estimator is only used for building workflow config
No finished training job found associated with this estimator. Please make sure this estimator is only used for building workflow config
.........................................................................!
Tuning Job complete.
```

```
[17]: # --- 6. Analyze Results ---

# Get a summary of all tuning jobs
results = tuner.analytics().dataframe()
print("All Tuning Job Results:")
print(results.sort_values('FinalObjectiveValue', ascending=False))

# Get the name of the best job
best_job_name = tuner.best_training_job()
print(f"\nBest Training Job Name: {best_job_name}")
```

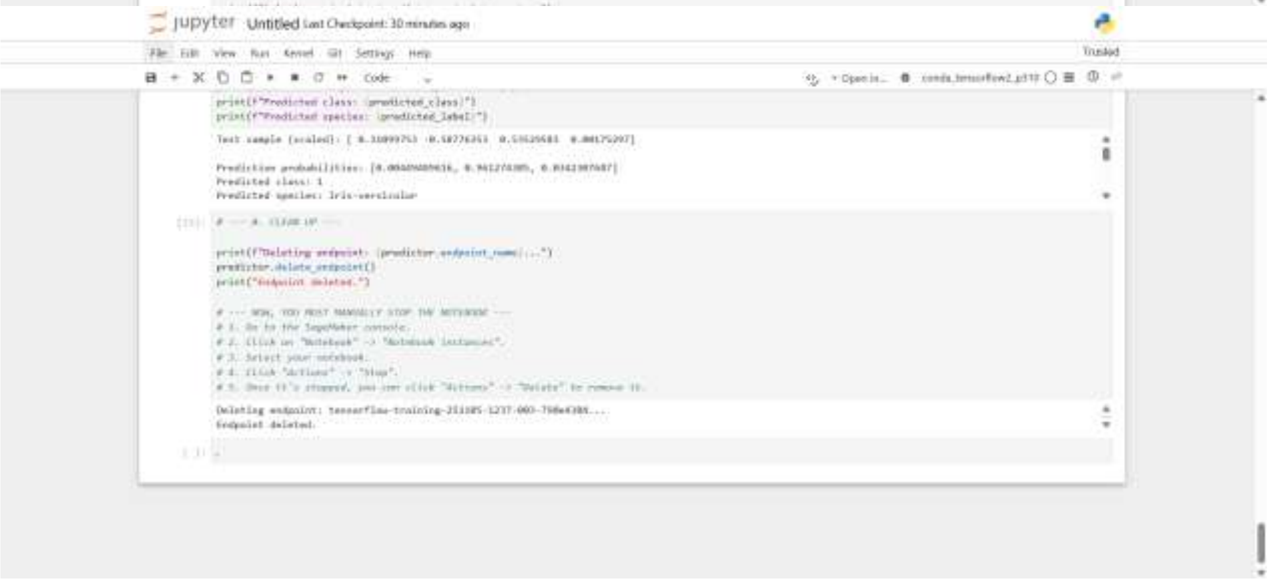
jupyter Untitled Last Checkpoint: 29 minutes ago

File Edit View Run Kernel Git Settings Help — Trusted

conda_tensorflow2_p310

```
    initial_instance_count=1,
    instance_type='ml.m5.large'   # <-- changed from ml.t2.medium
)

print(f"\n✅ Endpoint '{predictor.endpoint_name}' deployed successfully!")

# --- 7c. Optional: Test the Endpoint ---
import numpy as np

# Example test input (adjust shape based on your model)
test_data = np.random.rand(1, 4)  # e.g., 4 features if your model expects that

print("\nSending test prediction request...")
prediction = predictor.predict(test_data)
print("Prediction result:", prediction)
```

```
All Tuning Job Results:
   hidden-units  learning-rate                                  TrainingJobName  \
0          44.0       0.005796  tensorflow-training-251105-1237-006-[illegible]
3          57.0       0.000440  tensorflow-training-251105-1237-003-790a4384
1          32.0       0.000139  tensorflow-training-251105-1237-005-193b9713
2          45.0       0.000810  tensorflow-training-251105-1237-004-[illegible]
4          31.0       0.000148  tensorflow-training-251105-1237-002-ce5e6743
5          23.0       0.000114  tensorflow-training-251105-1237-001-bfa8582a

  TrainingJobStatus  FinalObjectiveValue         TrainingStartTime  \
0         Completed             1.000000 2025-11-05 12:42:13+00:00
3         Completed             1.000000 2025-11-05 12:40:57+00:00
1         Completed             0.000000 2025-11-05 12:42:08+00:00
2         Completed             0.000000 2025-11-05 12:40:56+00:00
4         Completed             0.666667 2025-11-05 12:37:59+00:00
5         Completed             0.266667 2025-11-05 12:37:57+00:00
```



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.

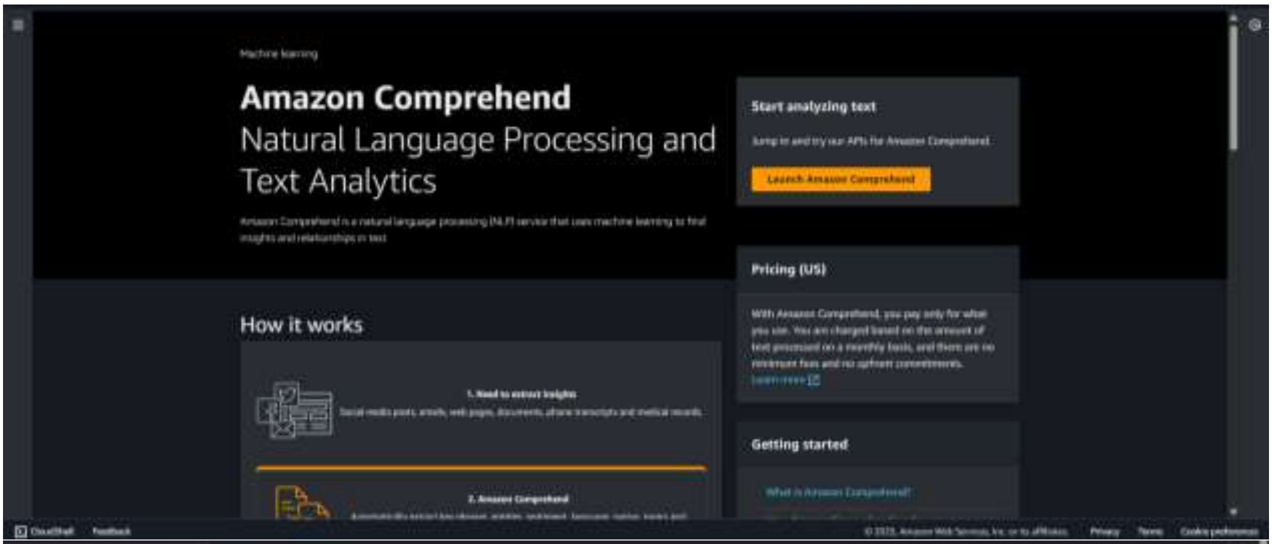
Tools:
- Personal Computer
- Web Browser
- Internet Connection

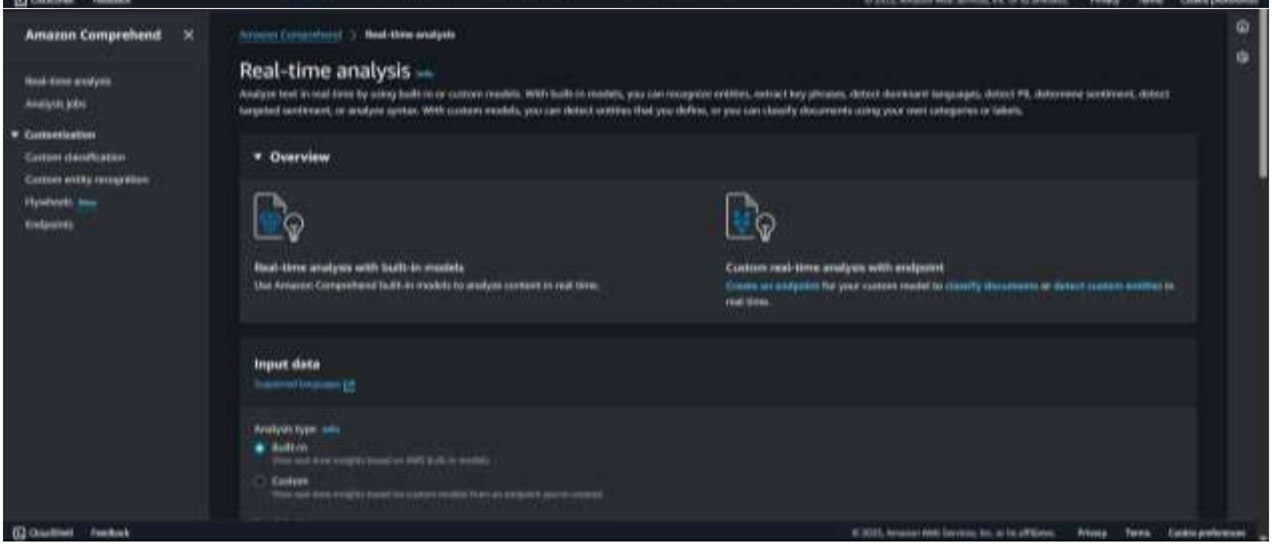
Software Used:
- Amazon Web Services (AWS) Management Console
- Amazon Comprehend

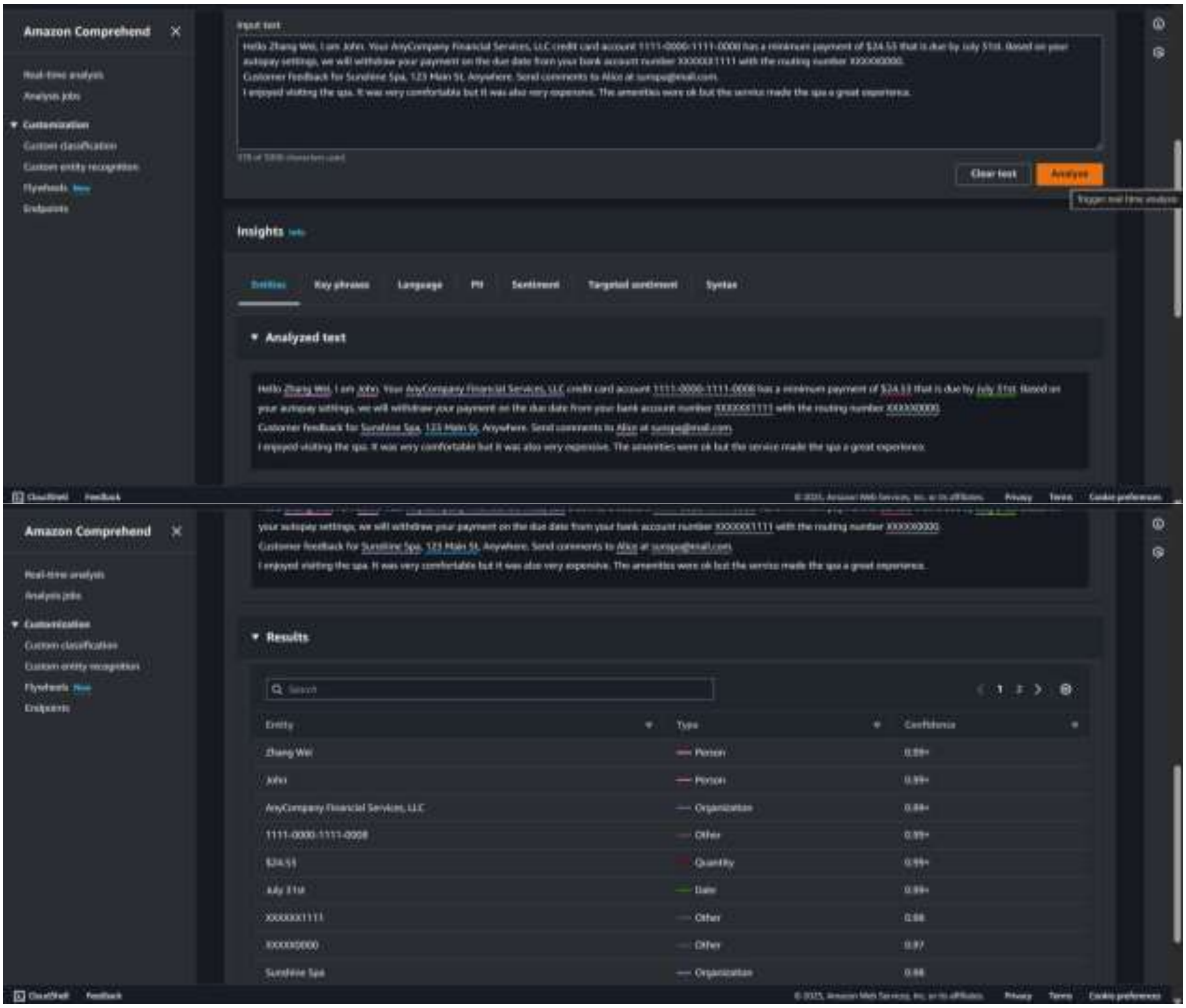
Theory / Concept: Amazon Comprehend is a natural language processing (NLP) service from Amazon Web Services that uses machine learning to uncover valuable insights and relationships in text. It is a fully managed service, meaning you don't need to train your own models to use it.
One of its key features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







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.

<u>**Practical 07**</u>

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

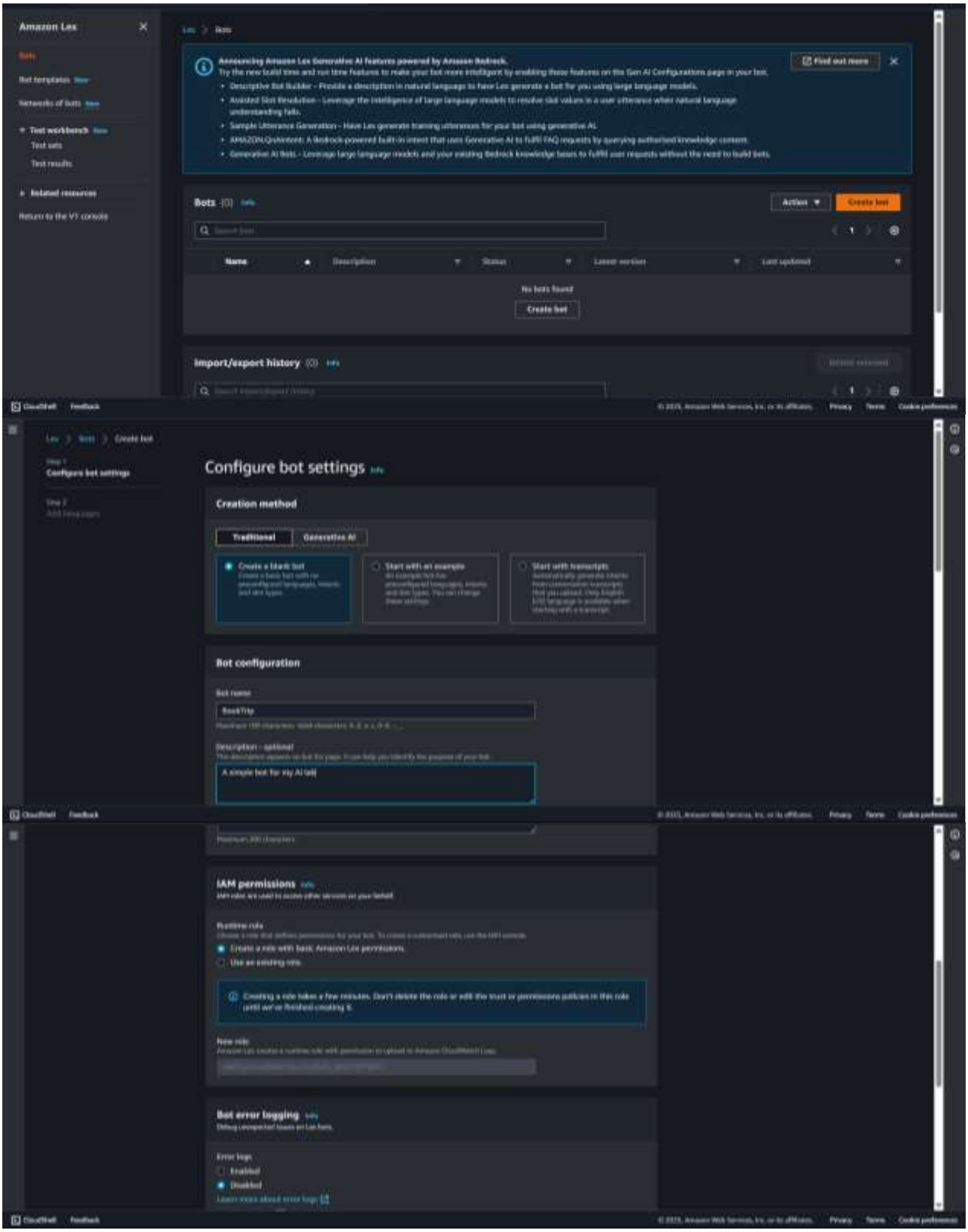
Theory / Concept: Amazon Lex is an AWS service for building conversational interfaces (chatbots) using both voice and text. This lab uses the **Traditional** creation method, which provides direct, manual control over the bot's logic.
The core components we built are:
1. **Intent:** An intent represents 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.

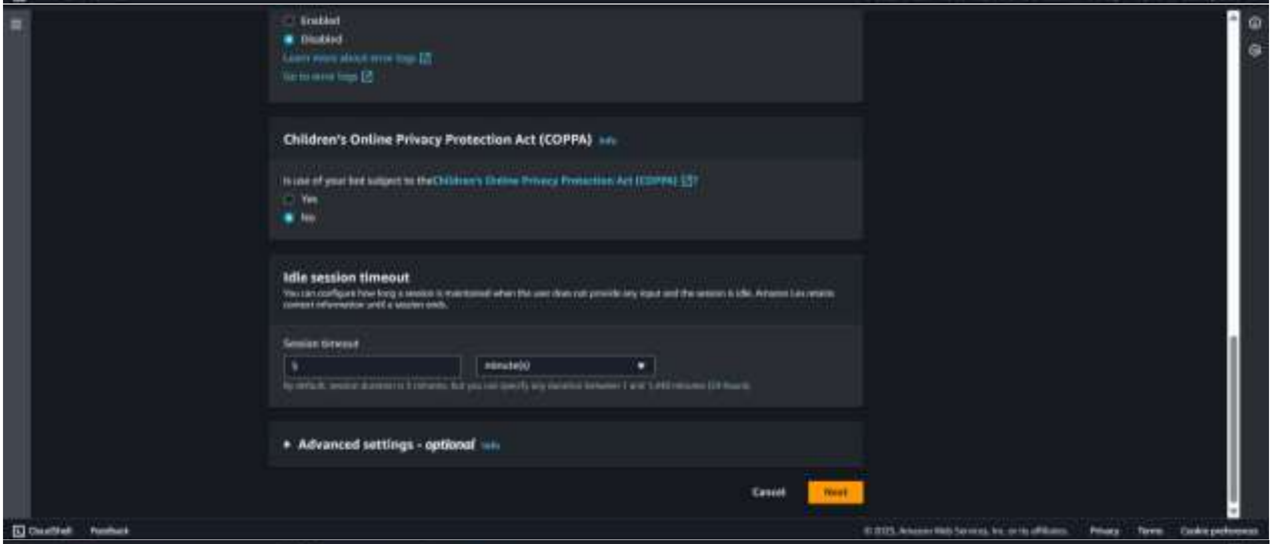
Output:

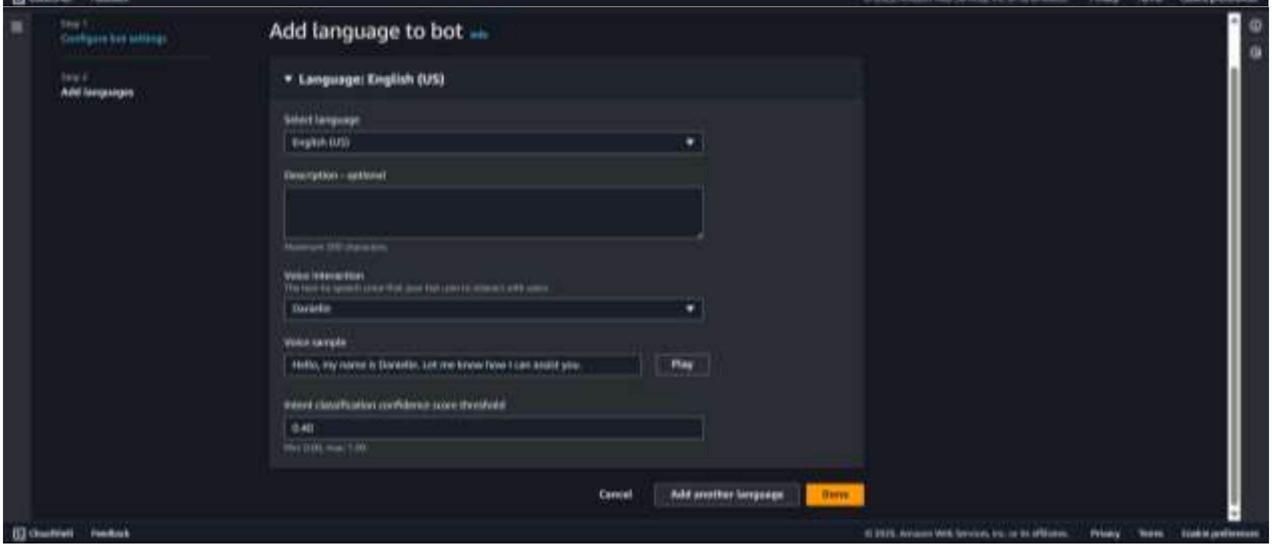
1. Bot Creation: Screenshot of the "Create bot" page, showing the "Traditional" and "Create a blank bot" options selected, with the Bot name set to BookTrip. *(Insert your screenshot of the bot creation settings here.)*
2. Intent & Utterances: Screenshot of the BookTrip intent page, showing the list 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)*

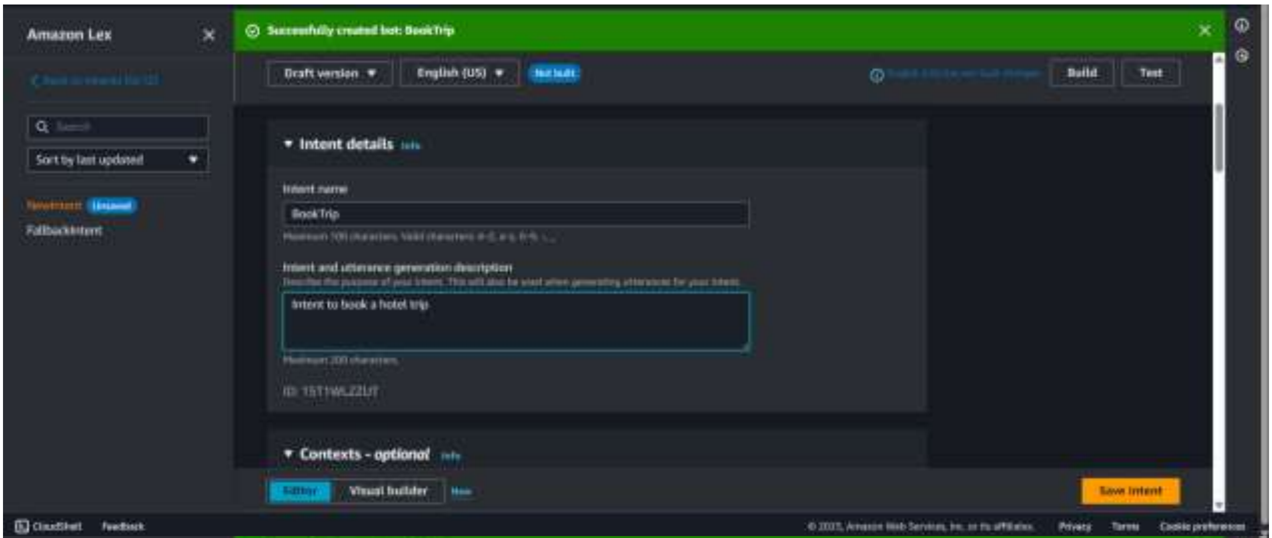
7. IAM Role Deletion: Screenshot of the IAM "Roles" page, showing the search for the bot's role and the "0 matches" result, confirming the role (or search) is gone. *(Insert your screenshot of the IAM Roles page - image_1a9a82.png)*

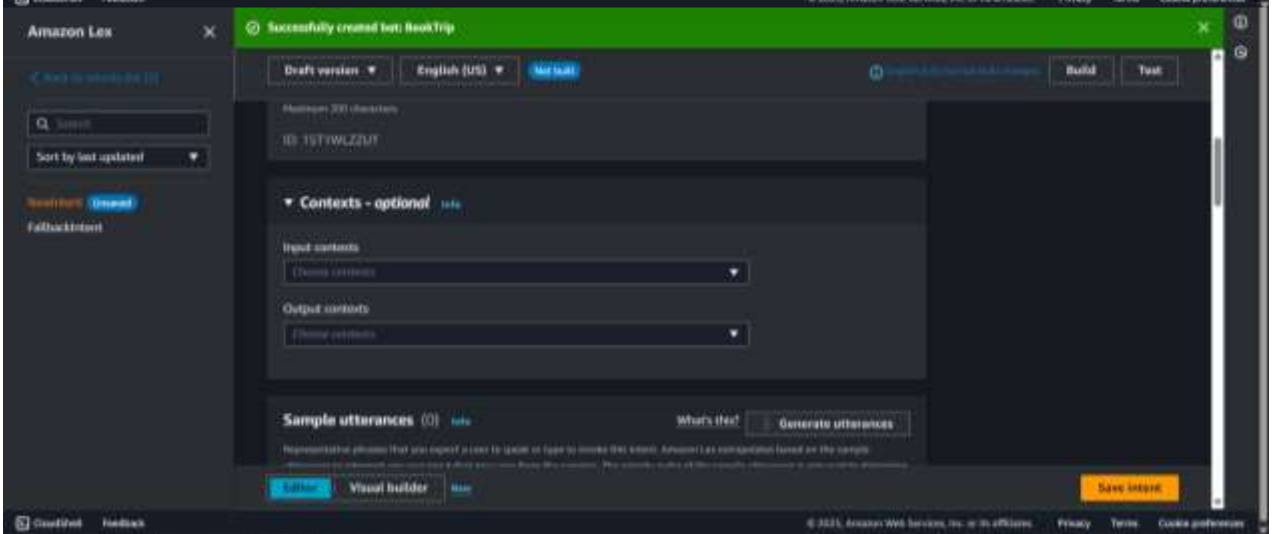


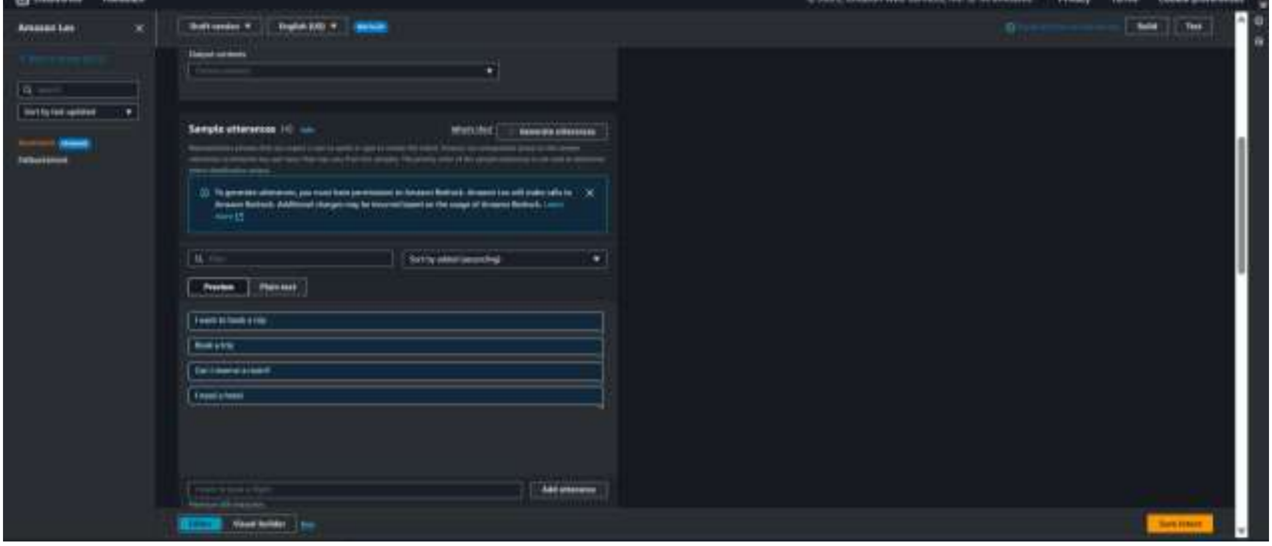

Bot error logging
Error logs
Enabled
Disabled
Children's Online Privacy Protection Act (COPPA)
Yes
No
Idle session timeout
Session timeout
5
minute(s)
CloudShell
Feedback
Privacy
Terms
Cookie preferences

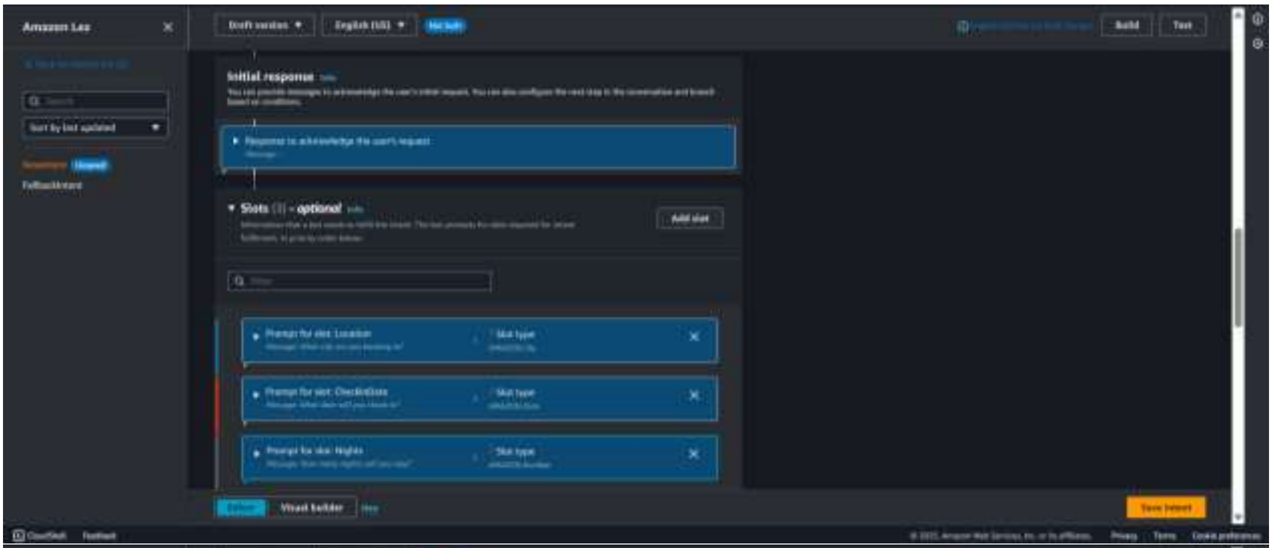

Enabled
Disabled
Children's Online Privacy Protection Act (COPPA)
Yes
No
Idle session timeout
Session timeout
5
minute(s)
Advanced settings - optional
Cancel
Next
CloudShell
Feedback
Privacy
Terms
Cookie preferences

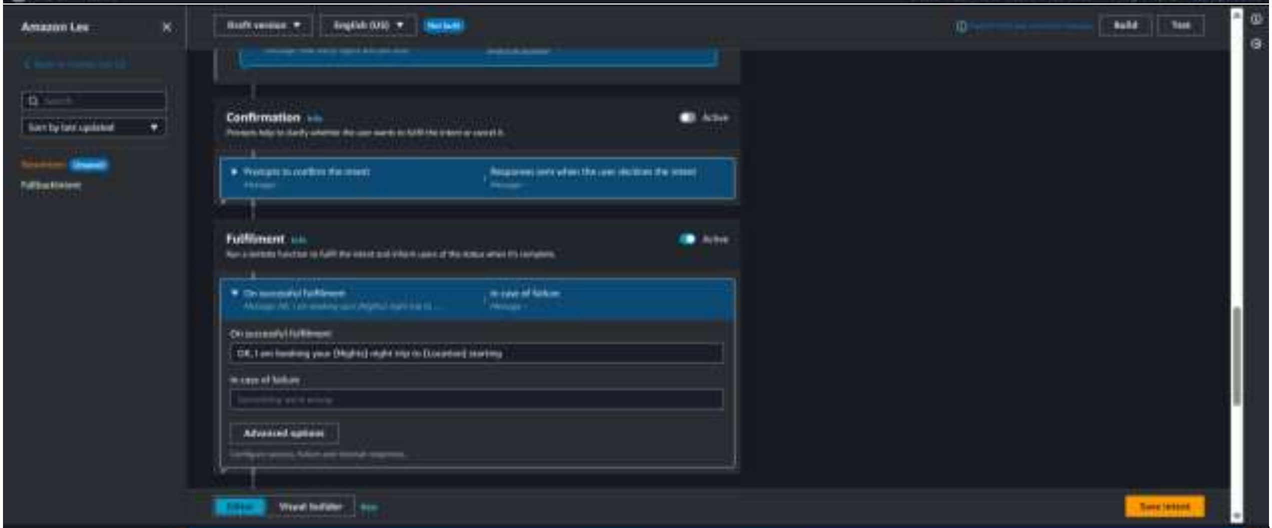

Add language to bot
Language: English (US)
Select language
English (US)
Description - optional
Voice interaction
Danielle
Voice sample
Hello, my name is Danielle. Let me know how I can assist you.
Play
Intent classification confidence score threshold
0.40
Cancel
Add another language
Done
CloudShell
Feedback
Privacy
Terms
Cookie preferences

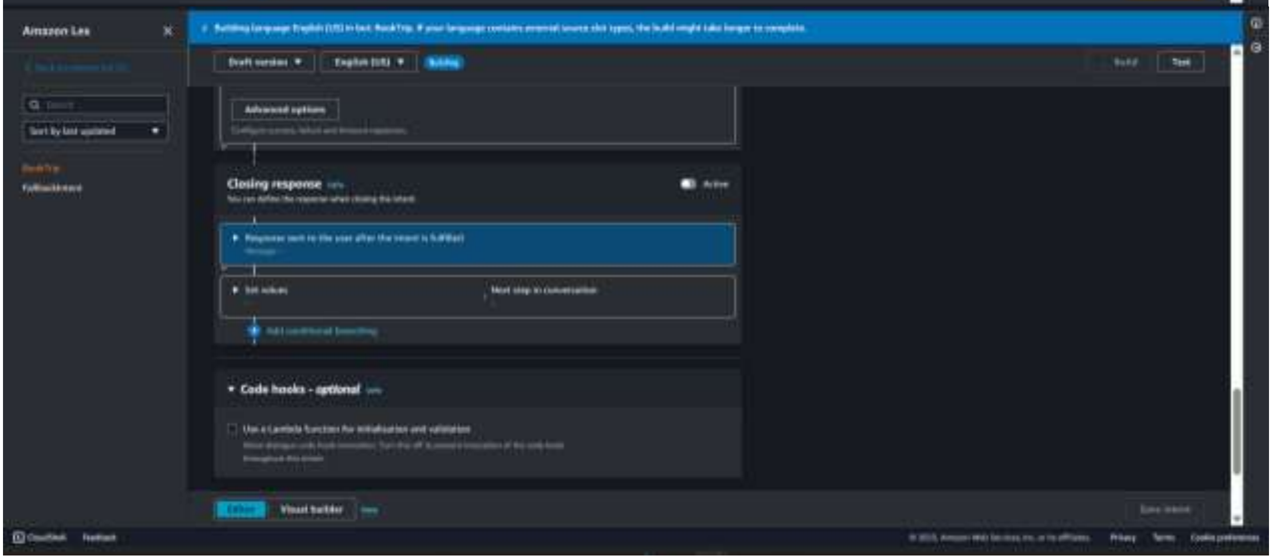

Amazon Lex
Successfully created bot: BookTrip
Draft version
English (US)
Not built
Build
Test
Sort by last updated
FallbackIntent
Intent details
Intent name
BookTrip
Intent and utterance generation description
Intent to book a hotel trip
ID: 1STIWLZZUT
Contexts - optional
Editor
Visual builder
Save intent
CloudShell
Feedback
Privacy
Terms
Cookie preferences

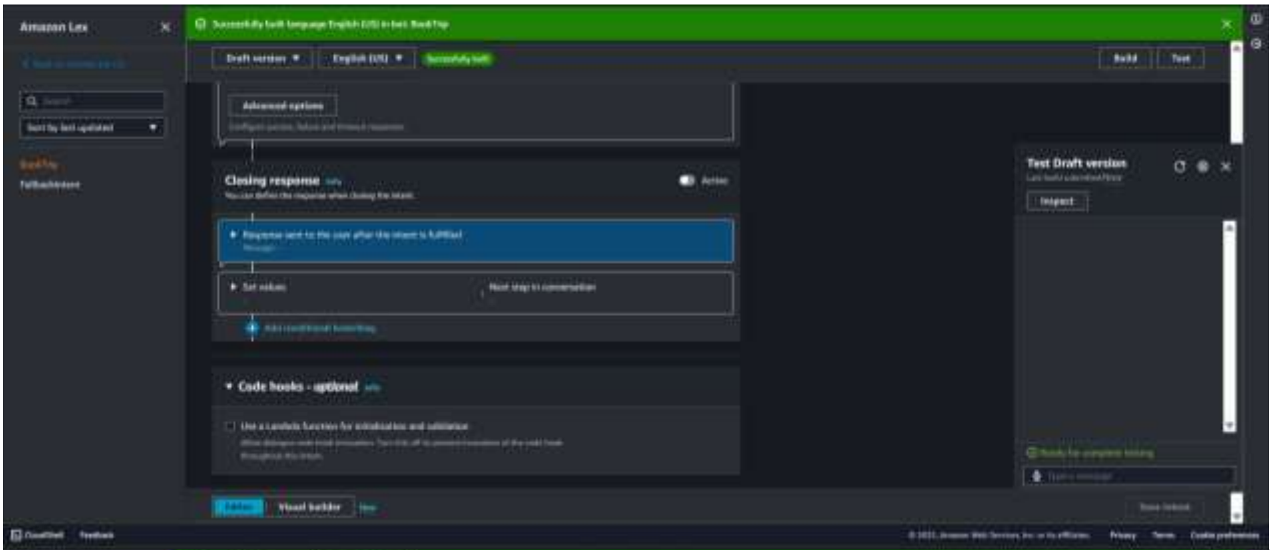

Amazon Lex
Successfully created bot: BookTrip
Draft version
English (US)
Not built
Build
Test
Sort by last updated
FallbackIntent
ID: 1STIWLZZUT
Contexts - optional
Input contexts
Output contexts
Sample utterances
What's this?
Generate utterances
Editor
Visual builder
Save intent
CloudShell
Feedback
Privacy
Terms
Cookie preferences

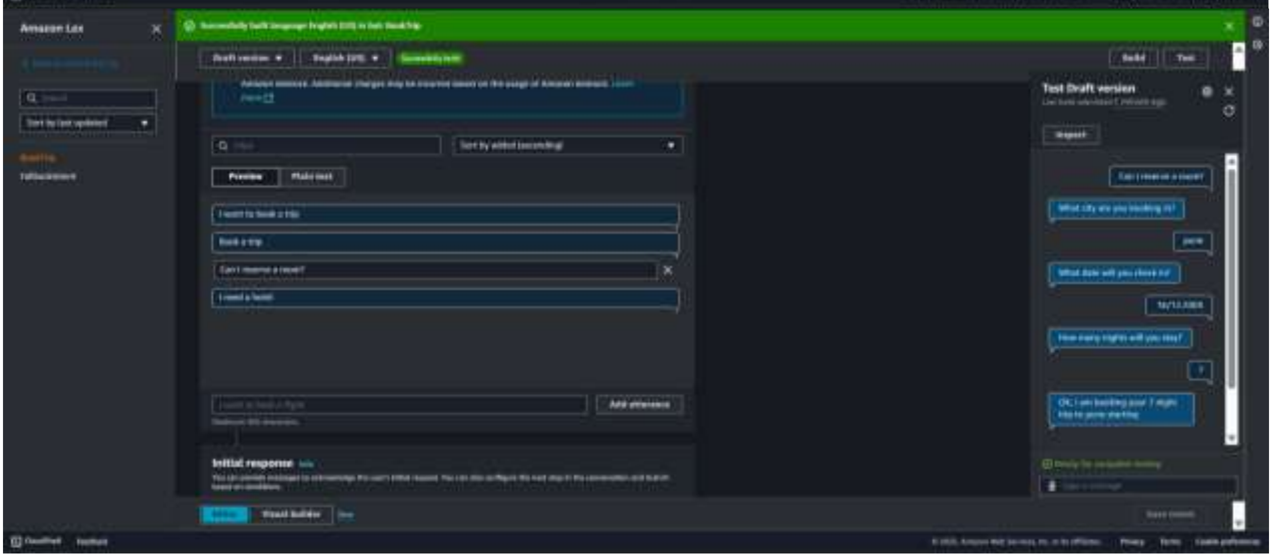



Amazon Lex
Draft version
English (US)
Build
Test
Initial response
Slots (3) - optional
Add slot
Prompt for slot: Location
Prompt for slot: CheckInDate
Prompt for slot: Nights
Slot type
Visual builder
Save intent


Amazon Lex
Draft version
English (US)
Build
Test
Confirmation
Active
Prompts to confirm the intent
Fulfillment
On successful fulfillment
In case of failure
Advanced options
Visual builder
Save intent


Amazon Lex
Draft version
English (US)
Build
Test
Advanced options
Closing response
Active
Code hooks - optional
Visual builder





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.

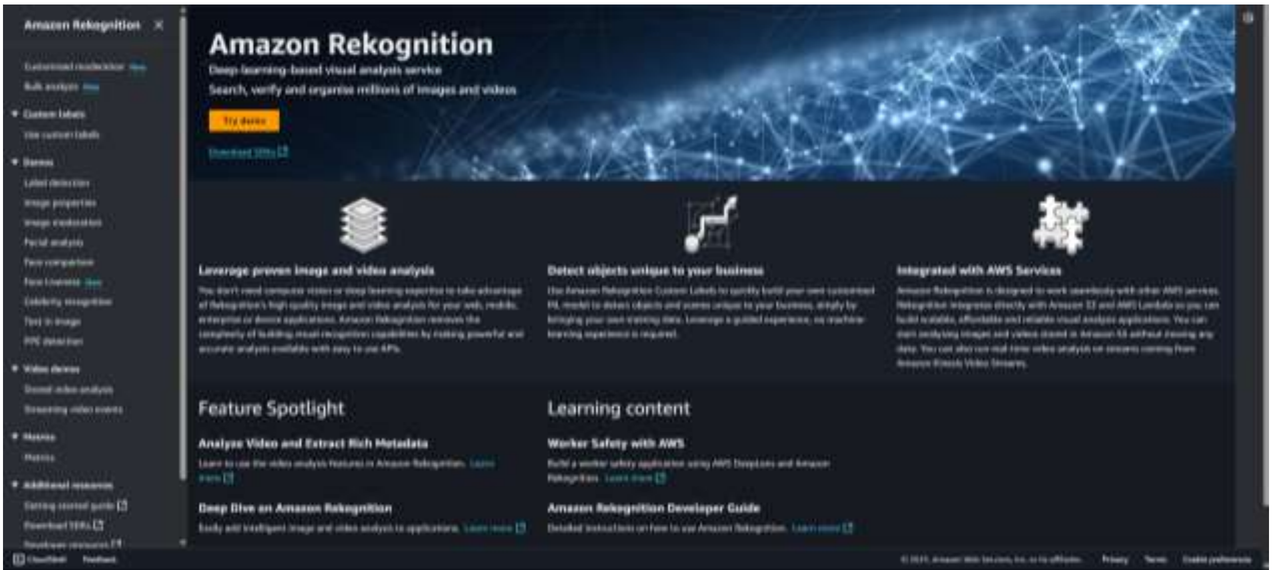
Tools:
- Personal Computer
- Web Browser
- Internet Connection
- A sample image for uploading (e.g., a photo from your phone or a file from the internet)

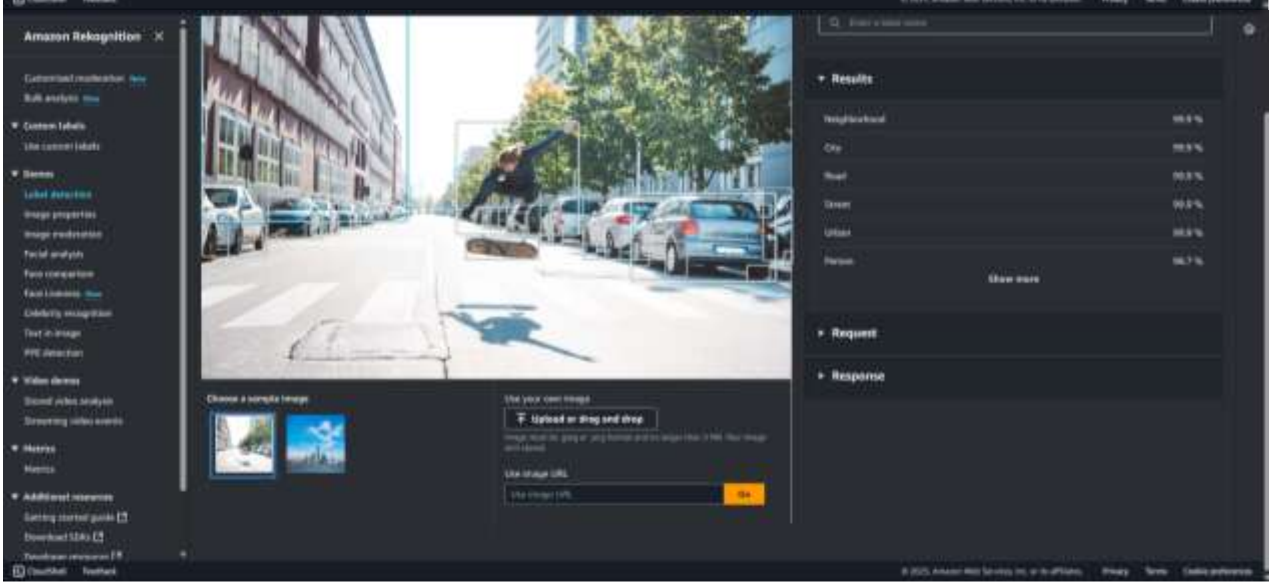
Software Used:
- Amazon Web Services (AWS) Management Console
- Amazon Rekognition

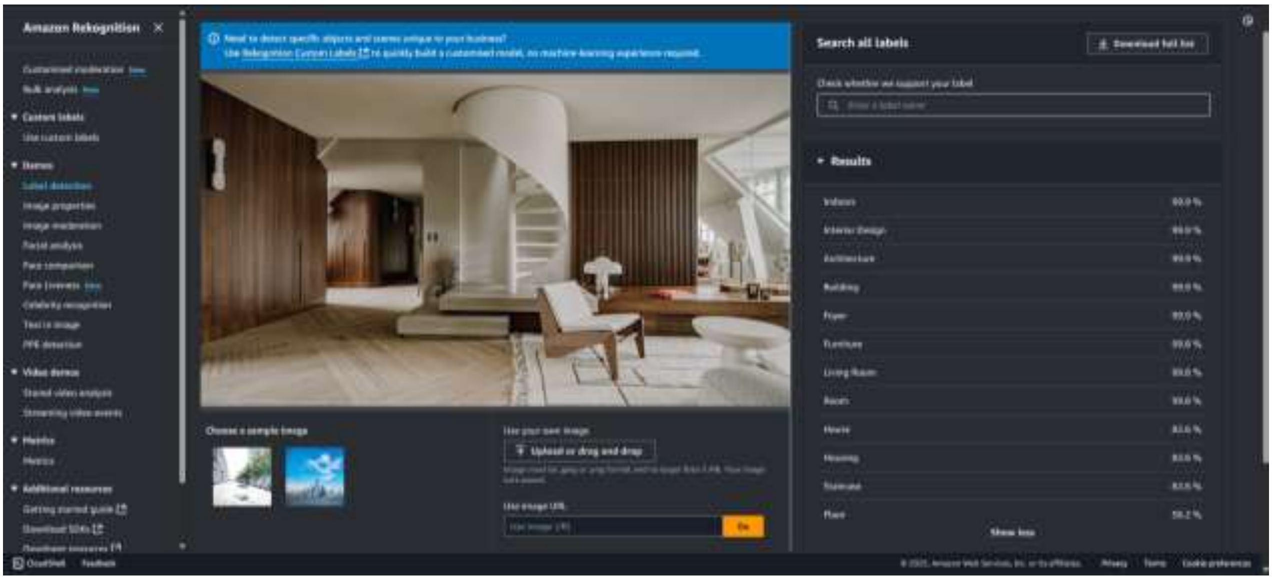
Theory / Concept: Object detection is a computer vision task that identifies and locates objects within an image or video. Amazon Rekognition is a fully managed, deep-learning-based visual analysis service that automates this process. It can detect objects, scenes, activities, faces, text, and more.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







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.