Name of Student: Pushkar Sane		
Roll Number: 45	Lab Assignment Number: 7, 8, 9	
Title of Lab Assignment: XGBoost Algorithm, Data Split for Sagemaker, Hyperparameter Search Optimization, Hyperparameters Optimization Strategies, Bias Variance Tradeoff, L2 Regularization (Ridge Regression), L1 Regularization (Lasso Regression), Hyperparameters Optimization Using GridSearchCV.		
DOP: 21-02-2024	DOS: 28-03-2024	
CO Mapped: CO4	PO Mapped: PO1, PO2, PO3, PO4, PO5, PO6, PO7, PO8, PO9, PO11, PO12, PSO1, PSO2	Signature:

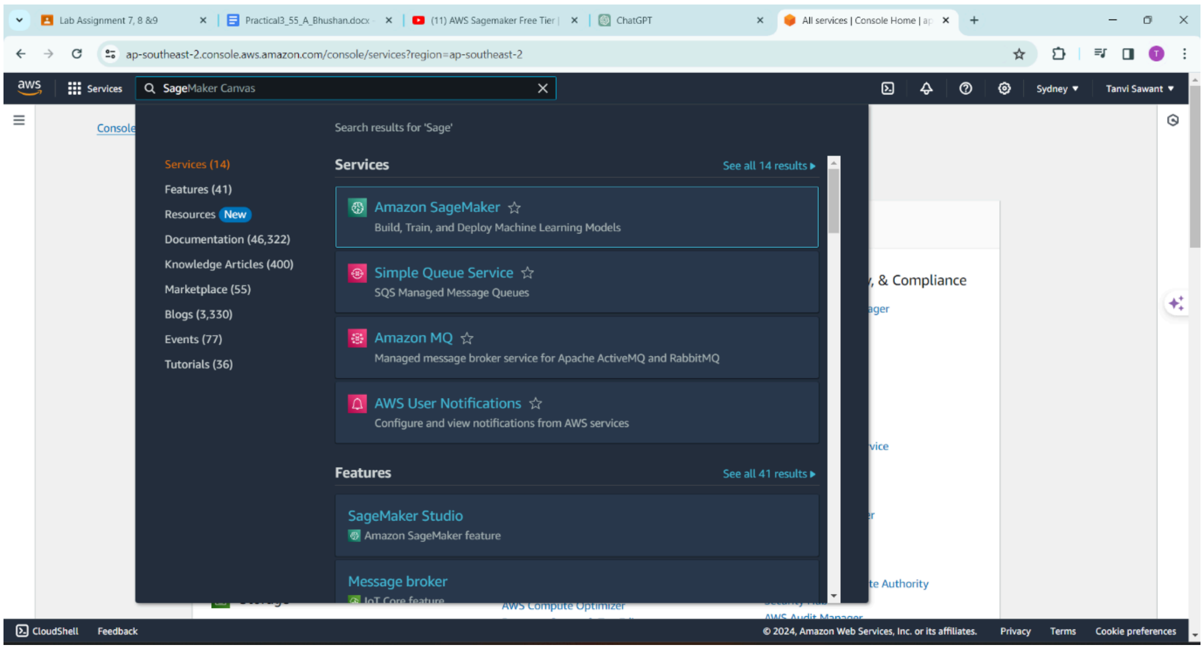
Practical No. 7, 8, 9

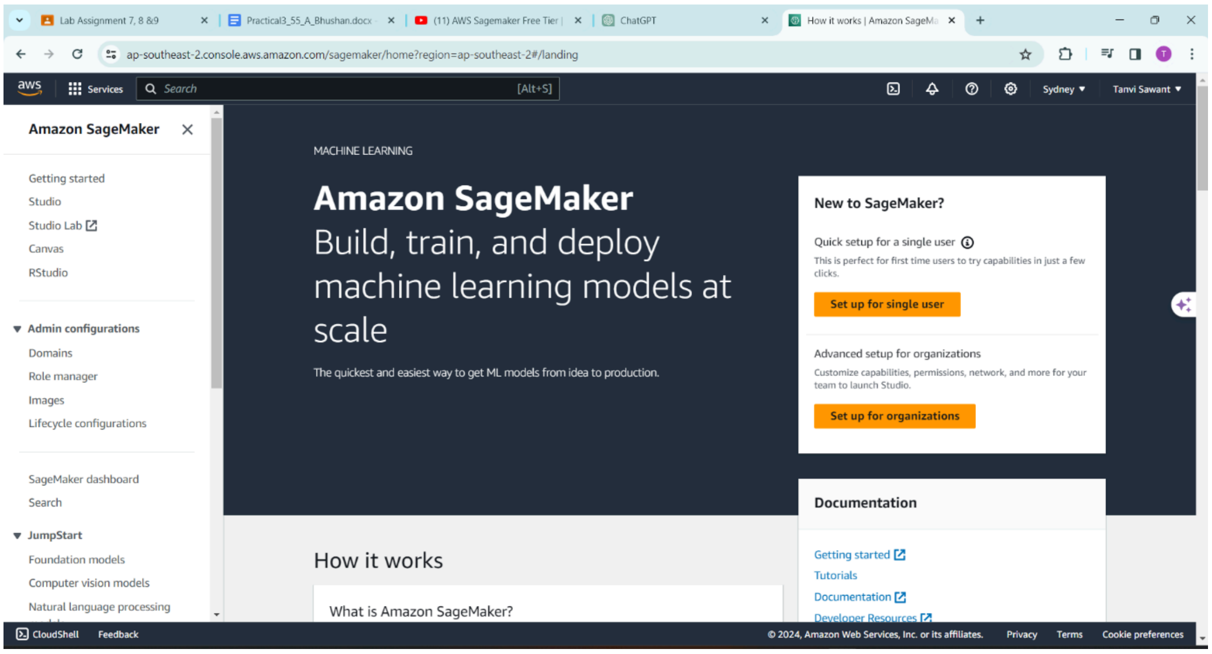
Sign Up for AWS Account:

- Go to the AWS website and sign up for a free tier account if you haven't already.
- Follow the steps to create your account, providing necessary information and setting up your billing preferences.

Navigate to Amazon SageMaker:

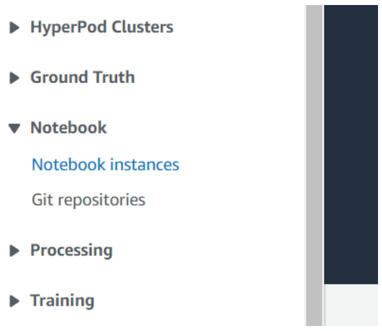
- Once logged in to your AWS Management Console, navigate to the Amazon SageMaker service.
- Click on "Services" in the top-left corner, then type "SageMaker" in the search bar, and click on the SageMaker service.




Amazon SageMaker
Getting started
Studio
Studio Lab
Canvas
RStudio
Admin configurations
Domains
Role manager
Images
Lifecycle configurations
SageMaker dashboard
Search
JumpStart
Foundation models
Computer vision models
Natural language processing
MACHINE LEARNING
Amazon SageMaker
Build, train, and deploy machine learning models at scale
The quickest and easiest way to get ML models from idea to production.
New to SageMaker?
Quick setup for a single user
This is perfect for first time users to try capabilities in just a few clicks.
Set up for single user
Advanced setup for organizations
Customize capabilities, permissions, network, and more for your team to launch Studio.
Set up for organizations
Documentation
Getting started
Tutorials
Documentation
Developer Resources
How it works
What is Amazon SageMaker?

Create a Notebook Instance:

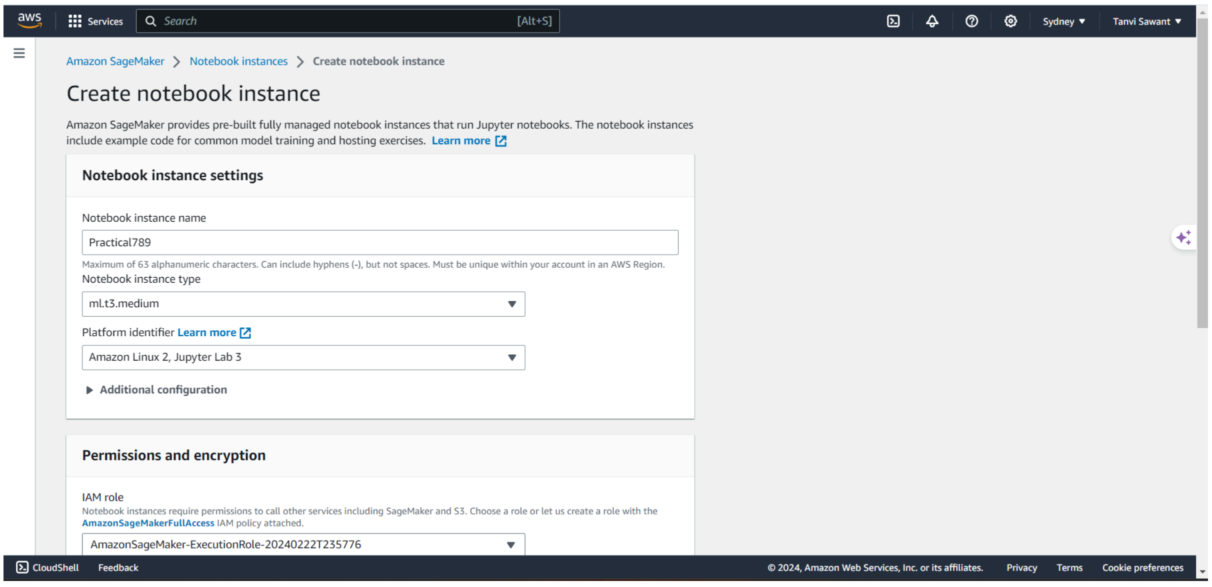
- In the SageMaker dashboard, click on "Notebook instances" in the left sidebar.
- Click on the "Create notebook instance" button.
- Enter a name for your notebook instance, choose an instance type eligible for the free tier (like **ml.t2.medium**), and leave other settings as default.



Configure Notebook Instance:

- In the "Notebook instance settings" section, give your notebook instance a name. Something descriptive like "MyMLNotebook" would work.
- For "Notebook instance type," choose an instance type that is eligible for the free tier, such as **ml.t2.medium**. This ensures you don't incur any charges beyond the free tier limits.

- Leave other settings as default for now. You can explore advanced settings later as needed.


aws
Services
Search [Alt+S]
Sydney
Tanvi Sawant
Amazon SageMaker > Notebook instances > Create notebook instance

Create notebook instance

Amazon SageMaker provides pre-built fully managed notebook instances that run Jupyter notebooks. The notebook instances include example code for common model training and hosting exercises. Learn more

Notebook instance settings

Notebook instance name: Practical789

Maximum of 63 alphanumeric characters. Can include hyphens (-), but not spaces. Must be unique within your account in an AWS Region.

Notebook instance type: ml.t3.medium

Platform identifier Learn more: Amazon Linux 2, Jupyter Lab 3

Additional configuration

Permissions and encryption

IAM role

Notebook instances require permissions to call other services including SageMaker and S3. Choose a role or let us create a role with the AmazonSageMakerFullAccess IAM policy attached.

AmazonSageMaker-ExecutionRole-20240222T235776

Create IAM Role (Optional):

If you haven't created an IAM role before, you might be prompted to do so. This role grants necessary permissions to SageMaker to access other AWS services on your behalf.
Click on "Create a new role" if prompted, and follow the instructions to create a new IAM role. You can give it a name like "SageMakerRole".

Encryption (Optional):

If you want to encrypt the data stored on the notebook instance's EBS volume, you can choose an AWS Key Management Service (KMS) key. This step is optional for the purposes of this tutorial.

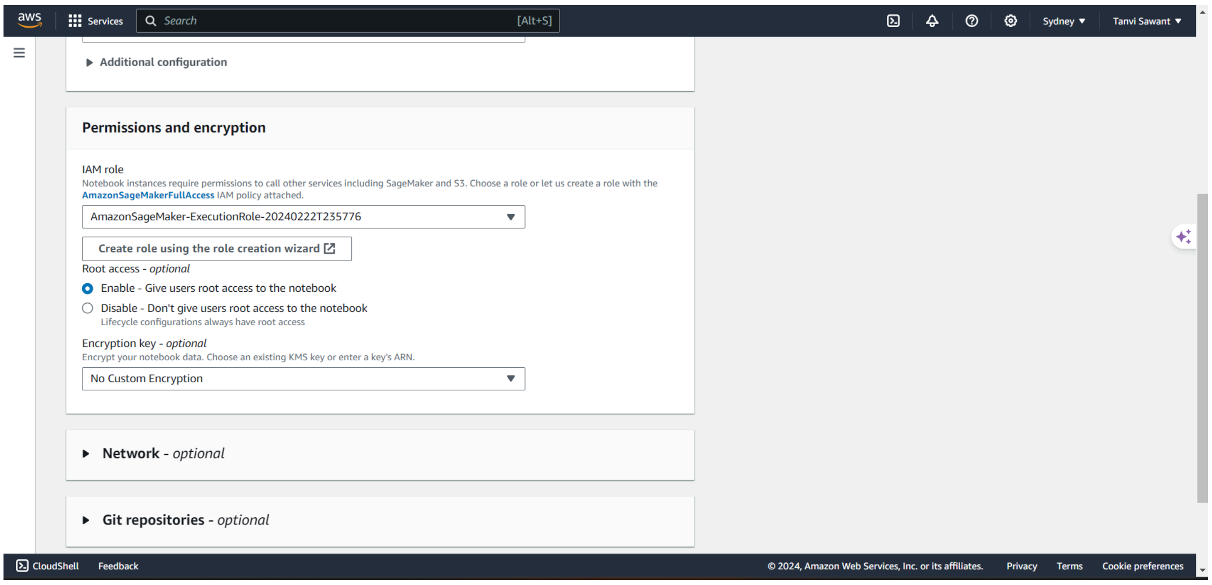
Networking Configuration (Optional):

You can choose a VPC (Virtual Private Cloud) and Subnet for your notebook instance if you have specific networking requirements. For simplicity, you can leave this as default.

Create Notebook Instance:

Once you've configured the settings according to your preferences, scroll down and click on the "Create notebook instance" button at the bottom.

Amazon SageMaker will now provision your notebook instance, which may take a few minutes.



Additional configuration

Permissions and encryption

IAM role
Notebook instances require permissions to call other services including SageMaker and S3. Choose a role or let us create a role with the AmazonSageMakerFullAccess IAM policy attached.

AmazonSageMaker-ExecutionRole-20240222T235776

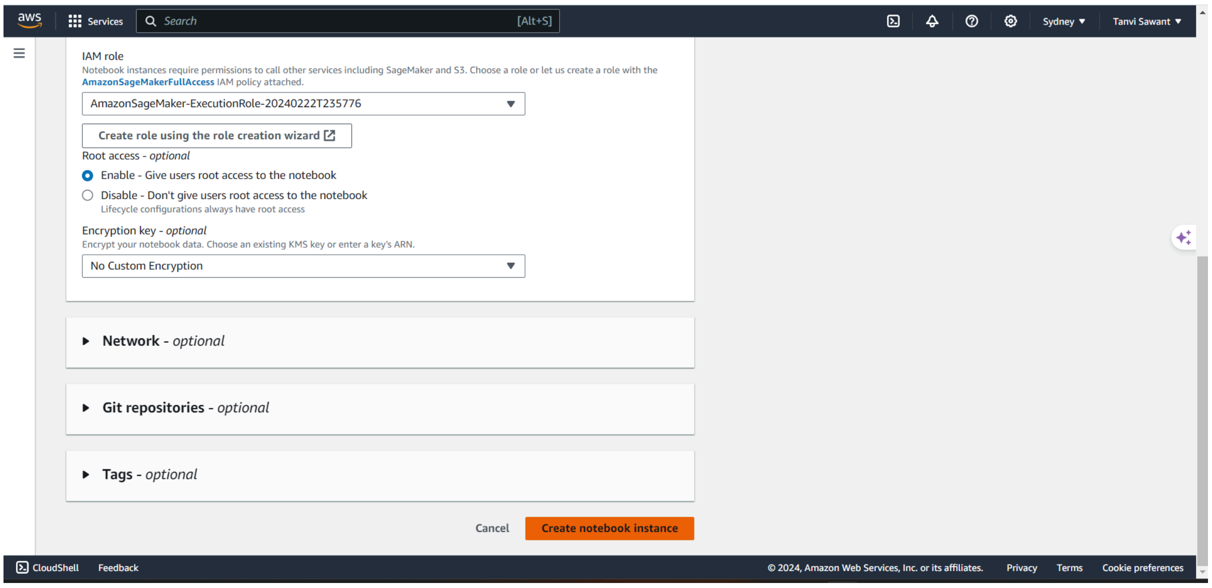
Create role using the role creation wizard

Root access - *optional*
[✓ Enable - Give users root access to the notebook]
Disable - Don't give users root access to the notebook
Lifecycle configurations always have root access

Encryption key - *optional*
Encrypt your notebook data. Choose an existing KMS key or enter a key's ARN.
No Custom Encryption

Network - *optional*

Git repositories - *optional*



IAM role
Notebook instances require permissions to call other services including SageMaker and S3. Choose a role or let us create a role with the AmazonSageMakerFullAccess IAM policy attached.

AmazonSageMaker-ExecutionRole-20240222T235776

Create role using the role creation wizard

Root access - *optional*
[✓ Enable - Give users root access to the notebook]
Disable - Don't give users root access to the notebook
Lifecycle configurations always have root access

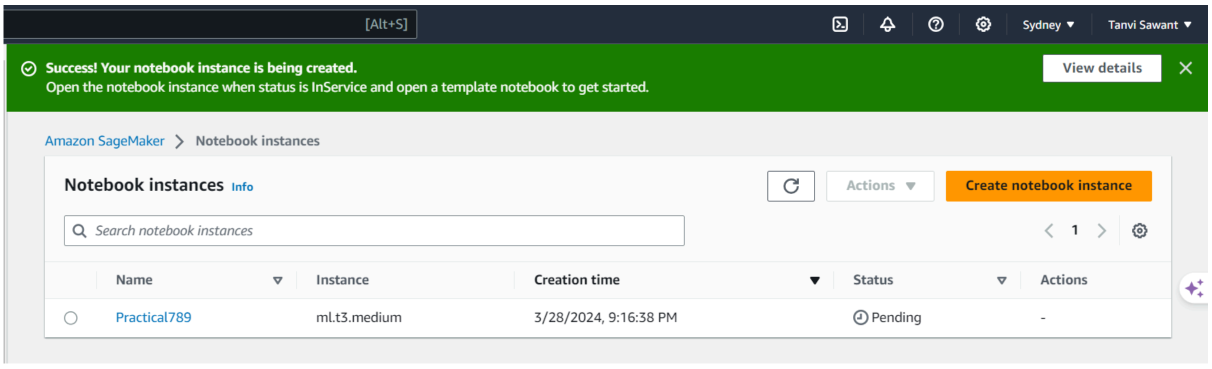
Encryption key - *optional*
Encrypt your notebook data. Choose an existing KMS key or enter a key's ARN.
No Custom Encryption

Network - *optional*

Git repositories - *optional*

Tags - *optional*

Cancel Create notebook instance

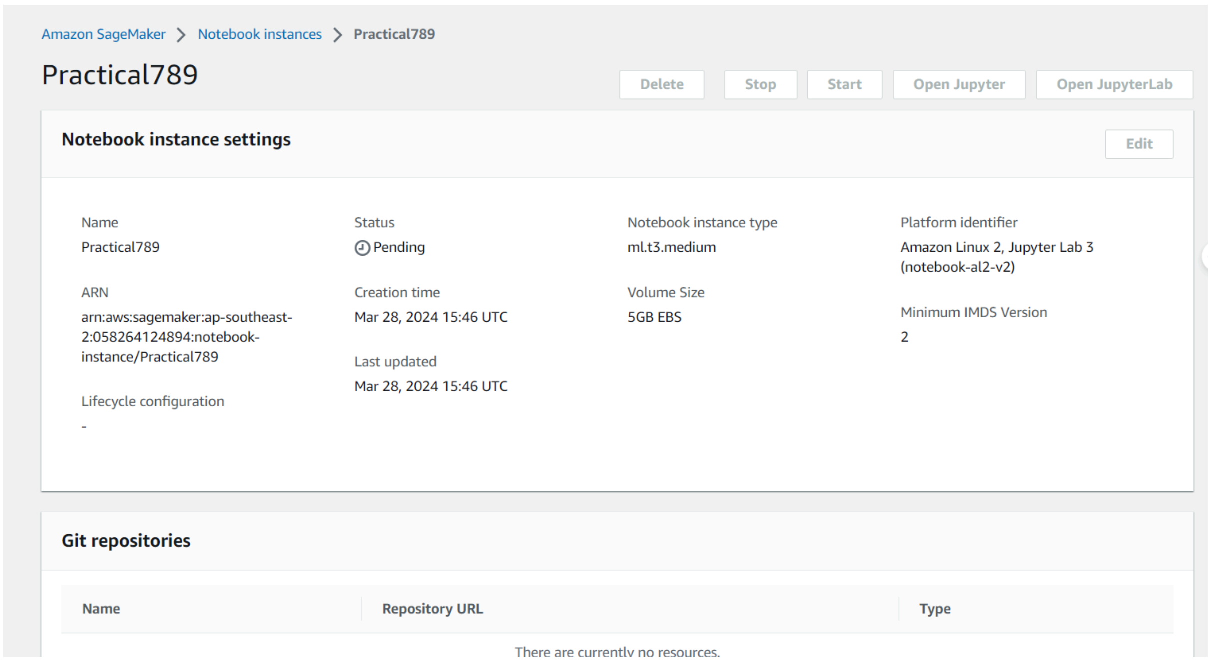


Success! Your notebook instance is being created.
Open the notebook instance when status is InService and open a template notebook to get started.

Amazon SageMaker > Notebook instances

Notebook instances Info

Name	Instance	Creation time	Status	Actions
Practical789	ml.t3.medium	3/28/2024, 9:16:38 PM	Pending	-



Amazon SageMaker > Notebook instances > Practical789

Practical789

Delete | Stop | Start | Open Jupyter | Open JupyterLab

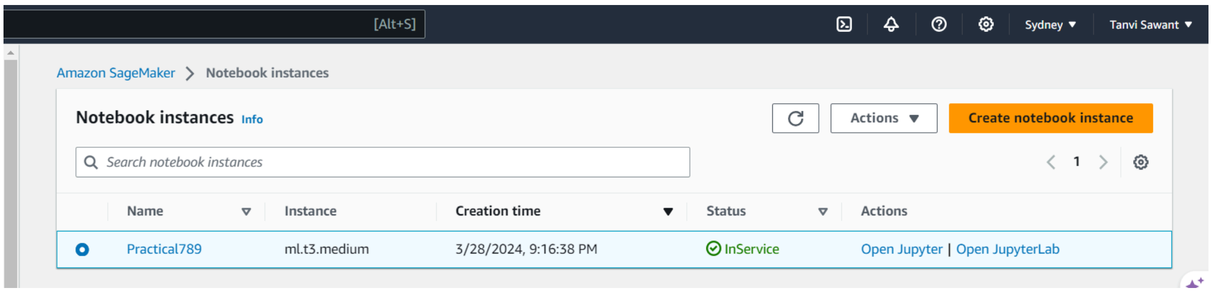
Notebook instance settings Edit

Name	Status	Notebook instance type	Platform identifier
Practical789	Pending	ml.t3.medium	Amazon Linux 2, Jupyter Lab 3 (notebook-al2-v2)
ARN	Creation time	Volume Size	Minimum IMDS Version
arn:aws:sagemaker:ap-southeast-2:058264124894:notebook-instance/Practical789	Mar 28, 2024 15:46 UTC	5GB EBS	2
Lifecycle configuration	Last updated		
-	Mar 28, 2024 15:46 UTC		

Git repositories

Name	Repository URL	Type

There are currently no resources.



Amazon SageMaker > Notebook instances

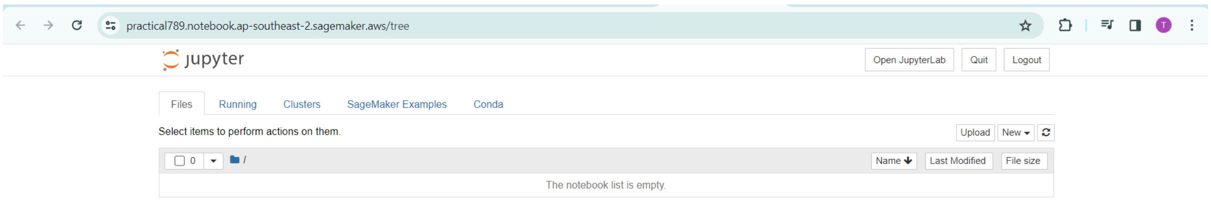
Notebook instances Info

Search notebook instances

	Name	Instance	Creation time	Status	Actions
●	Practical789	ml.t3.medium	3/28/2024, 9:16:38 PM	InService	Open Jupyter | Open JupyterLab

Open Jupyter Notebook:

- Once the notebook instance is created (it may take a few minutes), click on "Open Jupyter" next to your notebook instance name.
- This will open a Jupyter Notebook interface in a new tab.



practical789.notebook.ap-southeast-2.sagemaker.aws/tree

Select items to perform actions on them.

The notebook list is empty.

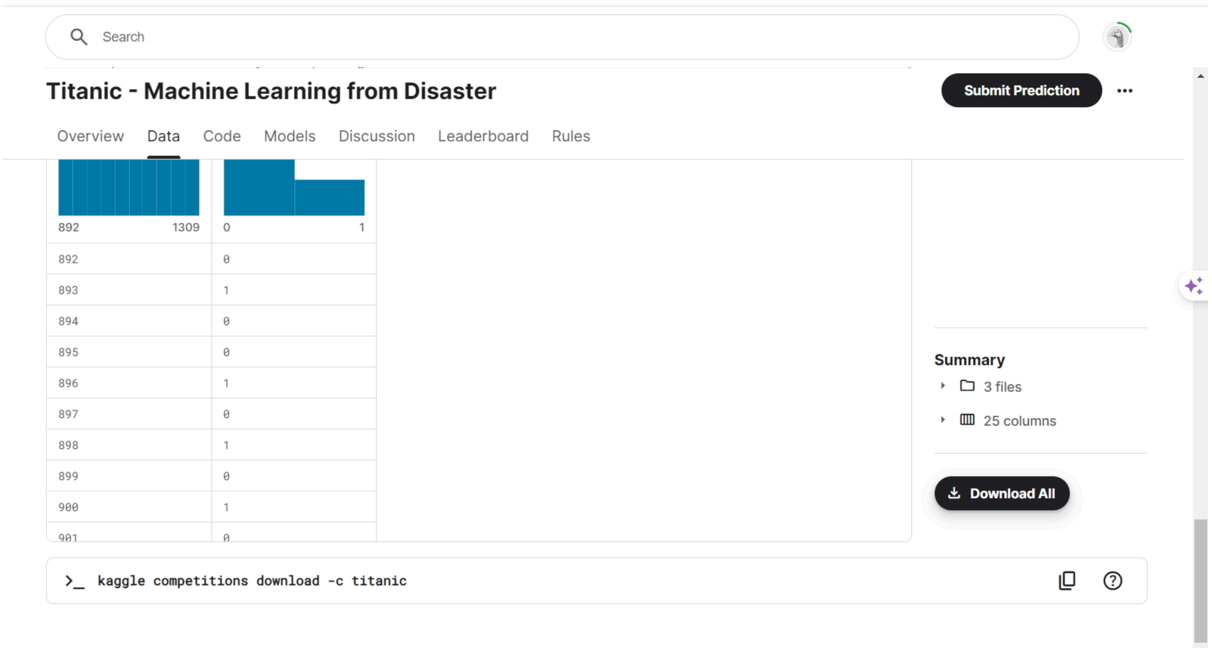
Dataset: Titanic: Machine Learning from Disaster

This dataset contains information about passengers aboard the Titanic, including details like age, gender, passenger class, ticket fare, and whether they survived the disaster. It's commonly used for classification tasks and is suitable for practicing data preprocessing, model building, and evaluation.

Here are the steps to download the dataset from Kaggle:

1. Sign in to your Kaggle account or create a new one.
2. Navigate to the Titanic competition page.
3. Scroll down to the "Data" section.
4. Click on the "Download All" button to download the entire dataset as a zip file.
5. Extract the zip file to access the dataset files (e.g., train.csv, test.csv).

Once you have the dataset files, you can upload them to your Jupyter Notebook environment on AWS SageMaker as explained earlier. This dataset is well-suited for practicing data preprocessing, model training, and evaluation tasks.

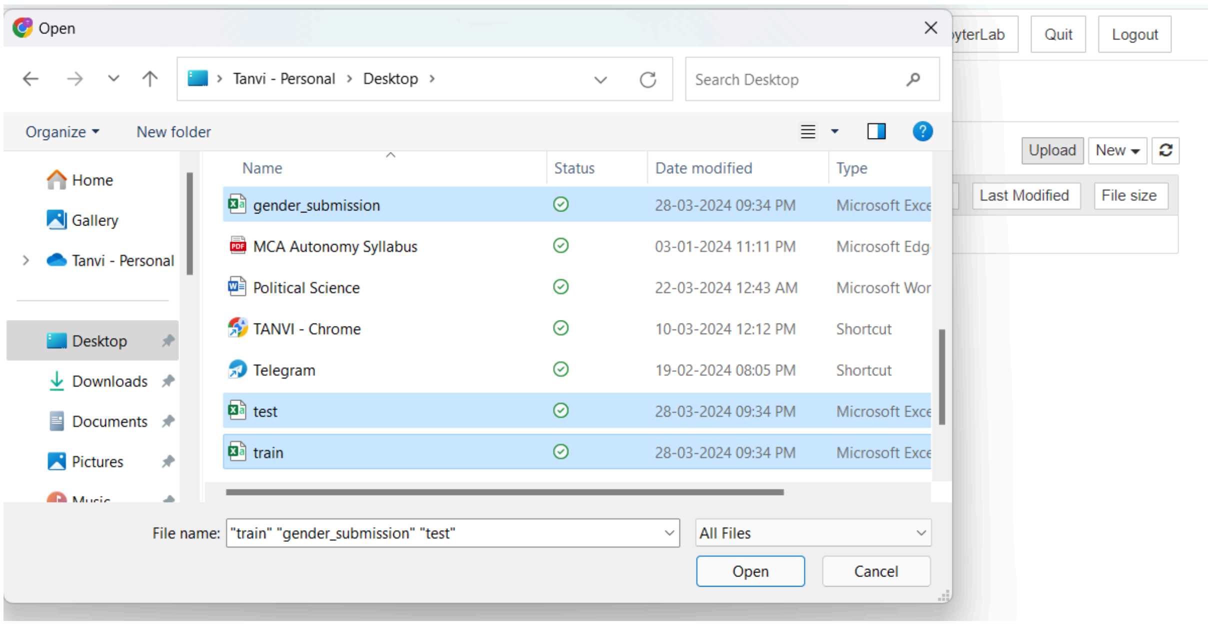


892 … 1309	0 … 1
892	0
893	1
894	0
895	0
896	1
897	0
898	1
899	0
900	1
901	0

```
>_ kaggle competitions download -c titanic
```

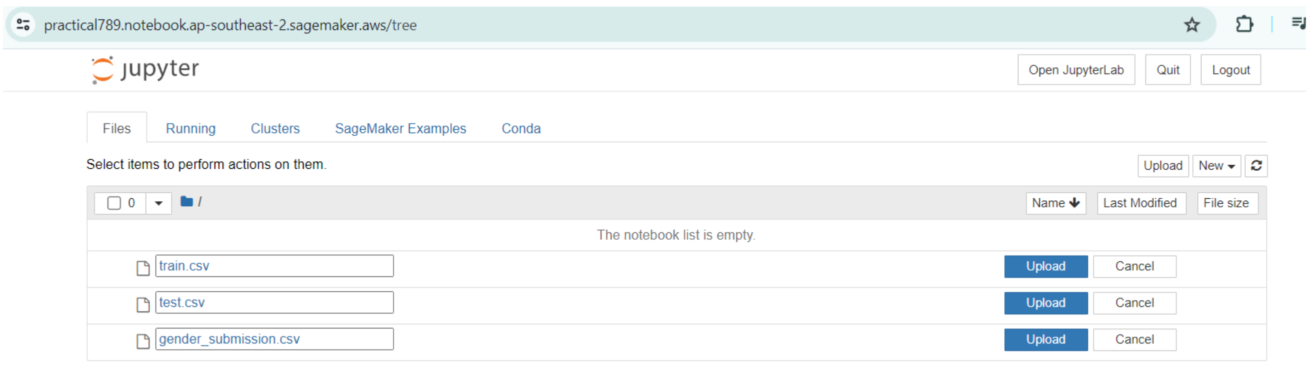
Prepare Your Data:

- Upload the dataset files (**train.csv**, **test.csv**, **gender_submission.csv**) to your Jupyter Notebook environment.



Name	Status	Date modified	Type
gender_submission	✓	28-03-2024 09:34 PM	Microsoft Exce
MCA Autonomy Syllabus	✓	03-01-2024 11:11 PM	Microsoft Edg
Political Science	✓	22-03-2024 12:43 AM	Microsoft Wor
TANVI - Chrome	✓	10-03-2024 12:12 PM	Shortcut
Telegram	✓	19-02-2024 08:05 PM	Shortcut
test	✓	28-03-2024 09:34 PM	Microsoft Exce
train	✓	28-03-2024 09:34 PM	Microsoft Exce

File name: "train" "gender_submission" "test"   All Files

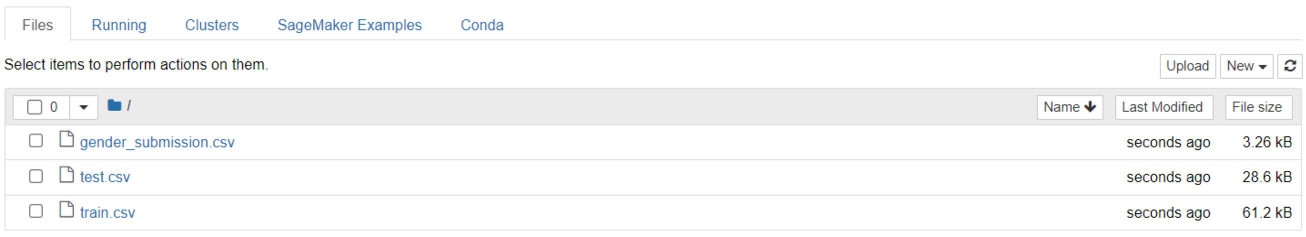


practical789.notebook.ap-southeast-2.sagemaker.aws/tree

Select items to perform actions on them.

The notebook list is empty.

train.csv	Upload	Cancel
test.csv	Upload	Cancel
gender_submission.csv	Upload	Cancel



Select items to perform actions on them.

Name	Last Modified	File size
gender_submission.csv	seconds ago	3.26 kB
test.csv	seconds ago	28.6 kB
train.csv	seconds ago	61.2 kB

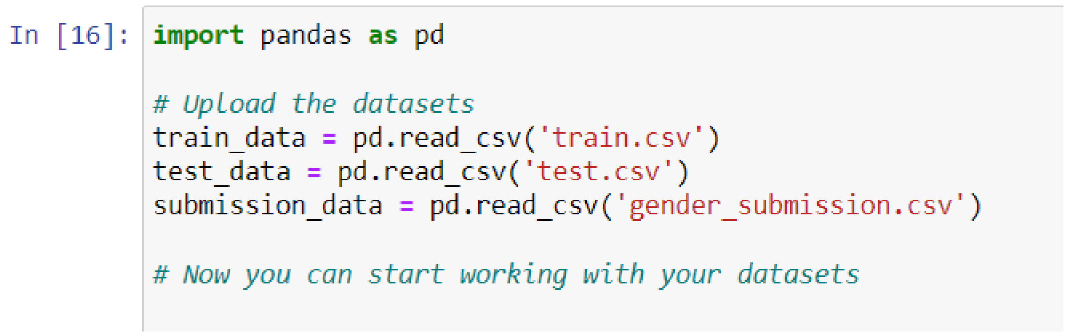
- Use pandas to load the datasets into separate DataFrames.
- Perform any necessary data cleaning and preprocessing steps.

Create a New Notebook:

- In the Jupyter Notebook interface, click on the "New" button on the top right corner.
- Select "Python 3" to create a new Python notebook.

Prepare Your Data:

- In this step, you'll upload your datasets (test.csv, train.csv, gender_submission.csv) to your Jupyter Notebook environment and start working with them.



```
In [16]: import pandas as pd

         # Upload the datasets
         train_data = pd.read_csv('train.csv')
         test_data = pd.read_csv('test.csv')
         submission_data = pd.read_csv('gender_submission.csv')

         # Now you can start working with your datasets
```

What is XGBoost Algorithm?

XGBoost is a robust machine-learning algorithm that can help you understand your data and make better decisions.

XGBoost is an implementation of gradient-boosting decision trees. It has been used by data scientists and researchers worldwide to optimize their machine-learning models.

What is XGBoost in Machine Learning?

XGBoost is designed for speed, ease of use, and performance on large datasets. It does not require optimization of the parameters or tuning, which means that it can be used immediately after installation without any further configuration.

XGBoost Features

XGBoost is a widespread implementation of gradient boosting.

❖ XGBoost offers regularization, which allows you to control overfitting by introducing L1/L2 penalties on the weights and biases of each tree.

❖ Another feature of XGBoost is its ability to handle sparse data sets using the weighted quantile sketch algorithm. This algorithm allows us to deal with non-zero entries in the feature matrix while retaining the same computational complexity as other algorithms.

❖ XGBoost also has a block structure for parallel learning. It makes it easy to scale up on multicore machines or clusters. It also uses cache awareness, which helps reduce memory usage when training models with large datasets.

❖ XGBoost offers out-of-core computing capabilities using disk-based data structures instead of in-memory ones during the computation phase.

XgBoost Formula

The XGBoost algorithm builds an ensemble of decision trees in a sequential manner, where each subsequent tree corrects the errors made by the previous ones. The final prediction is a weighted sum of the predictions from all the trees in the ensemble. The general formula for predicting the output of a new data point using XGBoost can be expressed as follows

$$\hat{y}_i = \phi(\mathbf{x}_i) = \sum_{k=1}^{K} f_k(\mathbf{x}_i)$$

where:

- $\hat{y}_i$ is the predicted output for the i-th data point.
- $\mathbf{x}_i$ represents the feature vector for the i-th data point.
- K is the total number of trees in the ensemble.
- $f_k(\mathbf{x}_i)$ is the prediction made by the k-th tree for the i-th data point.

Conclusion: Overall, XGBoost is widely used in both academia and industry due to its excellent performance, versatility, and efficiency in handling various machine learning tasks.

Performing data splitting, hyperparameter search optimization, and hyperparameter optimization strategies in AWS SageMaker involves several steps. Below is a detailed guide on how to perform these tasks:

1. Data Split for SageMaker.

Before training a machine learning model, it is essential to split the dataset into training, validation, and test sets. SageMaker provides utilities to perform this data split.

```
import sagemaker
from sagemaker import get_execution_role
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
```

```
# Assume you have a dataframe df with features and target
# Split data into train, validation, and test sets
train_data, temp_data = train_test_split(df, test_size=0.3, random_state=42)
val_data, test_data = train_test_split(temp_data, test_size=0.5, random_state=42)

# Save the data to CSV files
train_data.to_csv('train.csv', index=False, header=False)
val_data.to_csv('validation.csv', index=False, header=False)
test_data.to_csv('test.csv', index=False, header=False)

# Upload the data to S3
role = get_execution_role()
sess = sagemaker.Session()
bucket = sess.default_bucket()

train_data_s3_path = sess.upload_data(path='train.csv', bucket=bucket, key_prefix='data')
val_data_s3_path = sess.upload_data(path='validation.csv', bucket=bucket,
key_prefix='data')
test_data_s3_path = sess.upload_data(path='test.csv', bucket=bucket, key_prefix='data')
```

2. Hyperparameter Search Optimization

SageMaker provides built-in hyperparameter optimization (HPO) to automate the process of finding the best hyperparameters for your model.

```
from sagemaker.tuner import HyperparameterTuner, IntegerParameter,
ContinuousParameter

# Define hyperparameter ranges
hyperparameter_ranges = {
    'learning_rate': ContinuousParameter(0.001, 0.1),
    'num_trees': IntegerParameter(50, 200),
    'max_depth': IntegerParameter(5, 15)
}
```

```
# Create an estimator
estimator = sagemaker.estimator.Estimator(image_uri='image-uri', # specify your image uri
                                          role=role,
                                          instance_count=1,
                                          instance_type='ml.m5.large',
                                          hyperparameters={'objective': 'regression'},
                                          output_path='s3://{}/output'.format(bucket))

# Create a HyperparameterTuner object
tuner = HyperparameterTuner(estimator=estimator,
                            objective_metric_name='validation:rmse', # specify the metric to optimize
                            objective_type='Minimize',
                            hyperparameter_ranges=hyperparameter_ranges,
                            max_jobs=20,
                            max_parallel_jobs=4)

# Fit the tuner
tuner.fit({'train': train_data_s3_path, 'validation': val_data_s3_path})
```

3. Hyperparameter Optimization Strategies

AWS SageMaker's HPO supports multiple optimization strategies, such as random search, Bayesian optimization, and more. The strategy parameter in the HyperparameterTuner can be used to specify the optimization strategy.

1. Random Search

Random search is a simple and effective method for hyperparameter optimization. SageMaker's HyperparameterTuner supports random search as one of its optimization strategies.

```
from sagemaker.tuner import HyperparameterTuner, ContinuousParameter, IntegerParameter

# Define hyperparameter ranges
hyperparameter_ranges = {
    'learning_rate': ContinuousParameter(0.001, 0.1),
    'num_trees': IntegerParameter(50, 200),
    'max_depth': IntegerParameter(5, 15)
}
```

```
# Create an estimator
estimator = sagemaker.estimator.Estimator(image_uri='image-uri', # specify your image uri
                                          role=role,
                                          instance_count=1,
                                          instance_type='ml.m5.large',
                                          hyperparameters={'objective': 'regression'},
                                          output_path='s3://{}/output'.format(bucket))

# Create a HyperparameterTuner object with random search strategy
tuner_random = HyperparameterTuner(estimator=estimator,
                                   objective_metric_name='validation:rmse',
                                   objective_type='Minimize',
                                   hyperparameter_ranges=hyperparameter_ranges,
                                   max_jobs=20,
                                   max_parallel_jobs=4,
                                   strategy='Random')

# Fit the tuner
tuner_random.fit({'train': train_data_s3_path, 'validation': val_data_s3_path})
```

2. Bayesian Optimization

Bayesian optimization is an advanced method for hyperparameter optimization, which builds a probabilistic model of the objective function and uses it to select the most promising hyperparameters

```
# Create a HyperparameterTuner object with Bayesian optimization strategy
tuner_bayesian = HyperparameterTuner(estimator=estimator,
                                     objective_metric_name='validation:rmse',
                                     objective_type='Minimize',
                                     hyperparameter_ranges=hyperparameter_ranges,
                                     max_jobs=20,
                                     max_parallel_jobs=4,
                                     strategy='Bayesian')

# Fit the tuner
tuner_bayesian.fit({'train': train_data_s3_path, 'validation': val_data_s3_path})
```

3. Customized Optimization Strategy

You can also implement a customized optimization strategy by subclassing the HyperparameterTunerStrategy class and implementing the required methods.

```
from sagemaker.tuner import HyperparameterTunerStrategy

class CustomStrategy(HyperparameterTunerStrategy):
  def hyperparameter_tuning_job_config(self):
    config = super(CustomStrategy, self).hyperparameter_tuning_job_config()
    config['HyperParameterTuningJobConfig']['Strategy'] = 'Custom'
    return config

# Create a HyperparameterTuner object with custom strategy
tuner_custom = HyperparameterTuner(estimator=estimator,
                                   objective_metric_name='validation:rmse',
                                   objective_type='Minimize',
                                   hyperparameter_ranges=hyperparameter_ranges,
                                   max_jobs=20,
                                   max_parallel_jobs=4,
                                   strategy=CustomStrategy)

# Fit the tuner
tuner_custom.fit({'train': train_data_s3_path, 'validation': val_data_s3_path})
```

4. Bias-Variance Tradeoff

The bias-variance tradeoff is a fundamental concept in machine learning that relates to the model's ability to generalize to unseen data. A high-bias model is too simplistic and may underfit the training data (high training error), whereas a high-variance model is too complex and may overfit the training data (low training error but high validation error).

To manage the bias-variance tradeoff, you can adjust hyperparameters and model complexity. SageMaker's Hyperparameter Optimization (HPO) can help find the optimal hyperparameters that balance bias and variance by searching for the hyperparameter values that result in the best validation performance.

1. Define Hyperparameter Ranges

```
from sagemaker.tuner import HyperparameterTuner, ContinuousParameter, IntegerParameter

# Define hyperparameter ranges
hyperparameter_ranges = {
   'learning_rate': ContinuousParameter(0.001, 0.1),
   'num_trees': IntegerParameter(50, 200),
   'max_depth': IntegerParameter(5, 15)
}
```

2. Create an Estimator

```
import sagemaker

# Create an estimator
estimator = sagemaker.estimator.Estimator(image_uri='image-uri', # specify your image uri
                                          role=role,
                                          instance_count=1,
                                          instance_type='ml.m5.large',
                                          hyperparameters={'objective': 'regression'},
                                          output_path='s3://{}/output'.format(bucket))
```

3. Create a HyperparameterTuner Object

```
# Create a HyperparameterTuner object
tuner = HyperparameterTuner(estimator=estimator,
                            objective_metric_name='validation:rmse',
                            objective_type='Minimize',
                            hyperparameter_ranges=hyperparameter_ranges,
                            max_jobs=20,
                            max_parallel_jobs=4,
                                strategy='Bayesian')  # Use Bayesian optimization as the optimization
strategy
```

4. Fit the HyperparameterTuner

```
# Fit the tuner
tuner.fit({'train': train_data_s3_path, 'validation': val_data_s3_path})
```

5. Analyze the Results

After the hyperparameter tuning job is completed, you can analyze the results to understand how different hyperparameters affect the bias-variance tradeoff. You can use the Amazon SageMaker console or SDK to analyze the results and identify the optimal hyperparameters that balance bias and variance.

```
# Get the best hyperparameters
best_hyperparameters = tuner.best_estimator().hyperparameters()

print("Best hyperparameters:")
print(best_hyperparameters)
```

5. L2 Regularization (Ridge Regression) and L1 Regularization (Lasso Regression)

```
hyperparameters={
    'alpha': 1.0,  # L2 regularization parameter
    'objective': 'reg:linear'
}

hyperparameters={
    'alpha': 1.0,  # L1 regularization parameter
    'objective': 'reg:linear',
    'lambda': 1.0  # L2 regularization parameter (for Lasso in XGBoost)
}
```

6. Hyperparameters Optimization Using GridSearchCV

```
from sklearn.model_selection import GridSearchCV
from sagemaker.sklearn.estimator import SKLearn

# Define hyperparameter grid
param_grid = {
    'alpha': [0.1, 1.0, 10.0],
    'learning_rate': [0.001, 0.01, 0.1],
    'max_depth': [3, 5, 7],
    'n_estimators': [50, 100, 200]
}

# Create a SKLearn estimator
sklearn_estimator = SKLearn(entry_point='script.py', # your script
                            role=role,
                            instance_count=1,
                            instance_type='ml.m5.large',
                            framework_version='0.23-1',
                            sagemaker_session=sess)

# Create GridSearchCV object
grid_search = GridSearchCV(estimator=sklearn_estimator,
                           param_grid=param_grid,
                           cv=3,
                           scoring='neg_mean_squared_error',
                           n_jobs=-1)

# Fit GridSearchCV
grid_search.fit({'train': train_data_s3_path})
```

Conclusion

In this guide, we explored the essential aspects of hyperparameter optimization (HPO) in AWS SageMaker, focusing on managing the bias-variance tradeoff, which is a fundamental concept in machine learning.

We began by defining the hyperparameter ranges that we wanted to optimize, such as the learning rate, number of trees, and maximum depth. These hyperparameters play a crucial role in determining the complexity of the model and, consequently, its ability to generalize to unseen data.

Next, we created an estimator using SageMaker's Estimator class, specifying the necessary configurations such as the Docker image URI, IAM role, instance type, and hyperparameters.

We then utilized SageMaker's HyperparameterTuner to perform the hyperparameter optimization. We selected Bayesian optimization as the optimization strategy, which is an advanced method that builds a probabilistic model of the objective function to select the most promising hyperparameters. The HyperparameterTuner automatically searched for the optimal hyperparameters within the defined ranges, aiming to minimize the validation root mean squared error (RMSE).

After fitting the HyperparameterTuner, we analyzed the results to retrieve the best hyperparameters that balance the bias-variance tradeoff effectively. By understanding how different hyperparameters affect the model's generalization performance, we can make informed decisions to adjust the model's complexity and improve its ability to generalize to unseen data.

In conclusion, AWS SageMaker's Hyperparameter Optimization provides a powerful and efficient way to find the optimal hyperparameters that balance bias and variance, thereby enhancing the model's performance and generalization capabilities. By carefully tuning the hyperparameters and analyzing the results, machine learning practitioners can develop models that deliver superior performance on real-world tasks.