End-to-End Machine Learning with Amazon SageMaker

Building, training, and deploying machine learning models with Amazon SageMaker have become pervasive practices in the realms of artificial intelligence and data science. SageMaker provides an extensive suite of tools and services that not only simplify but also optimize the entire machine-learning workflow. This accessibility and efficiency make it equally valuable for both newcomers and seasoned practitioners.

From the crucial steps of data preprocessing and model training to the intricate process of hyperparameter tuning and the deployment of models to scalable production environments, SageMaker offers a seamless and scalable solution that caters to all your machine learning requirements. Its adaptability is further augmented by seamless integration with various AWS services, solidifying its position as the go-to choice among data scientists and engineers seeking to develop robust and dependable machine-learning solutions.

In this document, we will delve into the creation of a custom machine learning model using scikit-learn (sklearn) on an open-source Kaggle dataset related to advertising. We will also explore the crucial aspects of model training and hyperparameter tuning, ensuring that our model reaches its full potential in terms of accuracy and performance. Last but not least, we will guide you through the deployment process, illustrating how to create an endpoint and leverage this endpoint to make predictions on test data, thereby completing the end-to-end machine-learning pipeline with Amazon SageMaker.

📘 Steps Involved

To implement this solution, complete the following high-level steps:

1. Create an S3 bucket and upload the data
2. Creating Sagemaker Domain and notebook instance
3. Building Model in notebook instance
4. Training the Model and deploying it for prediction
5. Hyperparameter Tuning and updating the prediction

👀 Prerequisite

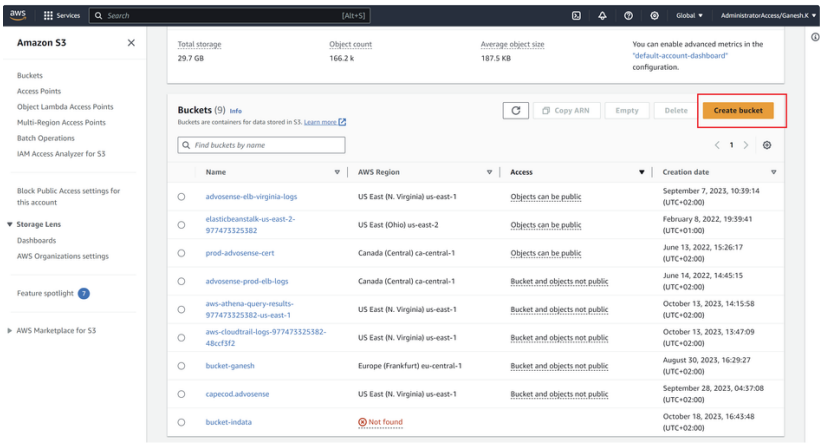
1. Python
2. Sagemaker
3. Machine learning library - Scikitlearn
4. Jupyter Notebooks

💻 Code

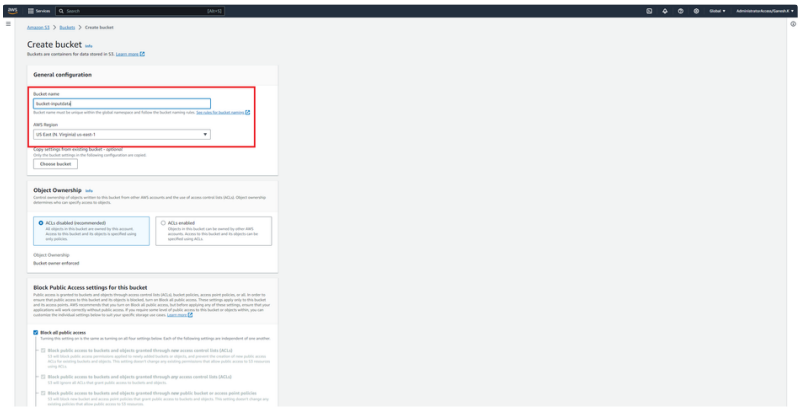
The code files and the current PDF can be found in the Bitbucket repository here

Step 1: Create an S3 bucket and upload the data

1. The project uses the existing bucket (**bucket--ganesh**) instead of creating a new one in case not sure how to create an S3 Bucket here is an example
   a. Open the Amazon S3 console and select the **Buckets** page



   b. Choose **Create bucket** to load the input data

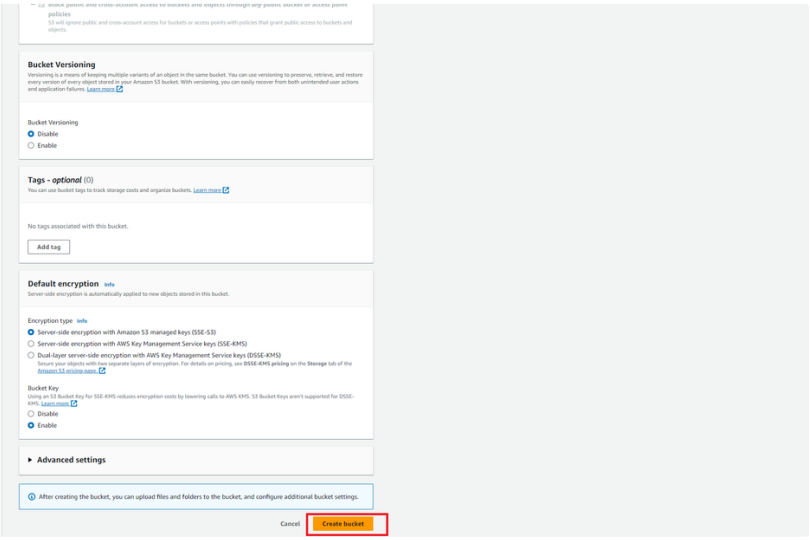


c. Under **General configuration**, do the following:

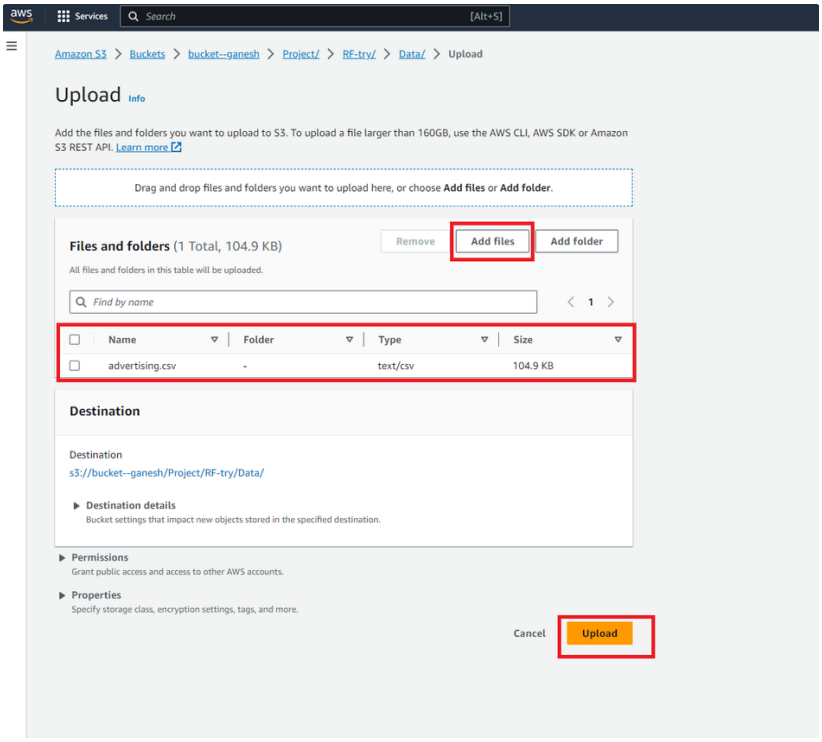
   i. For the **Bucket name**, enter a globally unique name that meets the Amazon S3 Bucket naming rules. Bucket names can contain only lowercase letters, numbers, dots (.), and hyphens (-).

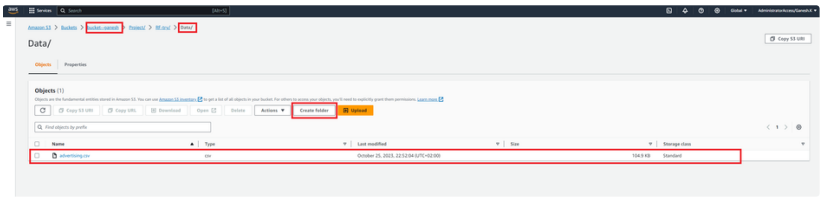
   ii. For **AWS Region**, choose a Region. Later in the tutorial, you must create your Lambda function in the same Region.

d. Leave all other options set to their default values and choose **Create bucket**.



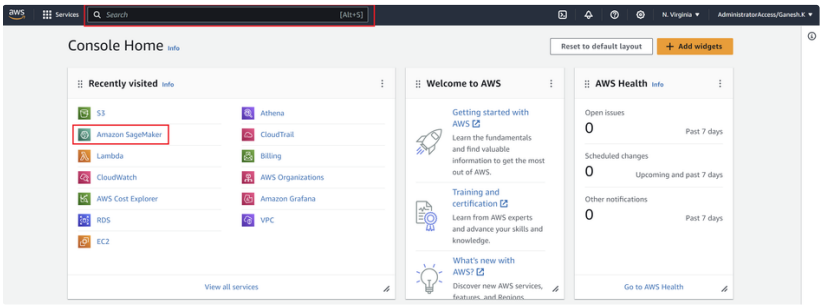
2. Upload the **data**, in the **bucket** after creating a **Data** folder in it. Creating a folder is simple, go to the respective bucket click the **create folder** option give the name **Data**, and create it like creating a folder in the computer. Now navigate inside the folder click **upload** and add the data file to **upload.**



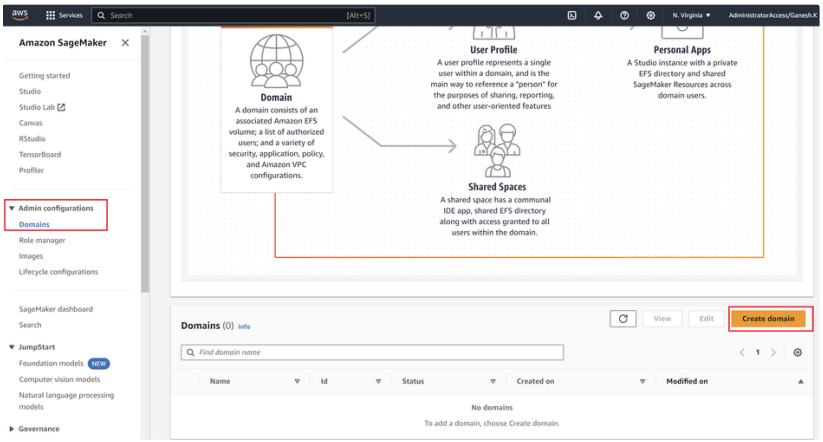


Step 2: Creating Sagemaker Domain and notebook instance

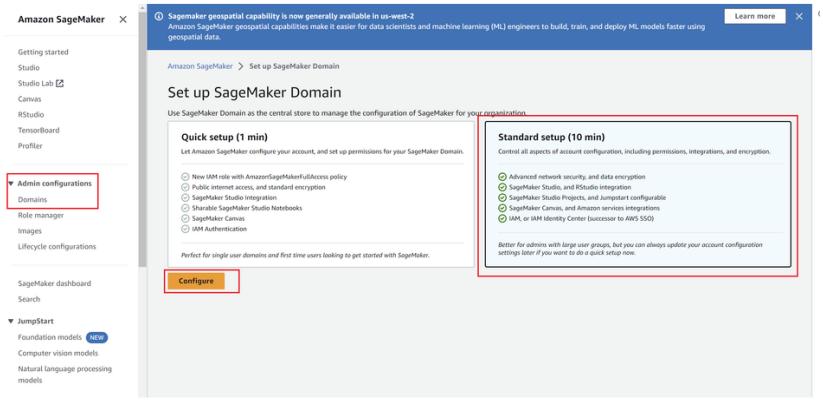
1. Open the Amazon S3 console and select the **Sagemaker** or search for it in the above search bar



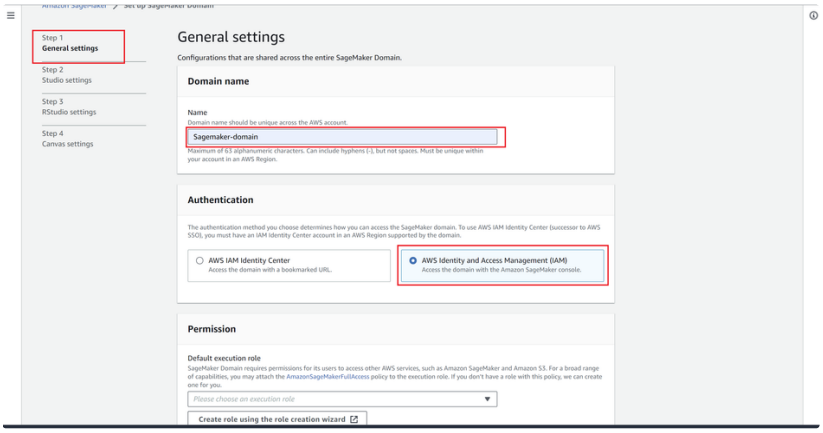
2. In the Sagemaker console, select **Domains** under **Admin Configuration**



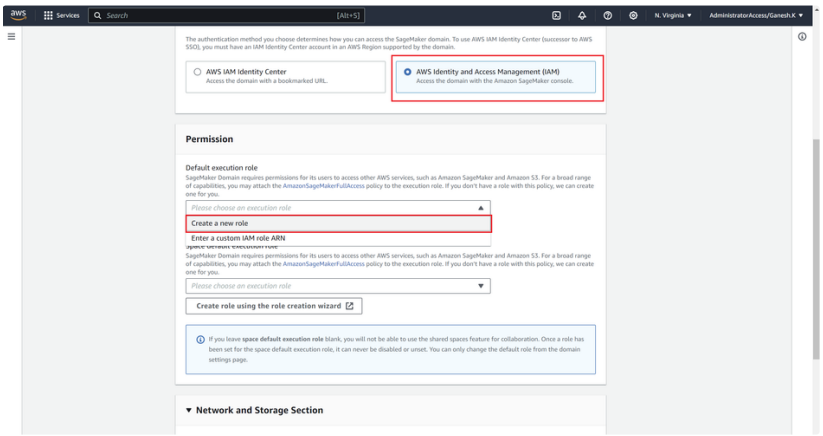
3. In the Set up SageMaker Domain page select **Standard setup** and click **Configure**

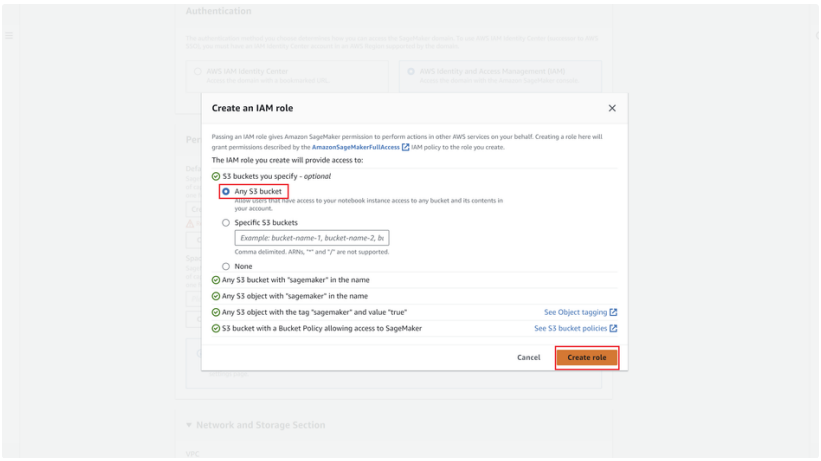


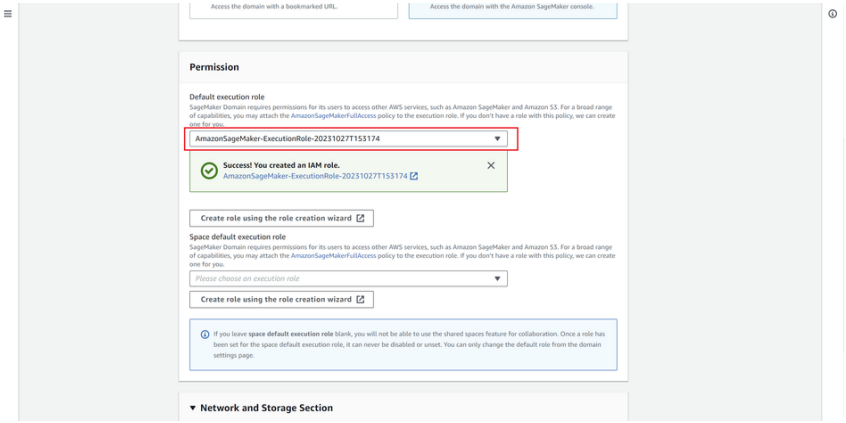
4. In the Configuration page, give a **name for the domain**



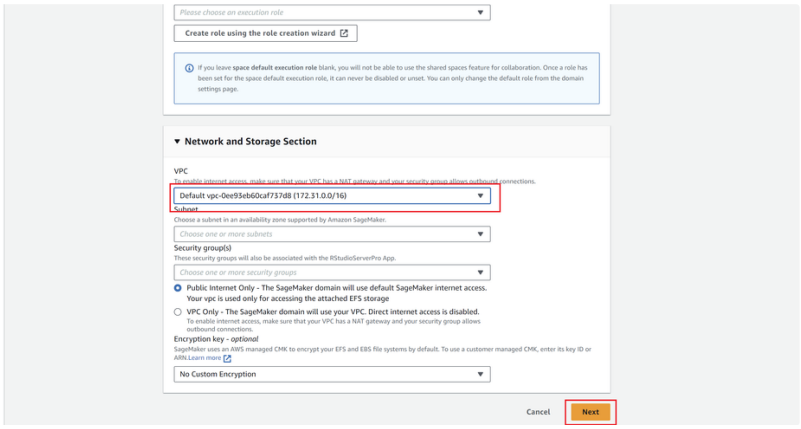
5. Under permission **create a new role**, in the new role page select any s3 bucket, so it can access **any buckets in the same region**



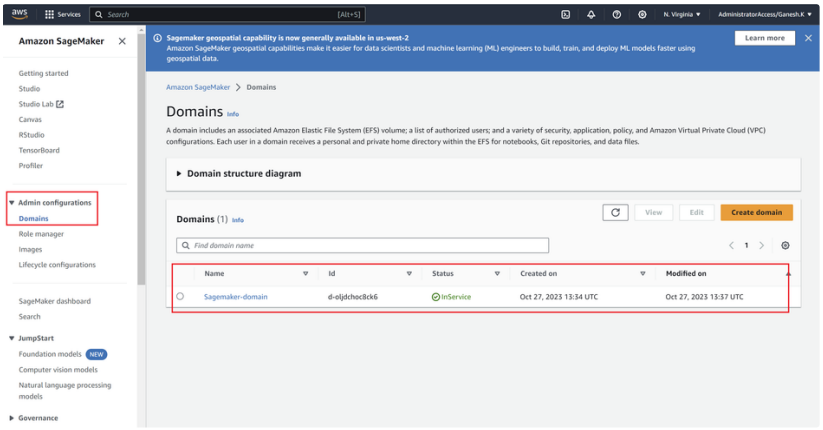




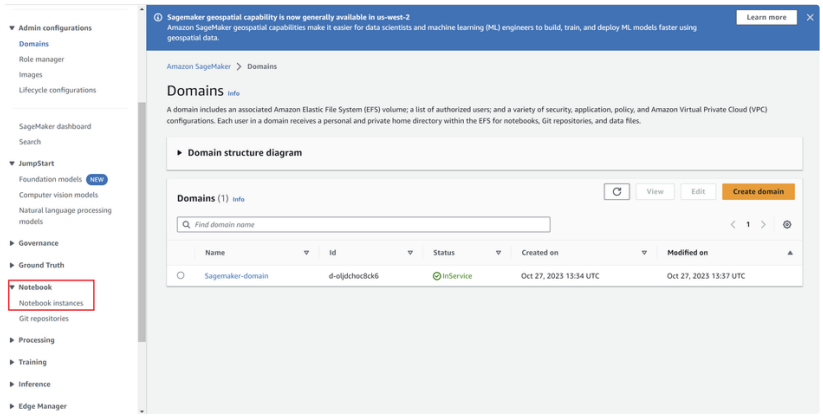
6. Select **default VPC** and click **Next**



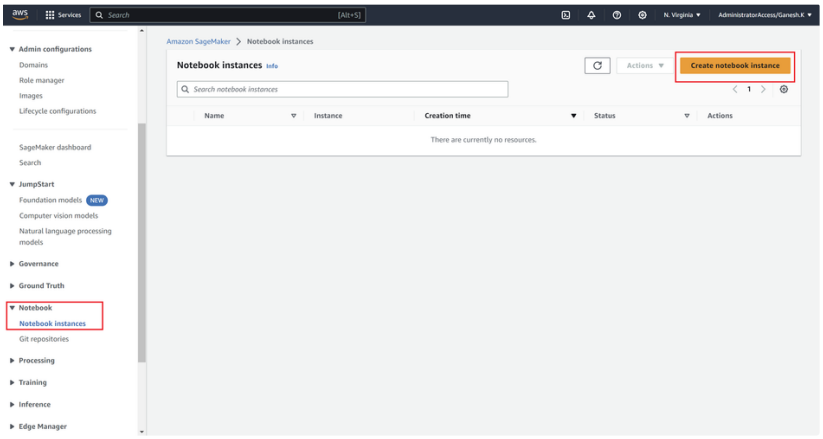
7. Other settings in the subsequent steps can be left as it is, so keep clicking **Next two times followed by Submit on the third time**
8. Creating a Domain might take 5 to 10 minutes



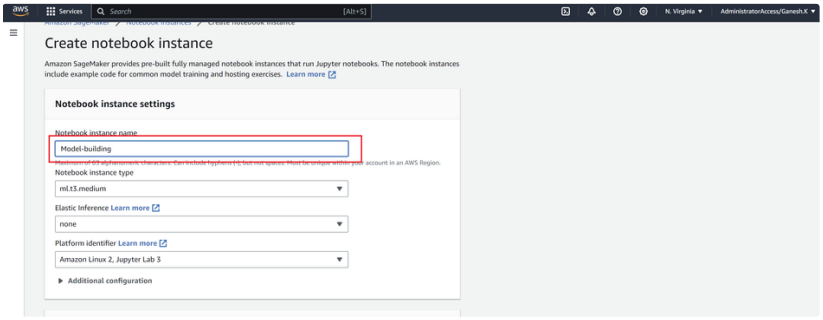
9. Once the Domain, is created we can create **a Notebook Instance.** In the same Sagemaker console, select **Notebook Instances from Notebook**



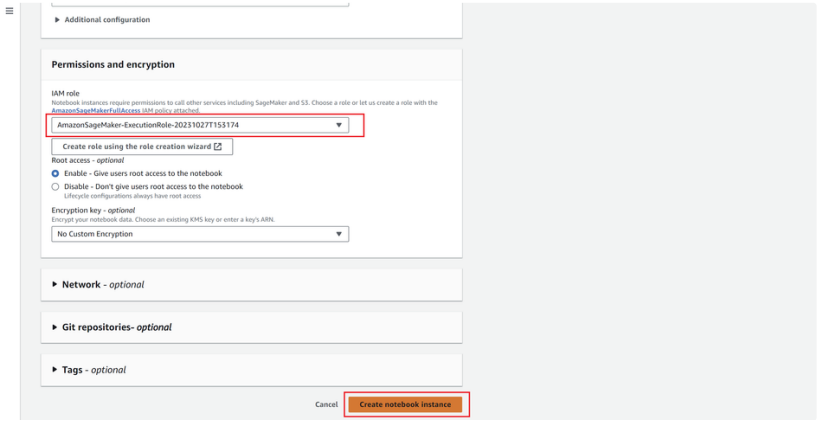
10. In the Notebook Instances page select **Create Notebook Instance**



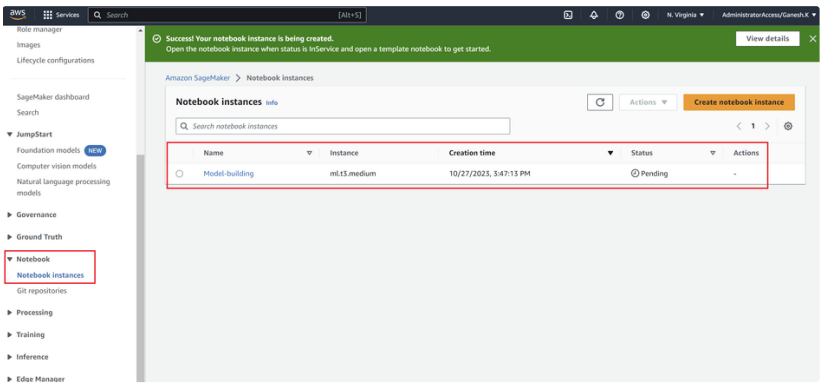
11. On the creation page, give a **name for the notebook.** Adhere to the AWS naming conventions



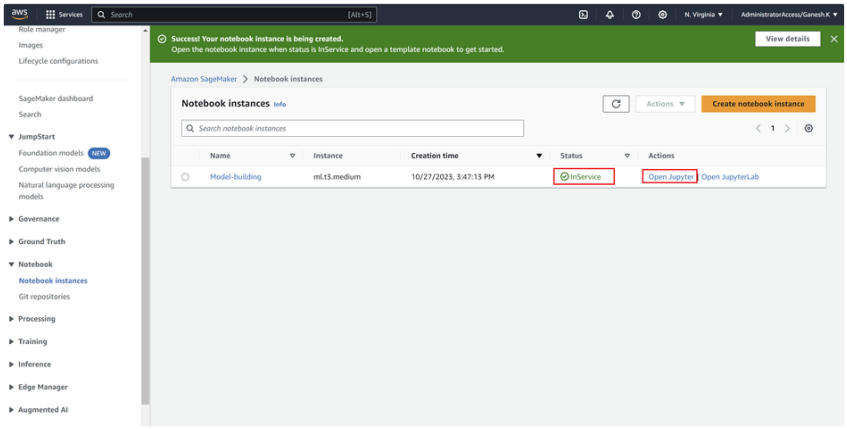
12. Under permission, select the **role** that has been created during **domain creation in Step 5** and click **Create Notebook Instance**. Other settings can be left as it was



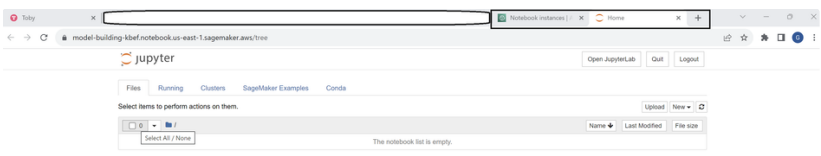
13. The instance takes a few minutes to create



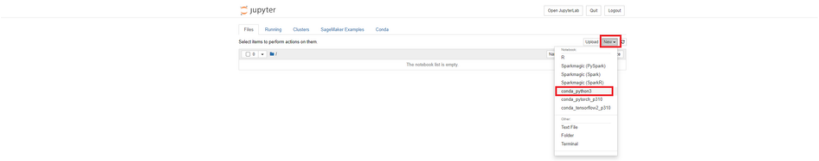
14. After creation, select **Open Jupyter**



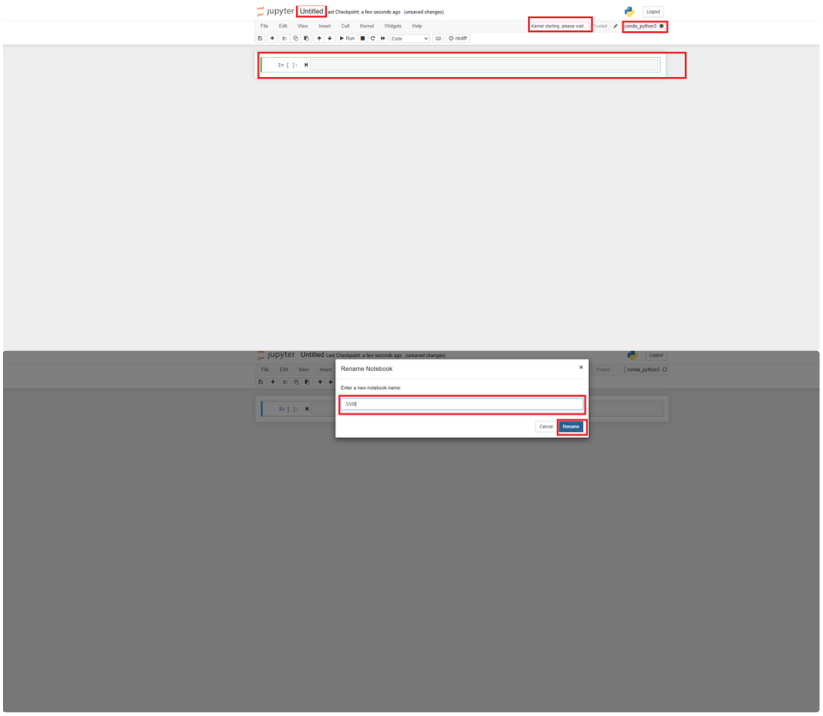
15. In the next window, the **Jupyter Notebook opens, as it is in the local machine**



16. Create a notebook to write the code. Select **New and Conda_python3** as shown below



17. A **new notebook file** opens in the new tab. Double-click the **untitled** and give a **new name** for the file



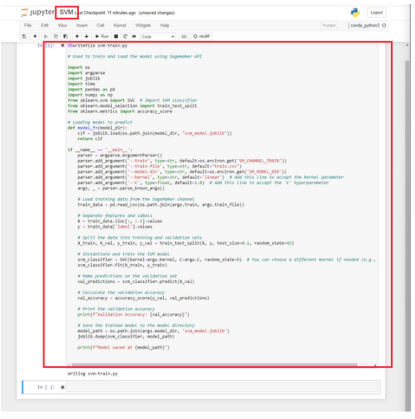
Step 3: Building Model in notebook instance

1. Build a model file for the data. In this project, the model is SVM, therefore building a model accordingly is given below. This script is designed to train an SVM classifier using SageMaker. It takes training data as input, specifies SVM parameters such as the kernel and 'C' hyperparameter, and saves the trained model for later use. This trained model can then be used for making predictions on new data.

```
%%writefile svm-train.py

# Used to train and load the model using Sagemaker API

import os
import argparse
import joblib
import time
import pandas as pd
import numpy as np
from sklearn.svm import SVC  # Import SVM classifier
from sklearn.model_selection import train_test_split
```

```python
from sklearn.metrics import accuracy_score

# Loading model to predict
def model_fn(model_dir):
    clf = joblib.load(os.path.join(model_dir, "svm_model.joblib"))
    return clf

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--train', type=str, default=os.environ.get('SM_CHANNEL_TRAIN'))
    parser.add_argument('--train-file', type=str, default="train.csv")
    parser.add_argument('--model-dir', type=str, default=os.environ.get('SM_MODEL_DIR'))
    parser.add_argument('--kernel', type=str, default='linear')  # Add this line to accept the kernel parameter
    parser.add_argument('--C', type=float, default=1.0)  # Add this line to accept the 'C' hyperparameter
    args, _ = parser.parse_known_args()

    # Load training data from the SageMaker channel
    train_data = pd.read_csv(os.path.join(args.train, args.train_file))

    # Separate features and labels
    X = train_data.iloc[:, :-1].values
    y = train_data['label'].values

    # Split the data into training and validation sets
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.1, random_state=42)

    # Instantiate and train the SVM model
    svm_classifier = SVC(kernel=args.kernel, C=args.C, random_state=0)  # You can choose a different kernel if needed (e.g., 'rbf')
    svm_classifier.fit(X_train, y_train)

    # Make predictions on the validation set
    val_predictions = svm_classifier.predict(X_val)

    # Calculate the validation accuracy
    val_accuracy = accuracy_score(y_val, val_predictions)

    # Print the validation accuracy
    print(f"Validation Accuracy: {val_accuracy}")

    # Save the trained model to the model directory
    model_path = os.path.join(args.model_dir, 'svm_model.joblib')
    joblib.dump(svm_classifier, model_path)

    print(f"Model saved at {model_path}")
```

2. The above script writes a file **svm-train.py** in the notebook directory which will be used later. The **code file is on the left** and the **directory** containing the code file and script file is shown **on the right**



3. **Import the necessary headers** to be used and **run the cell**

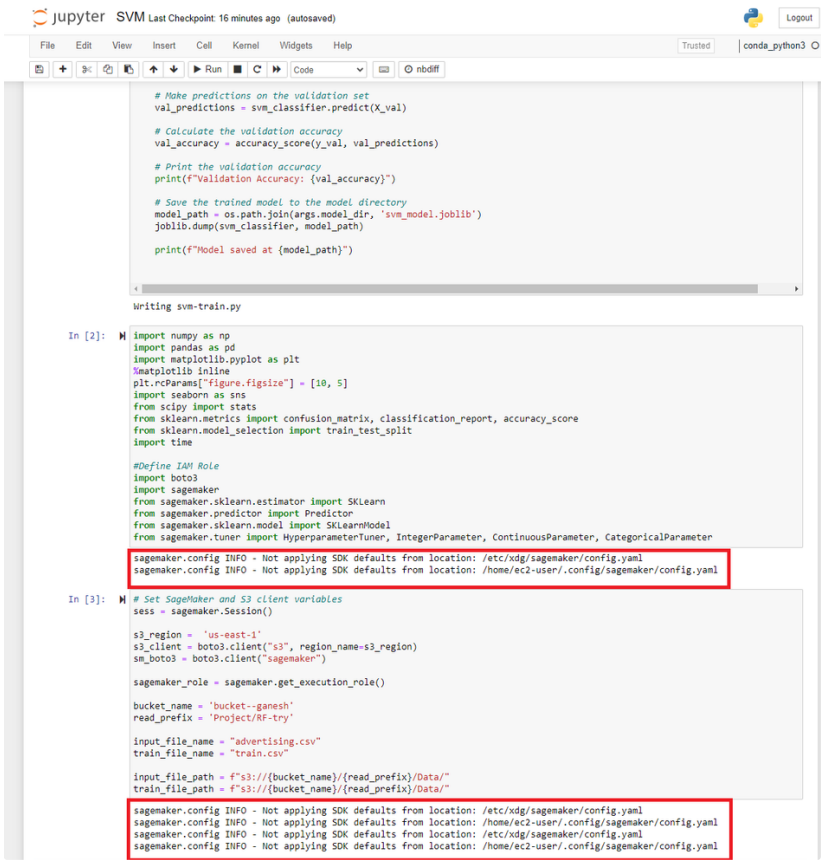
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams["figure.figsize"] = [10, 5]
import seaborn as sns
from scipy import stats
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
from sklearn.model_selection import train_test_split
import time

#Define IAM Role
import boto3
import sagemaker
from sagemaker.sklearn.estimator import SKLearn
from sagemaker.predictor import Predictor
from sagemaker.sklearn.model import SKLearnModel
from sagemaker.tuner import HyperparameterTuner, IntegerParameter, ContinuousParameter, CategoricalParameter
```

4. Define **role, region, path, and session** for the s3 bucket and sagemaker as well as define the file name

```
# Set SageMaker and S3 client variables
sess = sagemaker.Session()

s3_region =  'us-east-1'
s3_client = boto3.client("s3", region_name=s3_region)
sm_boto3 = boto3.client("sagemaker")

sagemaker_role = sagemaker.get_execution_role()

bucket_name = 'bucket--ganesh'
read_prefix = 'Project/RF-try'

input_file_name = "advertising.csv"
train_file_name = "train.csv"

input_file_path = f"s3://{bucket_name}/{read_prefix}/Data/"
train_file_path = f"s3://{bucket_name}/{read_prefix}/Data/"
```

5. Now **load the data** into the sagemaker and run the cell. It should display the data frame table that **confirms the steps are correct till now**.

```
# Load the data.
data = pd.read_csv(input_file_path + input_file_name)
data
```



```
jupyter SVM Last Checkpoint: 16 minutes ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Run Code nbdiff
```

```
# Make predictions on the validation set
val_predictions = svm_classifier.predict(X_val)

# Calculate the validation accuracy
val_accuracy = accuracy_score(y_val, val_predictions)

# Print the validation accuracy
print(f"Validation Accuracy: {val_accuracy}")

# Save the trained model to the model directory
model_path = os.path.join(args.model_dir, 'svm_model.joblib')
joblib.dump(svm_classifier, model_path)

print(f"Model saved at {model_path}")
```

```
Writing svm-train.py
```

In [2]:

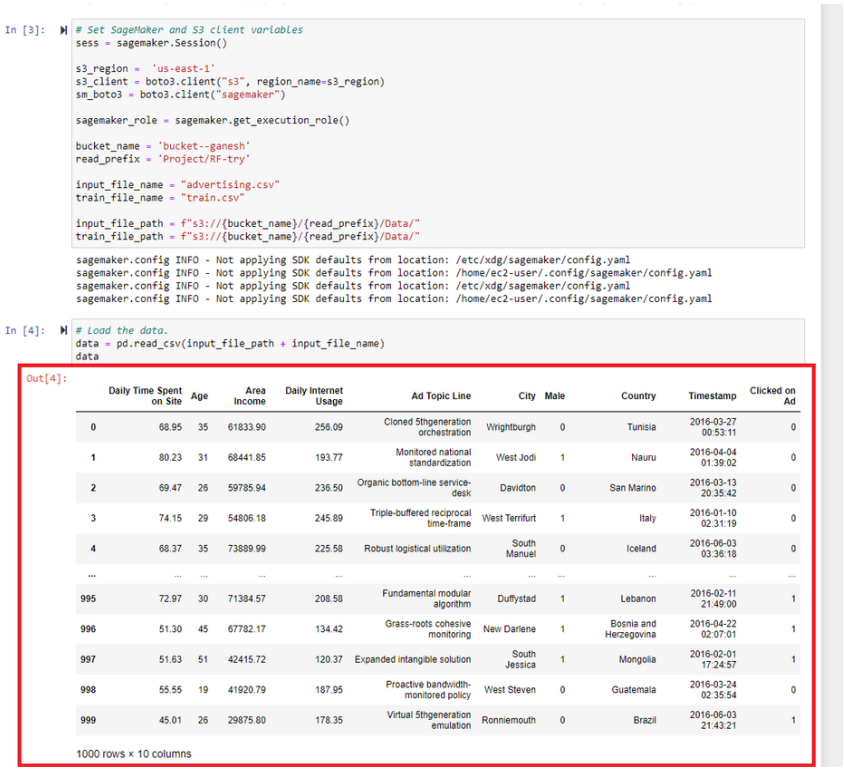
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams["figure.figsize"] = [10, 5]
import seaborn as sns
from scipy import stats
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
from sklearn.model_selection import train_test_split
import time

#Define IAM Role
import boto3
import sagemaker
from sagemaker.sklearn.estimator import SKLearn
from sagemaker.predictor import Predictor
from sagemaker.sklearn.model import SKLearnModel
from sagemaker.tuner import HyperparameterTuner, IntegerParameter, ContinuousParameter, CategoricalParameter
```

```
sagemaker.config INFO - Not applying SDK defaults from location: /etc/xdg/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /home/ec2-user/.config/sagemaker/config.yaml
```

In [3]:

```
# Set SageMaker and S3 client variables
sess = sagemaker.Session()

s3_region =  'us-east-1'
s3_client = boto3.client("s3", region_name=s3_region)
sm_boto3 = boto3.client("sagemaker")

sagemaker_role = sagemaker.get_execution_role()

bucket_name = 'bucket--ganesh'
read_prefix = 'Project/RF-try'

input_file_name = "advertising.csv"
train_file_name = "train.csv"

input_file_path = f"s3://{bucket_name}/{read_prefix}/Data/"
train_file_path = f"s3://{bucket_name}/{read_prefix}/Data/"
```

```
sagemaker.config INFO - Not applying SDK defaults from location: /etc/xdg/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /home/ec2-user/.config/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /etc/xdg/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /home/ec2-user/.config/sagemaker/config.yaml
```



In [3]:

```
# Set SageMaker and S3 client variables
sess = sagemaker.Session()

s3_region =  'us-east-1'
s3_client = boto3.client("s3", region_name=s3_region)
sm_boto3 = boto3.client("sagemaker")

sagemaker_role = sagemaker.get_execution_role()

bucket_name = 'bucket--ganesh'
read_prefix = 'Project/RF-try'

input_file_name = "advertising.csv"
train_file_name = "train.csv"

input_file_path = f"s3://{bucket_name}/{read_prefix}/Data/"
train_file_path = f"s3://{bucket_name}/{read_prefix}/Data/"
```

```
sagemaker.config INFO - Not applying SDK defaults from location: /etc/xdg/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /home/ec2-user/.config/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /etc/xdg/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /home/ec2-user/.config/sagemaker/config.yaml
```

In [4]:

```
# Load the data.
data = pd.read_csv(input_file_path + input_file_name)
data
```

Out[4]:

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Ad Topic Line	City	Male	Country	Timestamp	Clicked on Ad
0	68.95	35	61833.90	256.09	Cloned 5thgeneration orchestration	Wrightburgh	0	Tunisia	2016-03-27 00:53:11	0
1	80.23	31	68441.85	193.77	Monitored national standardization	West Jodi	1	Nauru	2016-04-04 01:39:02	0
2	69.47	26	59785.94	236.50	Organic bottom-line service-desk	Davidton	0	San Marino	2016-03-13 20:35:42	0
3	74.15	29	54806.18	245.89	Triple-buffered reciprocal time-frame	West Terrifurt	1	Italy	2016-01-10 02:31:19	0
4	68.37	35	73889.99	225.58	Robust logistical utilization	South Manuel	0	Iceland	2016-06-03 03:36:18	0
...	...	...	...	...	...	...	...	...	...	...
995	72.97	30	71384.57	208.58	Fundamental modular algorithm	Duffystad	1	Lebanon	2016-02-11 21:49:00	1
996	51.30	45	67782.17	134.42	Grass-roots cohesive monitoring	New Darlene	1	Bosnia and Herzegovina	2016-04-22 02:07:01	1
997	51.63	51	42415.72	120.37	Expanded intangible solution	South Jessica	1	Mongolia	2016-02-01 17:24:57	1
998	55.55	19	41920.79	187.95	Proactive bandwidth-monitored policy	West Steven	0	Guatemala	2016-03-24 02:35:54	0
999	45.01	26	29875.80	178.35	Virtual 5thgeneration emulation	Ronniemouth	0	Brazil	2016-06-03 21:43:21	1

1000 rows × 10 columns

6. Now **preprocess the data** and **split the data into test and train**. Save the train file to load for training in the S3 bucket data folder where the data file is present.

```
# Timestamp conversion to seconds
data['Timestamp'] = (pd.to_datetime(data['Timestamp']) - pd.to_datetime(data['Timestamp'][0])).dt.total_seconds()
```

```
# Drop non-integer columns
data = data.drop(columns=['Ad Topic Line', 'Country', 'City'])

# Extract features and target variable
X = data.iloc[:, 1:-1].values.astype('float64')
y = data['Clicked on Ad'].values

# Pre-processing - normalization of data
X = (X - np.mean(X, axis=0)) / np.std(X, axis=0)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)

# Write the training data to a file
trainX = pd.DataFrame(X_train)
trainX['label'] = y_train

trainX.to_csv(train_file_path + train_file_name, index=False, header=True)
```

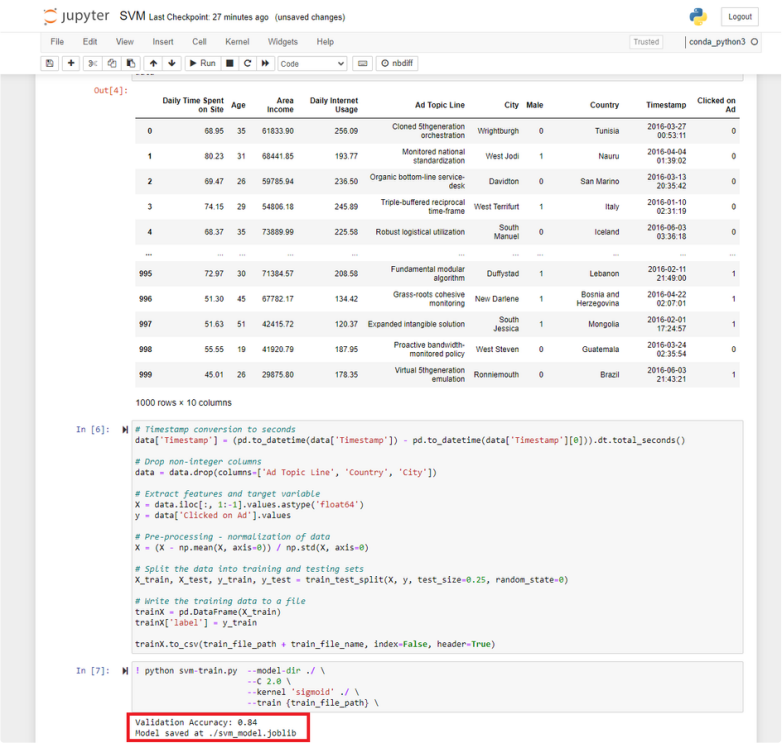
7. Now both **data and model are ready to train**

Step 4: Training the Model and deploying it for prediction

1. Before training **verify the code file/model** working with the below code

```
! python svm-train.py  --model-dir ./ \
                       --C 2.0 \
                       --kernel 'sigmoid' ./ \
                       --train {train_file_path} \

```

2. Need to see an **output model file** in the notebook directory and **validation accuracy** printed below the cell



Out[4]:

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Ad Topic Line	City	Male	Country	Timestamp	Clicked on Ad
0	68.95	35	61833.90	256.09	Cloned 5thgeneration orchestration	Wrightburgh	0	Tunisia	2016-03-27 00:53:11	0
1	80.23	31	68441.85	193.77	Monitored national standardization	West Jodi	1	Nauru	2016-04-04 01:39:02	0
2	69.47	26	59785.94	236.50	Organic bottom-line service-desk	Davidton	0	San Marino	2016-03-13 20:35:42	0
3	74.15	29	54806.18	245.89	Triple-buffered reciprocal time-frame	West Terrifurt	1	Italy	2016-01-10 02:31:19	0
4	68.37	35	73889.99	225.58	Robust logistical utilization	South Manuel	0	Iceland	2016-06-03 03:36:18	0
...	...	...	...	...	...	...	...	...	...	...
995	72.97	30	71384.57	208.58	Fundamental modular algorithm	Duffystad	1	Lebanon	2016-02-11 21:49:00	1
996	51.30	45	67782.17	134.42	Grass-roots cohesive monitoring	New Darlene	1	Bosnia and Herzegovina	2016-04-22 02:07:01	1
997	51.63	51	42415.72	120.37	Expanded intangible solution	South Jessica	1	Mongolia	2016-02-01 17:24:57	1
998	55.55	19	41920.79	187.95	Proactive bandwidth-monitored policy	West Steven	0	Guatemala	2016-03-24 02:35:54	0
999	45.01	26	29875.80	178.35	Virtual 5thgeneration emulation	Ronniemouth	0	Brazil	2016-06-03 21:43:21	1

1000 rows × 10 columns

In [6]:
```
# Timestamp conversion to seconds
data['Timestamp'] = (pd.to_datetime(data['Timestamp']) - pd.to_datetime(data['Timestamp'][0])).dt.total_seconds()

# Drop non-integer columns
data = data.drop(columns=['Ad Topic Line', 'Country', 'City'])

# Extract features and target variable
X = data.iloc[:, 1:-1].values.astype('float64')
y = data['Clicked on Ad'].values

# Pre-processing - normalization of data
X = (X - np.mean(X, axis=0)) / np.std(X, axis=0)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)

# Write the training data to a file
trainX = pd.DataFrame(X_train)
trainX['label'] = y_train

trainX.to_csv(train_file_path + train_file_name, index=False, header=True)
```

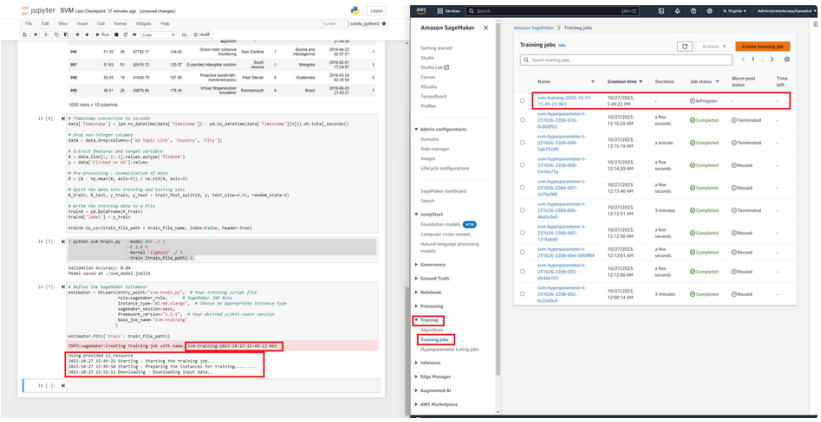
In [7]:
```
! python svm-train.py  --model-dir ./ \
                       --C 2.0 \
                       --kernel 'sigmoid' ./ \
                       --train {train_file_path} \
```

```
Validation Accuracy: 0.84
Model saved at ./svm_model.joblib
```

3. Now make the **Sagemaker train the model.** This takes some time, but the job can be seen in the sagemaker console.

```
# Define the SageMaker Estimator
estimator = SKLearn(entry_point="svm-train.py",  # Your training script file
                    role=sagemaker_role,      # SageMaker IAM Role
                    instance_type="ml.m4.xlarge",  # Choose an appropriate instance type
                    sagemaker_session=sess,
                    framework_version="1.2-1",  # Your desired scikit-learn version
                    base_job_name="svm-training"
                   )

estimator.fit({'train': train_file_path})
```

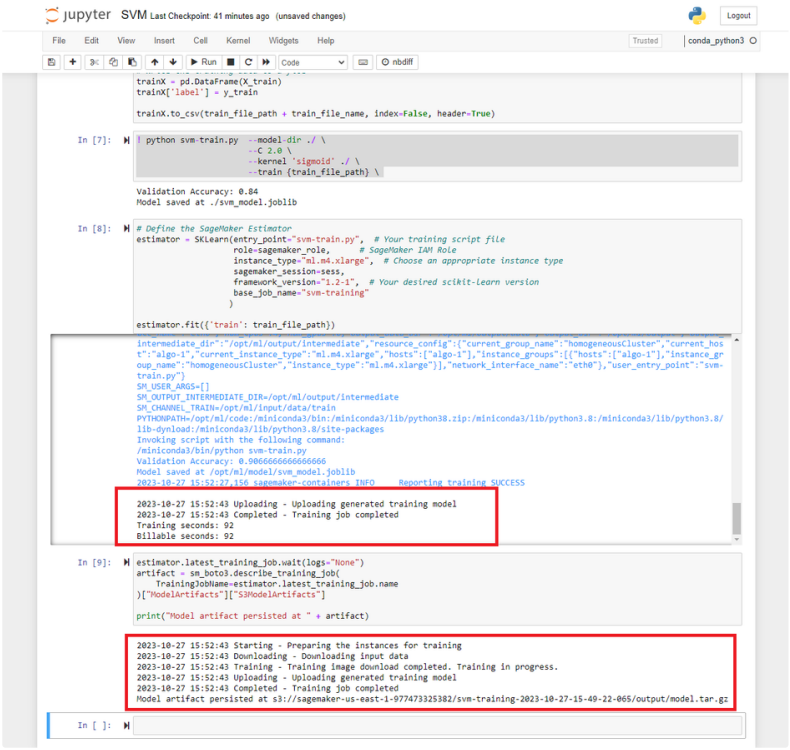
4. In the Sagemaker console, select **training Jobs from the Training.** The job created in the notebook is visible here with the status



5. Once it's **successful**, the **status** in the sagemaker is changed to **completed** which means the model is trained and ready to deploy
6. Now download the **artifacts, and hyperparameter** from the model and save it for later use

```
estimator.latest_training_job.wait(logs="None")
artifact = sm_boto3.describe_training_job(
```

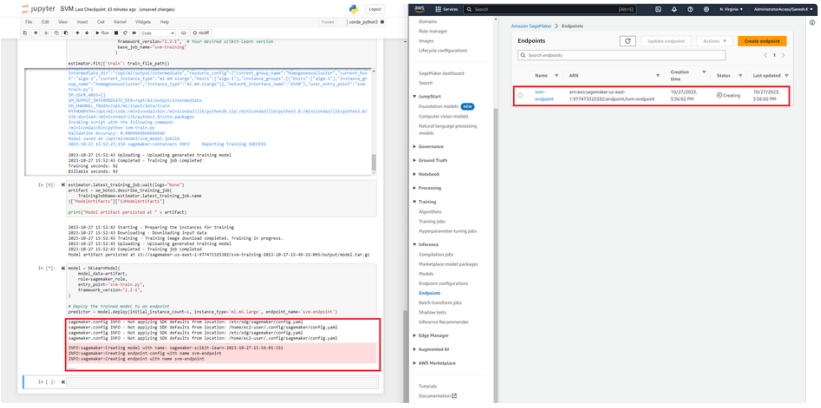
```
    TrainingJobName=estimator.latest_training_job.name
)["ModelArtifacts"]["S3ModelArtifacts"]

print("Model artifact persisted at " + artifact)
```



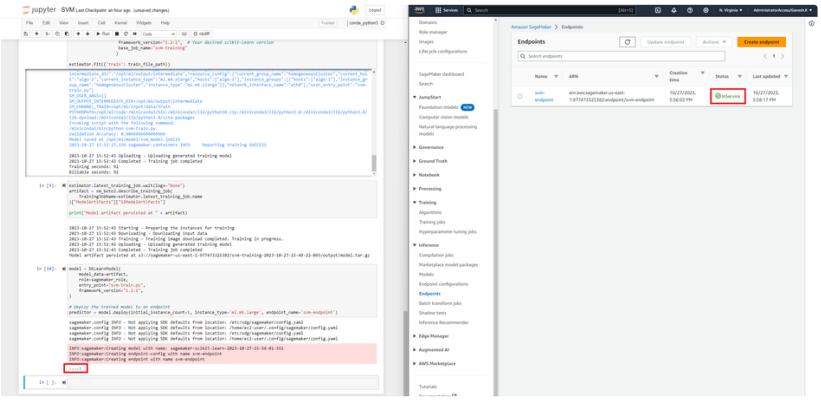
7. Now create the **endpoint for inference**

```
model = SKLearnModel(
    model_data=artifact,
    role=sagemaker_role,
    entry_point="svm-train.py",
    framework_version="1.2-1",
)

# Deploy the trained model to an endpoint
predictor = model.deploy(initial_instance_count=1, instance_type='ml.m5.large', endpoint_name='svm-endpoint')

```

8. Similar to the training job, even the endpoint can be seen in the sagemaker console in **Enpoints under Inference**



9. Once the creation is done, the **status** changes to **Inservice**

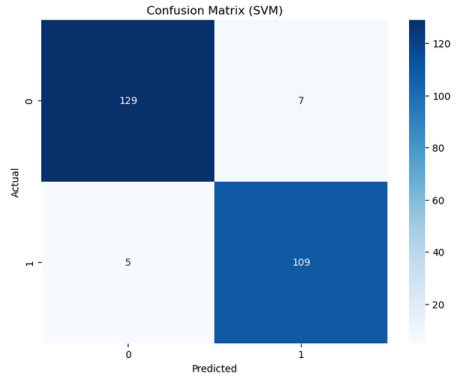


10. Now the **endpoint** can be used to **predict** the label for **test data**

```
predictions = predictor.predict(X_test)
print(classification_report(y_test, predictions))

cnf_matrix_rf = confusion_matrix(y_test, predictions)

# Create a heatmap for the Random Forest confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cnf_matrix_rf, annot=True, cmap='Blues', fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix (SVM)')
plt.show()

# Calculate and print accuracy
acc_rf = accuracy_score(y_test, predictions)
print(f"Accuracy using Random Forest: {acc_rf * 100:.2f}%")

```

```
predictions = predictor.predict(X_test)
print(classification_report(y_test, predictions))

cnf_matrix_rf = confusion_matrix(y_test, predictions)

# Create a heatmap for the Random Forest confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cnf_matrix_rf, annot=True, cmap='Blues', fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix (SVM)')
plt.show()

# Calculate and print accuracy
acc_rf = accuracy_score(y_test, predictions)
print(f"Accuracy using Random Forest: {acc_rf * 100:.2f}%")
```

```
              precision    recall  f1-score   support

           0       0.96      0.95      0.96       136
           1       0.94      0.96      0.95       114

    accuracy                           0.95       250
   macro avg       0.95      0.95      0.95       250
weighted avg       0.95      0.95      0.95       250
```



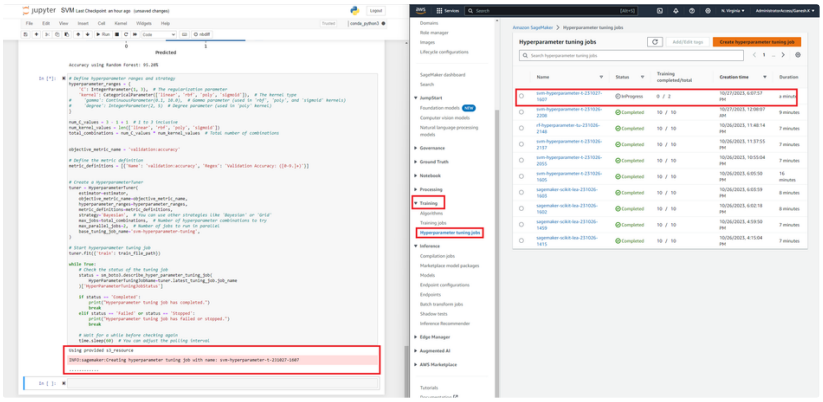

```
Accuracy using Random Forest: 95.20%
```

Step 5: Hyperparameter Tuning and updating the prediction

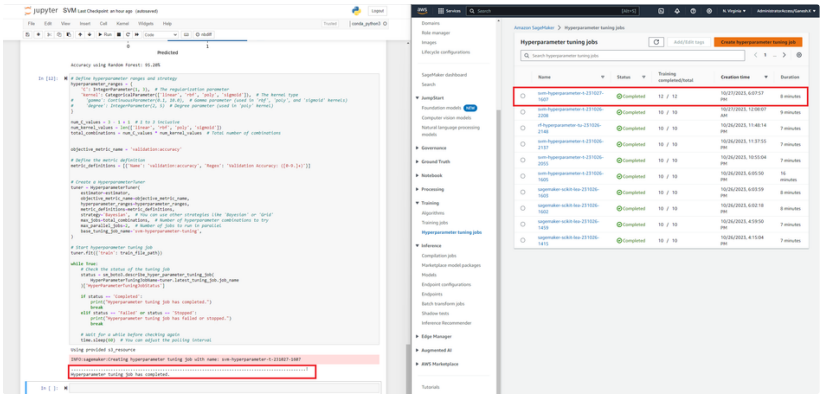
1. Now **hyperparameter tuning** to get the best parameter for the current model. In this project for demo, only **two parameters are used that is ‘C' and 'kernel’ for SVM**.

```
# Define hyperparameter ranges and strategy
hyperparameter_ranges = {
    'C': IntegerParameter(1, 3),  # The regularization parameter
    'kernel': CategoricalParameter(['linear', 'rbf', 'poly', 'sigmoid']),  # The kernel type
#     'gamma': ContinuousParameter(0.1, 10.0),  # Gamma parameter (used in 'rbf', 'poly', and 'sigmoid' kernels)
#     'degree': IntegerParameter(2, 5)  # Degree parameter (used in 'poly' kernel)
}

num_C_values = 3 - 1 + 1  # 1 to 3 inclusive
num_kernel_values = len(['linear', 'rbf', 'poly', 'sigmoid'])
total_combinations = num_C_values * num_kernel_values  # Total number of combinations


objective_metric_name = 'validation:accuracy'

# Define the metric definition
metric_definitions = [{'Name': 'validation:accuracy', 'Regex': 'Validation Accuracy: ([0-9.]+)'}]


# Create a HyperparameterTuner
tuner = HyperparameterTuner(
    estimator=estimator,
    objective_metric_name=objective_metric_name,
    hyperparameter_ranges=hyperparameter_ranges,
    metric_definitions=metric_definitions,
    strategy='Bayesian',  # You can use other strategies like 'Bayesian' or 'Grid'
    max_jobs=total_combinations,  # Number of hyperparameter combinations to try
    max_parallel_jobs=2,  # Number of jobs to run in parallel
    base_tuning_job_name='svm-hyperparameter-tuning',
)

# Start hyperparameter tuning job
tuner.fit({'train': train_file_path})

while True:
    # Check the status of the tuning job
    status = sm_boto3.describe_hyper_parameter_tuning_job(
        HyperParameterTuningJobName=tuner.latest_tuning_job.job_name
    )['HyperParameterTuningJobStatus']

    if status == 'Completed':
        print("Hyperparameter tuning job has completed.")
        break
    elif status == 'Failed' or status == 'Stopped':
        print("Hyperparameter tuning job has failed or stopped.")
        break

    # Wait for a while before checking again
    time.sleep(60)  # You can adjust the polling interval
```

2. Even the **hyperparameter** job can be visible the same as the **training job**, actually, it's down to the training job option itself. Since a **parallel** job is two, it runs as a pair of jobs from the total jobs.



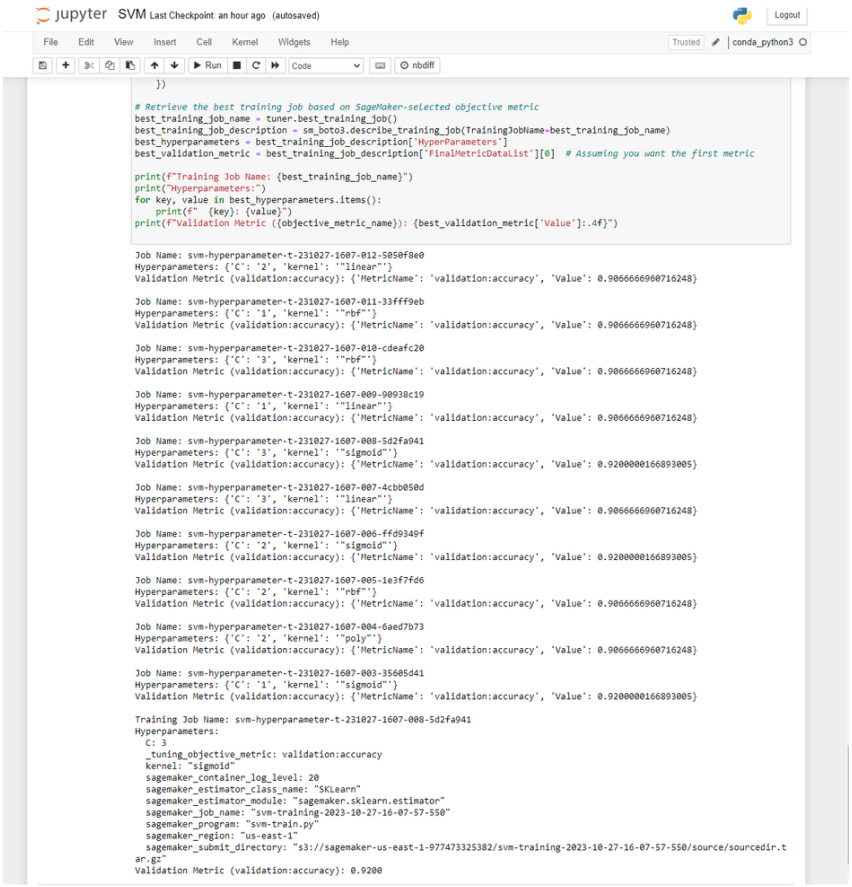
3. After **completion**



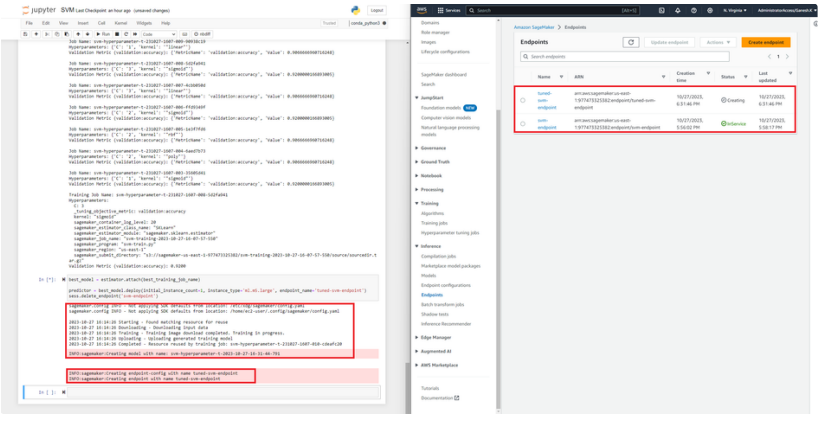
4. Also possible to print what parameters have been used and what accuracy each parameter setting has generated. Also the **Sagemaker's best pick**

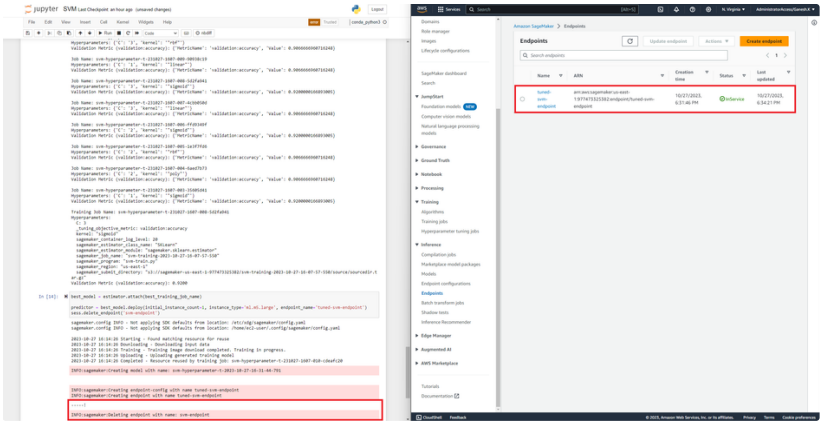
```python
# Retrieve the details of the completed hyperparameter tuning job
tuning_job_name = tuner.latest_tuning_job.job_name
training_job_summaries = sm_boto3.list_training_jobs_for_hyper_parameter_tuning_job(
    HyperParameterTuningJobName=tuning_job_name
)['TrainingJobSummaries']

# Get the best training job from SageMaker
tuning_job_description = sm_boto3.describe_hyper_parameter_tuning_job(
    HyperParameterTuningJobName=tuning_job_name
)

# Print hyperparameters and validation accuracy for each training job
results = []
for job_summary in training_job_summaries:
    job_name = job_summary['TrainingJobName']
    hyperparameters = job_summary['TunedHyperParameters']
    validation_metric = job_summary['FinalHyperParameterTuningJobObjectiveMetric']

    print(f"Job Name: {job_name}")
    print(f"Hyperparameters: {hyperparameters}")
    print(f"Validation Metric ({objective_metric_name}): {validation_metric}")
    print("")

    results.append({
        'TrainingJobName': job_name,
        'Hyperparameters': hyperparameters,
        'ValidationMetric': validation_metric
    })

# Retrieve the best training job based on SageMaker-selected objective metric
best_training_job_name = tuner.best_training_job()
best_training_job_description = sm_boto3.describe_training_job(TrainingJobName=best_training_job_name)
best_hyperparameters = best_training_job_description['HyperParameters']
best_validation_metric = best_training_job_description['FinalMetricDataList'][0]  # Assuming you want the first metric

print(f"Training Job Name: {best_training_job_name}")
print("Hyperparameters:")
```

```python
for key, value in best_hyperparameters.items():
    print(f"  {key}: {value}")
print(f"Validation Metric ({objective_metric_name}): {best_validation_metric['Value']:.4f}")

# Convert the results to a DataFrame for further analysis if needed
results_df = pd.DataFrame(results)
```



5. Now fit the best model create a **new endpoint and delete the old endpoint**



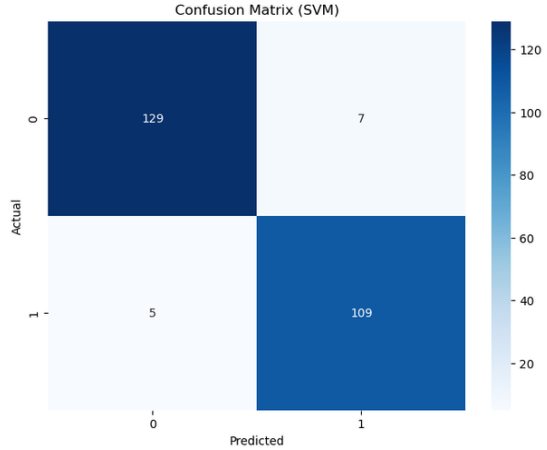


6. Now run the endpoint for **inference on test data**

```
predictions = predictor.predict(X_test)
print(classification_report(y_test, predictions))

cnf_matrix_rf = confusion_matrix(y_test, predictions)

# Create a heatmap for the Random Forest confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cnf_matrix_rf, annot=True, cmap='Blues', fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix (Random Forest)')
plt.show()

# Calculate and print accuracy
acc_rf = accuracy_score(y_test, predictions)
print(f"Accuracy using Random Forest: {acc_rf * 100:.2f}%")
```

```
predictions = predictor.predict(X_test)
print(classification_report(y_test, predictions))

cnf_matrix_rf = confusion_matrix(y_test, predictions)

# Create a heatmap for the Random Forest confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cnf_matrix_rf, annot=True, cmap='Blues', fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix (SVM)')
plt.show()

# Calculate and print accuracy
acc_rf = accuracy_score(y_test, predictions)
print(f"Accuracy using Random Forest: {acc_rf * 100:.2f}%")
```

```
              precision    recall  f1-score   support

           0       0.96      0.95      0.96       136
           1       0.94      0.96      0.95       114

    accuracy                           0.95       250
   macro avg       0.95      0.95      0.95       250
weighted avg       0.95      0.95      0.95       250
```




```
Accuracy using Random Forest: 95.20%
```

Although immediate improvements may not be readily apparent, as we scale up our data and leverage more intricate models, this particular step takes on greater significance. Consequently, we've undertaken the task of building, training, and optimizing a Support Vector Machine (SVM) model utilizing the advertising dataset. Moreover, we've established an endpoint that serves as the final inference point for our test data. This process stands as a pivotal phase in our data analysis and modeling endeavors, executed within the AWS infrastructure through Amazon SageMaker, a platform that streamlines machine learning development and deployment.