Configuration Manual

MSc Research Project
Programme Name

Tanmaya Kumar Dixit
Student ID: x23116668

School of Computing
National College of Ireland

Supervisor: Shaguna Gupta

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Tanmaya Kumar Dixit

Student ID: x23116668

Programme: Msc in Cloud Computing **Year:** 2023-2024

Module: Msc Research Project

Supervisor: Shaguna Gupta

Submission Due Date: 14/08/2024

Project Title: Securing Financial Sector in the Cloud: A Multi-Cloud Approach to Fraud Detection Using Secure Multi-Party Computation

Word Count: 4000 **Page Count** 25

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Tanmaya Kumar Dixit

Date: 14/08/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Tanmaya Kumar Dixit
X23116668

1 Datasets

I. Dataset download links

II. Anti-money laundering Dataset - https://www.kaggle.com/datasets/berkanoztas/synthetic-transaction-monitoring-dataset-aml/code

III. Credit card fraud Dataset- https://www.kaggle.com/datasets/iabhishekofficial/creditcard-fraud-detection/data

2 Dataset pre-processing

I. Open google collab and then click on Files option and then click on upload to session storage option and upload your credit card fraud dataset.

Following steps were used to combine two dataframes and create is_fradulent label column using K means clustering algorithm.

```python
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Load datasets
card_info = pd.read_csv('/content/cc_info.csv')
transaction_info = pd.read_csv('/content/transactions.csv')

# Check for missing values
print(card_info.isnull().sum())
print(transaction_info.isnull().sum())

# Merge the datasets on 'credit_card' column
df = transaction_info.merge(card_info, on='credit_card')
print(df.head())

# Preprocess data by scaling relevant features
scaler = StandardScaler()
features = ['transaction_dollar_amount', 'Long', 'Lat', 'credit_card_limit']
df[features] = scaler.fit_transform(df[features])

# Apply KMeans clustering
kmeans = KMeans(n_clusters=2, random_state=42)
df['cluster_label'] = kmeans.fit_predict(df[features])

# Label transactions as fraudulent based on clustering
cluster_fraud_label = df.groupby('cluster_label')['transaction_dollar_amount'].mean().idxmax()
df['is_fraudulent'] = df['cluster_label'].apply(lambda x: 1 if x == cluster_fraud_label else 0)

# Output the head of the dataframe to see the result
print(df.head())

# Save the combined and labeled dataset to a CSV file
df.to_csv('preprocessed_dataset.csv', index=False)
```

Dataset description

```python
df.head(10)
```

	credit_card	date	transaction_dollar_amount	Long	Lat	city	state	zipcode	credit_card_limit	cluster_label	is_fraudulent
0	1003715054175576	2015-09-11 00:32:40	-0.338757	-0.195624	-0.124321	Houston	PA	15342	0.565333	0	0
1	1003715054175576	2015-10-24 22:23:08	0.137514	-0.196623	-0.140504	Houston	PA	15342	0.565333	0	0
2	1003715054175576	2015-10-26 18:19:36	-0.300492	-0.197457	-0.115857	Houston	PA	15342	0.565333	0	0
3	1003715054175576	2015-10-22 19:41:10	0.402484	-0.195625	-0.119957	Houston	PA	15342	0.565333	0	0
4	1003715054175576	2015-10-26 20:08:22	-0.113818	-0.198832	-0.142989	Houston	PA	15342	0.565333	0	0
5	1003715054175576	2015-10-17 21:28:57	0.285522	-0.199073	-0.125523	Houston	PA	15342	0.565333	0	0
6	1003715054175576	2015-08-29 18:34:04	0.293945	-0.198806	-0.128298	Houston	PA	15342	0.565333	0	0
7	1003715054175576	2015-08-14 21:34:39	0.096201	-0.199098	-0.127701	Houston	PA	15342	0.565333	0	0
8	1003715054175576	2015-09-17 19:20:37	0.073739	3.645499	-1.148556	Houston	PA	15342	0.565333	0	0
9	1003715054175576	2015-09-11 18:59:04	-0.225084	-0.201256	-0.124227	Houston	PA	15342	0.565333	0	0

II Second dataset Anti-money laudenring dataset description

```python
import pandas as pd

# Load a CSV file
df = pd.read_csv('/content/SAML-D.csv')

# Display the first few rows of the DataFrame
df.head()
```

	Time	Date	Sender_account	Receiver_account	Amount	Payment_currency	Received_currency	Sender_bank_location	Receiver_bank_location	Payment_type	Is_laundering	Laundering_type
0	10:35:19	2022-10-07	8724731955	2769355426	1459.15	UK pounds	UK pounds	UK	UK	Cash Deposit	0.0	Normal_Cash_Deposits
1	10:35:20	2022-10-07	1491989064	8401255335	6019.64	UK pounds	Dirham	UK	UAE	Cross-border	0.0	Normal_Fan_Out
2	10:35:20	2022-10-07	287305149	4404767002	14328.44	UK pounds	UK pounds	UK	UK	Cheque	0.0	Normal_Small_Fan_Out
3	10:35:21	2022-10-07	5376652437	9600420220	11895.00	UK pounds	UK pounds	UK	UK	ACH	0.0	Normal_Fan_In
4	10:35:21	2022-10-07	9614186178	3803336972	115.25	UK pounds	UK pounds	UK	UK	Cash Deposit	0.0	Normal_Cash_Deposits

3 Setting up Environment for Encryption

Now before proceeding with encryption several tools and software were utilised to properly set up the required environment. We have locally done the encryption of each dataset. Let see step wise step process of setting up the development environment.

1. Download and install Visual Studio Code with C++ support. and open Visual Studio code.
2. Install GIT on the system. Below is the installation guideline to GIT https://github.com/git-guides/install-git
3. Download and install python https://www.python.org/downloads/
4. Download the latest version of Python 3.9
5. Run the installer and during installation, make sure to check the box that says, "Add Python to PATH."
6. Create a virtual environment in python and install SEAL into it
7. Building the SEAL Library:
    - https://cmake.org/download/ ,Visit the CMake official website and download the installer for windows. Run the installer and follow the instructions and add the system PATH during installation.

Installing SEAL library and its python binding:

- Clone the SEAL-Python Repository- git clone
  https://github.com/Huelse/SEAL-Python.git
  cd SEAL-Python
- Initialize and Update Submodules- git submodule update --init –recursive
- Build the SEAL Library- cd SEAL
  ```
  cd SEAL
  cmake -S . -B build -G "Ninja" -DSEAL_USE_MSGSL=OFF
  -DSEAL_USE_ZLIB=OFF
  cmake --build build
  cd ..
  ```
- Install Python Requirements-
  pip install numpy pybind11
- Build PySEAL-
  python setup.py build_ext -i

4 Dataset Encryption-

- Dataset Credit-card Fraud dataset Encryption was done using pythonscript belowpsedo code for it.

```
Import necessary libraries and modules (os, numpy, pandas, time,
SEAL-related classes)

Function setup_seal_environment():
    Initialize encryption parameters for CKKS scheme
    Set polynomial modulus degree to 8192
    Set coefficient modulus with specific bit sizes [60, 40, 40, 60]

    Create SEAL context with encryption parameters
    Create CKKSEncoder with the SEAL context
    Generate keys using KeyGenerator:
        Create public key
        Retrieve secret key
```

```
    Create Encryptor using public key and context
    Create Decryptor using secret key and context

    Return context, encoder, encryptor, decryptor

Function encrypt_data(encoder, encryptor, data_value):
    Convert data_value to a NumPy array with a float64 data type
    Set scale factor to 2^40
    Encode the data array using CKKSEncoder with the specified scale
    Encrypt the encoded data using Encryptor

    Return the ciphertext

Function process_dataset(dataset, columns, encoder, encryptor):
    For each specified column in the dataset:
        Initialize an empty list for encrypted values

        For each value in the column:
            Encrypt the value using encrypt_data()
            Convert the ciphertext to a string format
            Append the encrypted value to the list

        Replace the original column data with the encrypted values

    Return the modified dataset

Function main():
    Start a timer to measure the encryption process duration

    Call setup_seal_environment() to initialize SEAL components
    Load the dataset from a CSV file

    Print all column names in the dataset to verify them

    Specify the columns to encrypt, ensuring the names match those
in the dataset

    Try:
        Call process_dataset() to encrypt the specified columns
        Save the encrypted dataset to a new CSV file

        Measure and print the total time taken for encryption
    Except KeyError:
        Print an error message if a specified column is not found

If running as the main program:
    Call main()
```

credit_card	date	transactioı	Long	Lat	city	state	zipcode	credit_carı	cluster_lab	is_fraud
<seal.Ciphertext object at	11/09/2015 00:32	-0.33876	<seal.Ciphertext objec	<seal.Ciphertext object at 0x0(	Houston	PA	15342	0.565333	0	
<seal.Ciphertext object at	24/10/2015 22:23	0.137514	<seal.Ciphertext objec	<seal.Ciphertext object at 0x0(	Houston	PA	15342	0.565333	0	
<seal.Ciphertext object at	26/10/2015 18:19	-0.30049	<seal.Ciphertext objec	<seal.Ciphertext object at 0x0(	Houston	PA	15342	0.565333	0	
<seal.Ciphertext object at	22/10/2015 19:41	0.402484	<seal.Ciphertext objec	<seal.Ciphertext object at 0x0(	Houston	PA	15342	0.565333	0	
<seal.Ciphertext object at	26/10/2015 20:08	-0.11382	<seal.Ciphertext objec	<seal.Ciphertext object at 0x0(	Houston	PA	15342	0.565333	0	
<seal.Ciphertext object at	17/10/2015 21:28	0.285522	<seal.Ciphertext objec	<seal.Ciphertext object at 0x0(	Houston	PA	15342	0.565333	0	
<seal.Ciphertext object at	29/08/2015 18:34	0.293945	<seal.Ciphertext objec	<seal.Ciphertext object at 0x0(	Houston	PA	15342	0.565333	0	
<seal.Ciphertext object at	14/08/2015 21:34	0.096201	<seal.Ciphertext objec	<seal.Ciphertext object at 0x0(	Houston	PA	15342	0.565333	0	
<seal.Ciphertext object at	17/09/2015 19:20	0.073739	<seal.Ciphertext objec	<seal.Ciphertext object at 0x0(	Houston	PA	15342	0.565333	0	

- After running the python script for encryption credit card , long and lat columns are succesfully encrypted.

Anti-money laundering Encryption was done using python script below psedo code for it also.

```
Import necessary libraries and modules (os, numpy, pandas, time, SEAL-related classes)

Function setup_seal_environment():
    Initialize encryption parameters for CKKS scheme
    Set polynomial modulus degree to 8192
    Set coefficient modulus with specific bit sizes [60, 40, 40, 60]

    Create SEAL context with encryption parameters
    Create CKKSEncoder with the SEAL context
    Generate keys using KeyGenerator:
        Create public key
        Retrieve secret key

    Create Encryptor using public key and context
    Create Decryptor using secret key and context

    Return context, encoder, encryptor, decryptor

Function encrypt_data(encoder, encryptor, data_value):
    Convert data_value to a NumPy array with a float64 data type
    Set scale factor to 2^40
    Encode the data array using CKKSEncoder with the specified scale
    Encrypt the encoded data using Encryptor

    Return the ciphertext

Function process_dataset(dataset, columns, encoder, encryptor):
    For each specified column in the dataset:
        Initialize an empty list for encrypted values

        For each value in the column:
            Encrypt the value using encrypt_data()
            Convert the ciphertext to a string format
            Append the encrypted value to the list

        Replace the original column data with the encrypted values

    Return the modified dataset

Function main():
    Start a timer to measure the encryption process duration

    Call setup_seal_environment() to initialize SEAL components
    Load the dataset from a CSV file

    Print all column names in the dataset to verify them

    Specify the columns to encrypt, ensuring the names match those in the dataset
```

Try:
Call process_dataset() to encrypt the specified columns
Save the encrypted dataset to a new CSV file

Measure and print the total time taken for encryption
Except KeyError:
Print an error message if a specified column is not found

If running as the main program:
Call main()

- After running the python script for encryption, sender account and reciver account columns are succesfully encrypted.

Time	Date	Sender_account	Receiver_account	Amount	Payment_c	Received_	Sender_ba	Receiver_b	Payment_t	Is_launderi	Laundering_type
10:35:19	07/10/2022	<seal.Ciphertext object at 0	<seal.Ciphertext object at 0	1459.15	UK pounds	UK pounds	UK	UK	Cash Depo	0	Normal_Cash_Deposits
10:35:20	07/10/2022	<seal.Ciphertext object at 0	<seal.Ciphertext object at 0	6019.64	UK pounds	Dirham	UK	UAE	Cross-bord	0	Normal_Fan_Out
10:35:20	07/10/2022	<seal.Ciphertext object at 0	<seal.Ciphertext object at 0	14328.44	UK pounds	UK pounds	UK	UK	Cheque	0	Normal_Small_Fan_Out
10:35:21	07/10/2022	<seal.Ciphertext object at 0	<seal.Ciphertext object at 0	11895	UK pounds	UK pounds	UK	UK	ACH	0	Normal_Fan_In
10:35:21	07/10/2022	<seal.Ciphertext object at 0	<seal.Ciphertext object at 0	115.25	UK pounds	UK pounds	UK	UK	Cash Depo	0	Normal_Cash_Deposits
10:35:21	07/10/2022	<seal.Ciphertext object at 0	<seal.Ciphertext object at 0	5130.99	UK pounds	UK pounds	UK	UK	ACH	0	Normal_Group
10:35:23	07/10/2022	<seal.Ciphertext object at 0	<seal.Ciphertext object at 0	12176.52	UK pounds	UK pounds	UK	UK	ACH	0	Normal_Small_Fan_Out
10:35:23	07/10/2022	<seal.Ciphertext object at 0	<seal.Ciphertext object at 0	56.9	UK pounds	UK pounds	UK	UK	Credit card	0	Normal_Small_Fan_Out

5. Uploading each Encrypted dataset to respective cloud platform and Migrating to AWS S3 bucket.

1- Crating AWS S3 bucket
- Go to [AWS Management Console](#) and log in with your credentials.
- Search for S3 in search panel
- Click on create bucket
- Proceed with default settings and click create bucket
- Bucket created

2- AWS credentials
- Click on your profile and then click on security credentials
- Scroll down to Access key and create one and along with that you will see get you Secrect access key also right over there.
- Note down AWS Access key, AWS secret Access key and bucket name. As these credentials will be used in AZURE and .boto file in GCP , so that data from these cloud can easily be migrated to AWS S3 bucket.

3 - Credit card fraud datasets uploaded to Azure cloud platform.
Step 1: Create a Storage Account
- First, you need a storage account where your storage container (bucket) will reside.
- Log in to your Azure Portal (portal.azure.com).
- In the Azure Portal, click on "Create a resource" in the top left corner.
- Search for "Storage account" and select it from the results.
- Click "Create".
- Fill in the required details:
- Subscription: Choose your Azure subscription.

- Resource Group: Create a new resource group or select an existing one.
- Storage Account Name: Choose a unique name.
- Location: Choose the region for your storage to reside in.
- Performance: Choose between Standard and Premium (Standard is sufficient for most cases).
- Account kind: Choose "StorageV2 (general purpose v2)" as it supports all the latest features.
- Replication: Choose a replication strategy based on your durability and availability needs (e.g., LRS for locally redundant storage).
- Review any additional options, adjust as necessary, and click "Review + create".
- Once validated, click "Create". It may take a few minutes for the storage account to be set up.
- Step 2: Create a Container (Bucket)
- Once your storage account is ready, you can create a container within it.
- Go to your newly created storage account in the Azure Portal.
- Under the "Data storage" section, click on "Containers".
- Click "+ Container" to create a new container.
- Enter a name for your container.
- Set the Public access level:
- Private (no anonymous access)
- Blob (anonymous read access for blobs only)
- Container (anonymous read access for containers and blobs)
- Click "Create" to create the container.
- Step 3: Upload Data to the Container
- Now you’re ready to upload data to your new container.
- Open the container you just created by clicking on its name.
- Inside the container interface, click on "Upload".
- A blade will open where you can select files:
- Click on the "Folder" icon to select files from your computer.
- Choose the files you want to upload.
- You can set additional options such as:
- Overwrite if the file already exists.
- Access tier (Hot, Cool, or Archive) depending on how frequently you expect to access this data.

- Click "Upload" to start uploading your files.

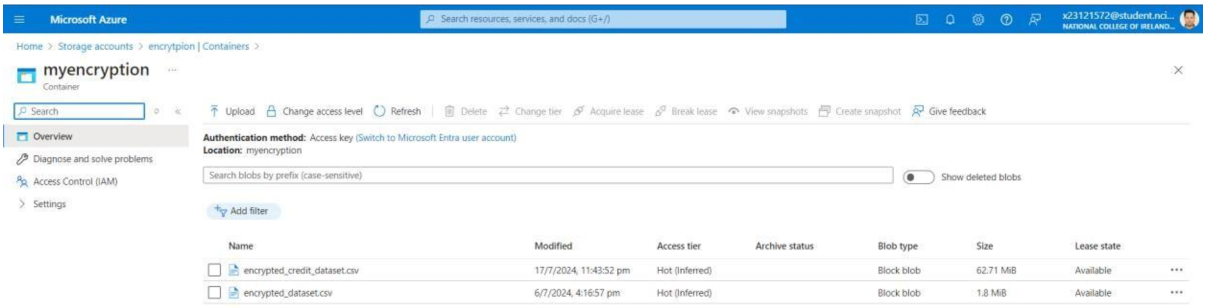


Figure- Encrypted dataset uploaded to Azure

4- For migrating Data from Azure to AWS S3 bucket we deployed a python script.Figure is the python script and Figure shows dataset sent to AWS S3 bucket

```python
import os
from azure.storage.blob import BlobServiceClient
import boto3

# Azure configuration
AZURE_ACCOUNT_NAME = 'encrytpion'  # Replace with your Azure account name
AZURE_ACCOUNT_KEY = '+q98dk+4DRgAjIjcuWkXb3n34AkIIpWI74pyNbgfj+z7Y6hWrG2SAdiyZxP0DDlx/MqaWRoBKIeJ+AStefumIw=='    # Replace with your Azure account key
AZURE_CONTAINER_NAME = 'myencryption'    # Replace with your Azure container name
BLOB_NAME = 'encrypted_credit_dataset.csv'         # Replace with the name of the blob you want to transfer

# AWS configuration
AWS_ACCESS_KEY_ID = 'AKIA6ODVAA2QQLWL4GB3'     # Replace with your AWS access key ID
AWS_SECRET_ACCESS_KEY = 'qGqjsQSOvNeaSBb9ofoIaSvniVIiic/YEgq1Eolj' # Replace with your AWS secret access key
AWS_BUCKET_NAME = 'myaws0222'         # Replace with your AWS S3 bucket name

# Initialize Azure Blob Service Client
azure_connection_string = f"DefaultEndpointsProtocol=https;AccountName={AZURE_ACCOUNT_NAME};AccountKey={AZURE_ACCOUNT_KEY};EndpointSuffix=core.windows.net"
blob_service_client = BlobServiceClient.from_connection_string(azure_connection_string)
blob_client = blob_service_client.get_blob_client(container=AZURE_CONTAINER_NAME, blob=BLOB_NAME)

# Download the blob from Azure
local_file_name = BLOB_NAME
with open(local_file_name, "wb") as download_file:
    download_file.write(blob_client.download_blob().readall())

# Initialize the AWS S3 client
s3_client = boto3.client(
    's3',
    aws_access_key_id=AWS_ACCESS_KEY_ID,
    aws_secret_access_key=AWS_SECRET_ACCESS_KEY
)

# Upload the file to AWS S3
s3_client.upload_file(local_file_name, AWS_BUCKET_NAME, local_file_name)

# Optional: Remove the local file after upload to clean up
os.remove(local_file_name)

print(f"File {BLOB_NAME} successfully transferred from Azure to AWS S3 bucket {AWS_BUCKET_NAME}.")
```

encrypted_credit_dataset.csv	csv	August 10, 2024, 10:12:32 (UTC+01:00)	62.7 MB	Standard

6. Uploading Anti-money laundering dataset to Google cloud platform and migrating it to AWS S3 bucket

- Go to GCP and create new project.

- Select your created project and the select BigQuery
- In BigQuerry next to your project name there will be three dots click on that and select create table
- Then click on Create table option and just keep the default settings same and just give name to your table and click create table
- Now go to cloud storage option in GCP and create a bucket, proceed with default setting and create the bucket.
- Now we have to unload the table created into the GCP bucket and upload your encrypted aml dataset.
- Now go back to big querry table and select export option and in that select export to GCS.
- Select browse, select your bucket and then select your uploaded dataset and then its is exported.
  - Now click on Activate cloud shell option which is next to present box symbol. Run below listed commands in the Figure . These commands are available in project directory which is uploaded in github under the file name GCP to S3 command.txt file.

```
gcloud config set project playground-s-11-ggtt6547  #<Provide your Google Cloud Project ID>

gsutil ls #<ServiceException: 401 Anonymous caller does not have storage.buckets.list access to the Google Cloud project>

gsutil config

gcloud auth login #<Go to the link, give necessary permission and then get the verification code to enter in shell console>

gsutil ls       #<To list all the GCP Cloud Storage existing Buckets>

cd ~

touch .boto             #<Create configuration file>

echo [Credentials] >> ~/.boto
echo aws_access_key_id = ENTER_AWS_ACCOUNT_ACCESS_KEY_ID >> ~/.boto
echo aws_secret_access_key = ENTER_AWS_ACCOUNT_SECRET_ACCESS_KEY_ID >> ~/.boto

cat .boto #<To print the values you echoe in the .boto file>

gsutil cp -r gs://product_outbound/products.csv s3://gcproducts/  #<"-r" is to copy multiple files recursively>
```

Figure- GCP command

- Once the commands are run successfully file will be transferred to AWS S3 bucket from GCP. See figure

☐	encrypted_MLdataset_full_final.csv	csv	August 13, 2024, 23:39:59 (UTC+01:00)	178.0 MB	Standard

Figure – GCP to S3 file transfered

7. IAM ROLE & EC2 Creation

1- IAMROLE

Step 1: Log in to the AWS Management Console

Log In: Go to AWS Management Console and log in with your credentials.

Step 2: Navigate to IAM Service

Open IAM: In the AWS Management Console, find and open the **IAM** service by typing "IAM" in the "Find Services" box or selecting it from the "Services" menu.

Step 3: Create a New Role

Create Role: In the IAM dashboard, go to the "Roles" section and click **"Create role"**.

Select Trust Relationship: Choose the service that will use this role.

Click Next: Proceed to the next step.

Step 4: Attach Permissions Policies

Attach Policies: In the "Attach permissions policies" section, search for and select:

AmazonS3FullAccess: Provides full access to Amazon S3.

AmazonSageMakerFullAccess: Provides full access to Amazon SageMaker.
Check the Boxes next to these policies to attach them to the role.
Click Next: Move on to the next step.
Step 5: Review and Create the Role
Name the Role: Provide a name for your role, such as SageMakerAndS3FullAccess.

1) **Description**: Optionally, add a description to detail the role's purpose, like "Provides full access to S3 and SageMaker for specific services."
2) **Review**: Check the configurations to ensure everything is correct.
3) **Create Role**: Click the **"Create role"** button to finalize the role creation.

2- EC2 creation

Step 1: Launch an instance

You can launch an EC2 instance using the AWS Management Console as described in the following procedure. This tutorial is intended to help you quickly launch your first instance, so it doesn't cover all possible options.

To launch an instance

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the navigation bar at the top of the screen, we display the current AWS Region — for example, **Ohio**. You can use the selected Region, or optionally select a Region that is closer to you.
3. From the EC2 console dashboard, in the **Launch instance** pane, choose **Launch instance**.
4. Under **Name and tags**, for **Name**, enter a descriptive name for your instance.
5. Under **Application and OS Images (Amazon Machine Image)**, do the following:
    a. Choose **Quick Start**, and then choose the operating system (OS) for your instance. For your first Linux instance, we recommend that you choose Amazon Linux.
    b. From **Amazon Machine Image (AMI)**, select an AMI that is marked **Free Tier eligible**.
6. Under **Instance type**, for **Instance type**, choose t2.micro, which is eligible for the Free Tier. In Regions where t2.micro is not available, t3.micro is eligible for the Free Tier.
7. Under **Key pair (login)**, for **Key pair name**, choose an existing key pair or choose **Create new key pair** to create your first key pair.
8. Under **Network settings**, notice that we selected your default VPC, selected the option to use the default subnet in an Availability Zone that we choose for you, and configured a security group with a rule that allows connections to your instance from anywhere. For your first instance, we recommend that you use the default settings. Otherwise, you can update your network settings as Under **Configure storage**, notice that we configured a root volume but no data volumes. This is sufficient for test purposes.
9. Review a summary of your instance configuration in the **Summary** panel, and when you're ready, choose **Launch instance**.
10. If the launch is successful, choose the ID of the instance from the **Success** notification to open the **Instances** page and monitor the status of the launch.
11. Select the check box for the instance. The initial instance state is pending. After the instance starts, its state changes to running. Choose the **Status and alarms** tab. After your instance passes its status checks, it is ready to receive connection requests.

Step2 – Connect to instance

The procedure that you use depends on the operating system of the instance. If you can't connect to your instance, see Troubleshoot issues connecting to your Amazon EC2 Linux instance for assistance.

Linux instances

You can connect to your Linux instance using any SSH client. If you are running Windows on your computer, open a terminal and run the ssh command to verify that you have an SSH client installed. If the command is not found, install OpenSSH for Windows.

To connect to your instance using SSH

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the navigation pane, choose **Instances**.
3. Select the instance and then choose **Connect**.
4. On the **Connect to instance** page, choose the **SSH client** tab.
5. (Optional) If you created a key pair when you launched the instance and downloaded the private key (.pem file) to a computer running Linux or macOS, run the example **chmod** command to set the permissions for your private key.
6. Copy the example SSH command. The following is an example, where *key-pair-name*.pem is the name of your private key file, *ec2-user* is the user name associated with the image, and the string after the @ symbol is the public DNS name of the instance.

ssh -i *key-pair-name*.pem *ec2-user@ec2-198-51-100-1.us-east-2.compute.amazonaws.com*

7. In a terminal window on your computer, run the **ssh** command that you saved in the previous step. If the private key file is not in the current directory, you must specify the fully-qualified path to the key file in this command.

The following is an example response:

The authenticity of host 'ec2-198-51-100-1.us-east-2.compute.amazonaws.com (198-51-100-1)' can't be established.

ECDSA key fingerprint is l4UB/neBad9tvkgJf1QZWxheQmR59WgrgzEimCG6kZY.

Are you sure you want to continue connecting (yes/no)?

8. (Optional) Verify that the fingerprint in the security alert matches the instance fingerprint contained in the console output when you first start an instance. To get the console output, choose **Actions**, **Monitor and troubleshoot**, **Get system log**. If the fingerprints don't match, someone might be attempting a man-in-the-middle attack. If they match, continue to the next step.
9. Enter **yes**.

8. SMPC Framework Creation

Secure Multi-Party Computation (SMPC):

Secure Multi Party Computation (SMPC) is a cryptographic technique through which several parties can jointly perform computation on the private data sent by them but the data remains unseen and insecure with other parties involved in the computation. The sizes of the subgroups which the parties get to know are just the output of the function and nothing else, thus they do not compromise on data privacy and security. For implementing it we will be using MPyc python library.

Why Use MPyC:

MPyC is used because it allow the implementation of SMPC protocols in Python and, therefore, it is more comfortable to build secure applications. Its asynchronous characteristic corresponds to beneficial communication, and it makes easy organization and the realization of the computations that preserve the privacy of the information of every individual. This makes MPyC useful in a variety of applications, from statistical analysis, to machine learning, where privacy and security are paramount.

- Connect to the EC2 instance created and install MPyC library in it by running command.
  pip install mpyc
- Now download the both Encrypted data from the S3 bucket using AWS cli, in ec2 aws cli is already present for verifying write the command shown in figure.

```
[ec2-user@ip-172-31-29-140 ~]$ aws --version
aws-cli/2.15.30 Python/3.9.16 Linux/6.1.97-104.177.amzn2023.x86_64 source/x86_64.amzn.2023 prompt/off
[ec2-user@ip-172-31-29-140 ~]$
```

Figure

- IF not present then run command:

  Update Your Instance
  sudo yum update -y
  Install AWS CLI
  For Amazon Linux :
  sudo yum install aws-cli -y
  Verify Installation
  aws --version
  Configure AWS CLI
  aws configure
  This command will prompt you to enter:
    - AWS Access Key ID
    - AWS Secret Access Key
    - Default region name
    - Default output format

- Now create a project directory in ec2 instance
- Download encrypted data from S3 to ec2 using command
  aws s3 cp s3://bucket.name/file.name ./

- Once both datasets are downloaded we will create python script for SMPC using MPyC library to extract necessary features from the encrypted data. Below is the script for encrypted anti-money laundering dataset.

```
from mpyc.runtime import mpc
import pandas as pd

async def secure_interaction_count_and_sum(data, amounts):
    """Count interactions and sum amounts securely using MPC."""
    secint = mpc.SecInt()  # Secure integer type for interaction counts
    secfxp = mpc.SecFxp() # Secure fixed-point type for transaction amounts
    interaction_dict = {}
    sum_dict = {}

    for (sender, receiver), amount in zip(data, amounts):
        pair = (str(sender), str(receiver))
```

```
        if pair not in interaction_dict:
            interaction_dict[pair] = secint(1)
            sum_dict[pair] = secfxp(amount)
        else:
            interaction_dict[pair] += 1
            sum_dict[pair] += secfxp(amount)

    # Decrypt results securely
    interaction_results = {pair: await mpc.output(count) for pair, count in
interaction_dict.items()}
    sum_results = {pair: await mpc.output(sum_amount) for pair, sum_amount in
sum_dict.items()}
    return interaction_results, sum_results

async def main():
    await mpc.start()  # Start MPC environment

    # Load your dataset
    dataset_path = 'encrypted_MLdataset.csv'
    dataset = pd.read_csv(dataset_path)

    # Prepare encrypted sender, receiver data, and transaction amounts for secure computation
    encrypted_interactions = list(zip(dataset['Sender_account'], dataset['Receiver_account']))
    transaction_amounts = dataset['Amount'].tolist()  # Make sure this is the correct column
name

    # Compute interaction counts and total transaction amounts securely
    interaction_counts, total_amounts = await
secure_interaction_count_and_sum(encrypted_interactions, transaction_amounts)

    # Map interaction counts and total transaction amounts back to the dataset
    dataset['interaction_count'] = dataset.apply(lambda row:
interaction_counts.get((str(row['Sender_account']), str(row['Receiver_account'])), 0), axis=1)
    dataset['total_transaction_amount'] = dataset.apply(lambda row:
total_amounts.get((str(row['Sender_account']), str(row['Receiver_account'])), 0.0), axis=1)

    # Save the enriched dataset
    dataset.to_csv('enriched_MLdataset_full.csv', index=False)
    print("Dataset has been enriched and saved.")

    await mpc.shutdown()  # Shutdown MPC environment

# Run the MPC computation
mpc.run(main())
```

Below is the script for encrypted credit-card fraud dataset.

```
from mpyc.runtime import mpc
import pandas as pd

async def secure_interaction_count_sum_and_average(data, amounts, locations):
```

```
"""Count transactions, sum amounts, and compute average by location securely using
MPC."""
secint = mpc.SecInt() # Secure integer type for transaction counts
secfxp = mpc.SecFxp(64) # Secure fixed-point type for transaction amounts and averages
transaction_dict = {}
sum_dict = {}
location_sum_dict = {}
location_count_dict = {}

for credit_card, amount, location in zip(data, amounts, locations):
card_str = str(credit_card)
location_str = str(location)

if card_str not in transaction_dict:
transaction_dict[card_str] = secint(1)
sum_dict[card_str] = secfxp(amount)
else:
transaction_dict[card_str]      +=      1
sum_dict[card_str] += secfxp(amount)

if location_str not in location_sum_dict:
location_sum_dict[location_str] = secfxp(amount)
location_count_dict[location_str] = secint(1)
else:
location_sum_dict[location_str] += secfxp(amount)
location_count_dict[location_str] += 1

# Decrypt results securely
transaction_results = {card: await mpc.output(count) for card, count in
transaction_dict.items()}
sum_results = {card: await mpc.output(total) for card, total in sum_dict.items()}
average_location_results = {loc: await mpc.output(total /
mpc.convert(location_count_dict[loc], secfxp)) for loc, total in location_sum_dict.items()}

return transaction_results, sum_results, average_location_results

async def main():
await mpc.start() # Start MPC environment

# Load your dataset
dataset_path = 'encrypted_credit_dataset.csv'
dataset = pd.read_csv(dataset_path)

# Prepare encrypted credit card, location data, and transaction amounts for secure
computation
encrypted_cards = dataset['credit_card'].tolist()
transaction_amounts = dataset['transaction_dollar_amount'].tolist()
locations = list(zip(dataset['Long'], dataset['Lat']))
```

```
# Compute transaction counts, total transaction amounts, and average transaction amounts by location securely
transaction_counts, total_amounts, average_amounts_by_location = await secure_interaction_count_sum_and_average(encrypted_cards, transaction_amounts, locations)

# Map transaction counts, total transaction amounts, and average transaction amounts by location back to the dataset
dataset['transaction_count'] = dataset['credit_card'].apply(lambda card: transaction_counts.get(str(card), 0))
dataset['total_transaction_amount'] = dataset['credit_card'].apply(lambda card: total_amounts.get(str(card), 0.0))
dataset['average_transaction_amount_by_location'] = dataset.apply(lambda row: average_amounts_by_location.get(str((row['Long'], row['Lat'])), 0.0), axis=1)

# Save the enriched dataset
dataset.to_csv(' enriched_credit_dataset_full.csv', index=False)
print("Dataset has been enriched and saved.")

await mpc.shutdown() # Shutdown MPC environment

# Run the MPC computation
mpc.run(main())
```

- Now both **'enriched_MLdataset_full.csv' and** ' enriched_credit_dataset_full.csv' are the final datasets with extracted features which will now be used for training and validating machine learning model for detecting fradulent activities. And will be uploaded to AWS S3 bucket using command:

  aws s3 cp ~/file.name s3://bucketname/

9. AWS SAGEMAKER

To create a SageMaker notebook instance

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose Notebook instances, and then choose Create notebook instance.
3. On the Create notebook instance page, provide the following information (if a field is not mentioned, leave the default values):
   a. For Notebook instance name, type a name for your notebook instance.
   b. For Notebook Instance type, choose ml.t2.medium. This is the least expensive instance type that notebook instances support, and is enough for this exercise. If a ml.t2.medium instance type isn't available in your current AWS Region, choose ml.t3.medium.
   c. For Platform Identifier, choose a platform type to create the notebook instance on. This platform type defines the Operating System and the JupyterLab version that your notebook instance is created with. For information about platform identifier type, see Amazon Linux 2 notebook instances. For information about JupyterLab versions, see JupyterLab versioning.

d. For IAM role, choose Create a new role, and then choose Create role. This IAM role automatically gets permissions to access any S3 bucket that has sagemaker in the name. It gets these permissions through the AmazonSageMakerFullAccess policy, which SageMaker attaches to the role.

4. Choose Create notebook instance.

In a few minutes, SageMaker launches a notebook instance and attaches a 5 GB of Amazon EBS storage volume to it. The notebook instance has a preconfigured Jupyter notebook server, SageMaker and AWS SDK libraries, and a set of Anaconda libraries.

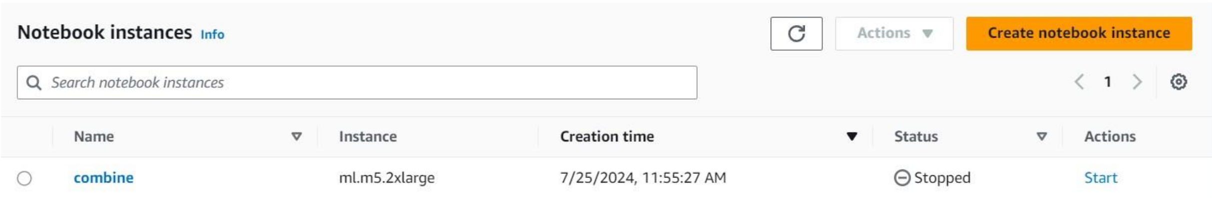


Figure- SAGEMAKER Notebook

Model traning script for all three datasets is uploaded in Github.

Results of each datasets ML model performance.

DATASET1- Anti money laundering dataset

With SMOTE -

```
Class distribution in the validation set:
Is_laundering
0    209524
1       191
Name: count, dtype: int64
Decision Tree (Training) Metrics:
Accuracy: 99.93%
Balanced Accuracy: 99.51%
Precision: 58.40%, Recall: 99.08%, F1 Score: 73.49%
MSE: 0.0006520762130465311, RMSE: 0.02553578299262686
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    838094
           1       0.58      0.99      0.73       765

    accuracy                           1.00    838859
   macro avg       0.79      1.00      0.87    838859
weighted avg       1.00      1.00      1.00    838859

Confusion Matrix:
[[837554    540]
 [     7    758]]
```

```
Decision Tree (Validation) Metrics:
Accuracy: 99.93%
Balanced Accuracy: 99.70%
Precision: 56.05%, Recall: 99.48%, F1 Score: 71.70%
MSE: 0.0007152564194263644, RMSE: 0.026744278255850622
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    209524
           1       0.56      0.99      0.72       191

    accuracy                           1.00    209715
   macro avg       0.78      1.00      0.86    209715
weighted avg       1.00      1.00      1.00    209715

Confusion Matrix:
[[209375    149]
 [     1    190]]
```

```
Random Forest (Training) Metrics:
Accuracy: 100.00%
Balanced Accuracy: 100.00%
Precision: 100.00%, Recall: 100.00%, F1 Score: 100.00%
MSE: 0.0, RMSE: 0.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    838094
           1       1.00      1.00      1.00       765

    accuracy                           1.00    838859
   macro avg       1.00      1.00      1.00    838859
weighted avg       1.00      1.00      1.00    838859

Confusion Matrix:
[[838094      0]
 [     0    765]]
```

```
Random Forest (Validation) Metrics:
Accuracy: 99.99%
Balanced Accuracy: 94.76%
Precision: 100.00%, Recall: 89.53%, F1 Score: 94.48%
MSE: 9.536752259018191e-05, RMSE: 0.009765629656616204
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    209524
           1       1.00      0.90      0.94       191

    accuracy                           1.00    209715
   macro avg       1.00      0.95      0.97    209715
weighted avg       1.00      1.00      1.00    209715

Confusion Matrix:
[[209524      0]
 [    20    171]]
```

```
Logistic Regression (Training) Metrics:
Accuracy: 51.77%
Balanced Accuracy: 55.75%
Precision: 0.11%, Recall: 59.74%, F1 Score: 0.23%
MSE: 0.482337317713704, RMSE: 0.694505088328159
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.52      0.68    838094
           1       0.00      0.60      0.00       765

    accuracy                           0.52    838859
   macro avg       0.50      0.56      0.34    838859
weighted avg       1.00      0.52      0.68    838859

Confusion Matrix:
[[433789 404305]
 [   308    457]]
```

```
Logistic Regression (Validation) Metrics:
Accuracy: 51.80%
Balanced Accuracy: 57.83%
Precision: 0.12%, Recall: 63.87%, F1 Score: 0.24%
MSE: 0.4820065326752974, RMSE: 0.6942669030533556
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.52      0.68    209524
           1       0.00      0.64      0.00       191

    accuracy                           0.52    209715
   macro avg       0.50      0.58      0.34    209715
weighted avg       1.00      0.52      0.68    209715

Confusion Matrix:
[[108509 101015]
 [    69    122]]
```

With SMOTE and Bagging-

```
Bagging Decision Tree (Validation) Metrics:
Accuracy: 0.9992847435805736, Balanced Accuracy: 0.997026631094305, Precision: 0.56047197640118, Recall: 0.9947643979057592, F1 Score: 0.7169811320754716, MSE: 0.0007152564194263644,
RMSE: 0.026744278255850622
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    209524
           1       0.56      0.99      0.72       191

    accuracy                           1.00    209715
   macro avg       0.78      1.00      0.86    209715
weighted avg       1.00      1.00      1.00    209715

Confusion Matrix:
[[209375    149]
 [     1    190]]
```

```
Class distribution in the validation set:
Is_laundering
0    209524
1       191
Name: count, dtype: int64
Bagging Decision Tree (Training) Metrics:
Accuracy: 0.9993515000733139, Balanced Accuracy: 0.9970634613482061, Precision: 0.584934665641814, Recall: 0.9947712418300654, F1 Score: 0.7366892545982575, MSE: 0.000648499926686129
6, RMSE: 0.025465661717028475
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    838094
           1       0.58      0.99      0.74       765

    accuracy                           1.00    838859
   macro avg       0.79      1.00      0.87    838859
weighted avg       1.00      1.00      1.00    838859

Confusion Matrix:
[[837554    540]
 [     4    761]]
```

```
Bagging Random Forest (Training) Metrics:
Accuracy: 0.9999964237136396, Balanced Accuracy: 0.9980392156862745, Precision: 1.0, Recall: 0.996078431372549, F1 Score: 0.9980353634577603, MSE: 3.57628636040145e-06, RMSE: 0.001
11071784543175
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    838094
           1       1.00      1.00      1.00       765

    accuracy                           1.00    838859
   macro avg       1.00      1.00      1.00    838859
weighted avg       1.00      1.00      1.00    838859

Confusion Matrix:
[[838094      0]
 [     3    762]]
```

```
Bagging Random Forest (Validation) Metrics:
Accuracy: 0.9999094008535393, Balanced Accuracy: 0.950261780104712, Precision: 1.0, Recall: 0.900523560209424, F1 Score: 0.9476584022038568, MSE: 9.059914646067282e-05, RMSE: 0.00951
8358391060551
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    209524
           1       1.00      0.90      0.95       191

    accuracy                           1.00    209715
   macro avg       1.00      0.95      0.97    209715
weighted avg       1.00      1.00      1.00    209715

Confusion Matrix:
[[209524      0]
 [    19    172]]
```

```
Bagging Logistic Regression (Training) Metrics:
Accuracy: 0.514720590707139, Balanced Accuracy: 0.5579743726314819, Precision: 0.0011295661483758804, Recall: 0.6013071895424836, F1 Score: 0.0022548964340773675, MSE: 0.485279409292
8609, RMSE: 0.6966199891568292
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.51      0.68    838094
           1       0.00      0.60      0.00       765

    accuracy                           0.51    838859
   macro avg       0.50      0.56      0.34    838859
weighted avg       1.00      0.51      0.68    838859

Confusion Matrix:
[[431318 406776]
 [   305    460]]
```

```
Bagging Logistic Regression (Validation) Metrics:
Accuracy: 0.515165820279037, Balanced Accuracy: 0.5795137265010863, Precision: 0.0012090590964494948, Recall: 0.643979057591623, F1 Score: 0.002413586727235266, MSE: 0.4848341797200
9635, RMSE: 0.6963003516587482
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.52      0.68    209524
           1       0.00      0.64      0.00       191

    accuracy                           0.52    209715
   macro avg       0.50      0.58      0.34    209715
weighted avg       1.00      0.52      0.68    209715

Confusion Matrix:
[[107915 101609]
 [    68    123]]
```

Dataset2- credit card fraud

With SMOTE-

```
Class distribution in the dataset:
is_fraudulent
0    289107
1      5481
Name: count, dtype: int64
Class distribution in the training set:
is_fraudulent
0    231285
1      4385
Name: count, dtype: int64
Class distribution in the validation set:
is_fraudulent
0    57822
1     1096
Name: count, dtype: int64
Decision Tree (Training) Metrics:
Accuracy: 1.0, Balanced Accuracy: 1.0, Precision: 1.0, Recall: 1.0, F1 Score: 1.0, MSE: 0.0, RMSE: 0.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    231285
           1       1.00      1.00      1.00      4385

    accuracy                           1.00    235670
   macro avg       1.00      1.00      1.00    235670
weighted avg       1.00      1.00      1.00    235670

Confusion Matrix:
[[231285      0]
 [     0   4385]]
```

```
Decision Tree (Validation) Metrics:
Accuracy: 1.0, Balanced Accuracy: 1.0, Precision: 1.0, Recall: 1.0, F1 Score: 1.0, MSE: 0.0, RMSE: 0.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     57822
           1       1.00      1.00      1.00      1096

    accuracy                           1.00     58918
   macro avg       1.00      1.00      1.00     58918
weighted avg       1.00      1.00      1.00     58918

Confusion Matrix:
[[57822     0]
 [    0  1096]]
```

```
Random Forest (Training) Metrics:
Accuracy: 1.0, Balanced Accuracy: 1.0, Precision: 1.0, Recall: 1.0, F1 Score: 1.0, MSE: 0.0, RMSE: 0.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    231285
           1       1.00      1.00      1.00      4385

    accuracy                           1.00    235670
   macro avg       1.00      1.00      1.00    235670
weighted avg       1.00      1.00      1.00    235670

Confusion Matrix:
[[231285      0]
 [     0   4385]]
```

```
Random Forest (Validation) Metrics:
Accuracy: 1.0, Balanced Accuracy: 1.0, Precision: 1.0, Recall: 1.0, F1 Score: 1.0, MSE: 0.0, RMSE: 0.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     57822
           1       1.00      1.00      1.00      1096

    accuracy                           1.00     58918
   macro avg       1.00      1.00      1.00     58918
weighted avg       1.00      1.00      1.00     58918

Confusion Matrix:
[[57822     0]
 [    0  1096]]
```

```
Logistic Regression (Training) Metrics:
Accuracy: 0.9999915135570926, Balanced Accuracy: 0.999995676330069, Precision: 0.9995441075906086, Recall: 1.0, F1 Score: 0.9997720018239854, MSE: 8.48644290745534e-06, RMSE: 0.002
3149997417802
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    231285
           1       1.00      1.00      1.00      4385

    accuracy                           1.00    235670
   macro avg       1.00      1.00      1.00    235670
weighted avg       1.00      1.00      1.00    235670

Confusion Matrix:
[[231283      2]
 [     0   4385]]
```

```
Logistic Regression (Validation) Metrics:
Accuracy: 0.9999321090328932, Balanced Accuracy: 0.9999654110892048, Precision: 0.9963636363636363, Recall: 1.0, F1 Score: 0.9981785063752276, MSE: 6.789096710682644e-05, RMSE: 0.008
239597508788062
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     57822
           1       1.00      1.00      1.00      1096

    accuracy                           1.00     58918
   macro avg       1.00      1.00      1.00     58918
weighted avg       1.00      1.00      1.00     58918

Confusion Matrix:
[[57818     4]
 [    0  1096]]
```

With SMOTE and Bagging:

```
Bagging Decision Tree (Validation) Metrics:
Accuracy: 1.0, Balanced Accuracy: 1.0, Precision: 1.0, Recall: 1.0, F1 Score: 1.0, MSE: 0.0, RMSE: 0.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     57822
           1       1.00      1.00      1.00      1096

    accuracy                           1.00     58918
   macro avg       1.00      1.00      1.00     58918
weighted avg       1.00      1.00      1.00     58918

Confusion Matrix:
[[57822     0]
 [    0  1096]]
```

```
Class distribution in the training set:
is_fraudulent
0    231285
1      4385
Name: count, dtype: int64
Class distribution in the validation set:
is_fraudulent
0    57822
1     1096
Name: count, dtype: int64
Bagging Decision Tree (Training) Metrics:
Accuracy: 1.0, Balanced Accuracy: 1.0, Precision: 1.0, Recall: 1.0, F1 Score: 1.0, MSE: 0.0, RMSE: 0.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    231285
           1       1.00      1.00      1.00      4385

    accuracy                           1.00    235670
   macro avg       1.00      1.00      1.00    235670
weighted avg       1.00      1.00      1.00    235670

Confusion Matrix:
[[231285      0]
 [     0   4385]]
```

```
Bagging Random Forest (Training) Metrics:
Accuracy: 1.0, Balanced Accuracy: 1.0, Precision: 1.0, Recall: 1.0, F1 Score: 1.0, MSE: 0.0, RMSE: 0.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    231285
           1       1.00      1.00      1.00      4385

    accuracy                           1.00    235670
   macro avg       1.00      1.00      1.00    235670
weighted avg       1.00      1.00      1.00    235670

Confusion Matrix:
[[231285      0]
 [     0   4385]]
```

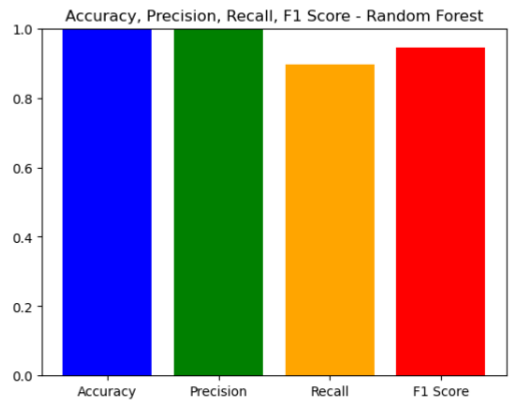
```
Bagging Random Forest (Validation) Metrics:
Accuracy: 1.0, Balanced Accuracy: 1.0, Precision: 1.0, Recall: 1.0, F1 Score: 1.0, MSE: 0.0, RMSE: 0.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     57822
           1       1.00      1.00      1.00      1096

    accuracy                           1.00     58918
   macro avg       1.00      1.00      1.00     58918
weighted avg       1.00      1.00      1.00     58918

Confusion Matrix:
[[57822     0]
 [    0  1096]]
```

```
Bagging Logistic Regression (Training) Metrics:
Accuracy: 0.9999915135570926, Balanced Accuracy: 0.999995676330069, Precision: 0.9995441075906086, Recall: 1.0, F1 Score: 0.9997720018239854, MSE: 8.48644290745534e-06, RMSE: 0.00291
3149997417802
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    231285
           1       1.00      1.00      1.00      4385

    accuracy                           1.00    235670
   macro avg       1.00      1.00      1.00    235670
weighted avg       1.00      1.00      1.00    235670

Confusion Matrix:
[[231283      2]
 [     0   4385]]
```

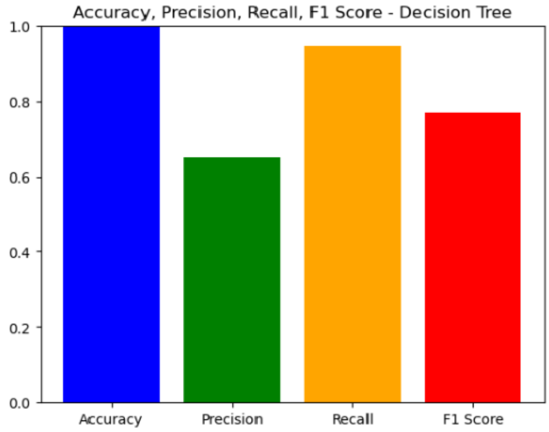
```
Bagging Logistic Regression (Validation) Metrics:
Accuracy: 0.9999151362911165, Balanced Accuracy: 0.999956763861506, Precision: 0.9954586739327884, Recall: 1.0, F1 Score: 0.9977241693218024, MSE: 8.486370888353304e-05, RMSE: 0.0092
12150068444013
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     57822
           1       1.00      1.00      1.00      1096

    accuracy                           1.00     58918
   macro avg       1.00      1.00      1.00     58918
weighted avg       1.00      1.00      1.00     58918

Confusion Matrix:
[[57817     5]
 [    0  1096]]
```

DATASET-3 combined dataset of both enriched AML and Credit card fraud datasets.

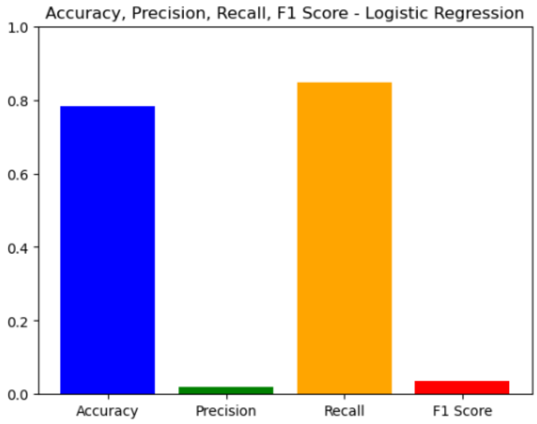
Without Smote-




```
Classification Report (Test) - Random Forest Model:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    267346
           1       1.00      0.90      0.95      1287

    accuracy                           1.00    268633
   macro avg       1.00      0.95      0.97    268633
weighted avg       1.00      1.00      1.00    268633

Accuracy Score (Test) - Random Forest Model:
99.95%
```

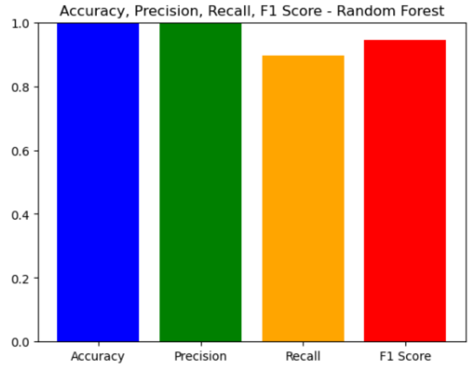



```
Classification Report (Test) - Decision Tree Model:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    267346
           1       0.65      0.95      0.77      1287

    accuracy                           1.00    268633
   macro avg       0.82      0.97      0.88    268633
weighted avg       1.00      1.00      1.00    268633

Accuracy Score (Test) - Decision Tree Model:
99.73%
```

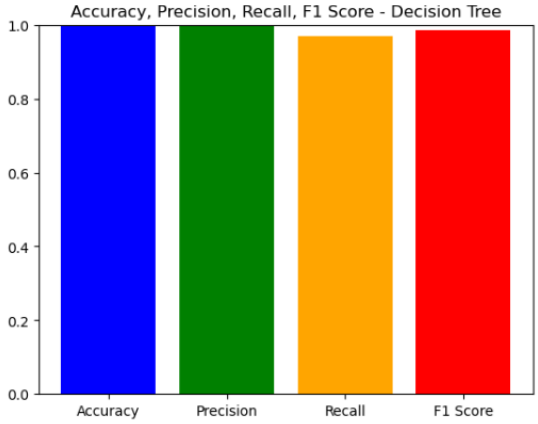



```
Classification Report (Test) - Logistic Regression Model:
              precision    recall  f1-score   support

           0       1.00      0.78      0.88    267346
           1       0.02      0.85      0.04      1287

    accuracy                           0.78    268633
   macro avg       0.51      0.82      0.46    268633
weighted avg       0.99      0.78      0.87    268633

Accuracy Score (Test) - Logistic Regression Model:
78.35%
```

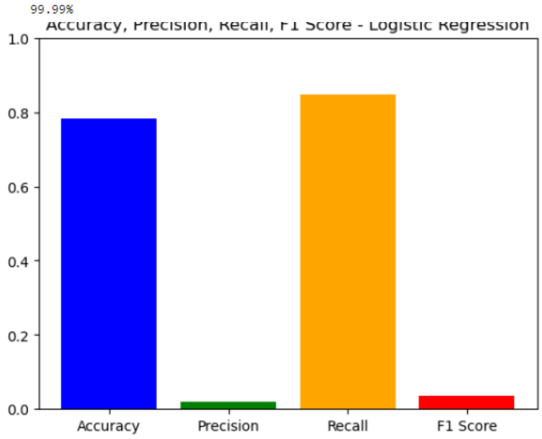
With SMOTE-




```
Classification Report (Test) - Random Forest Model:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    267346
           1       1.00      0.90      0.94      1287

    accuracy                           1.00    268633
   macro avg       1.00      0.95      0.97    268633
weighted avg       1.00      1.00      1.00    268633

Accuracy Score (Test) - Random Forest Model:
99.95%
```




```
Classification Report (Test) - Decision Tree Model:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    267346
           1       1.00      0.97      0.99      1287

    accuracy                           1.00    268633
   macro avg       1.00      0.99      0.99    268633
weighted avg       1.00      1.00      1.00    268633

Accuracy Score (Test) - Decision Tree Model:
99.99%
```




```
Classification Report (Test) - Logistic Regression Model:
              precision    recall  f1-score   support

           0       1.00      0.78      0.88    267346
           1       0.02      0.85      0.04      1287

    accuracy                           0.78    268633
   macro avg       0.51      0.82      0.46    268633
weighted avg       0.99      0.78      0.87    268633

Accuracy Score (Test) - Logistic Regression Model:
78.35%
```

References

Amazon.com. Available at: https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_create_for-user.html (Accessed: August 14, 2024)

Amazon.com. Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html (Accessed: August 14, 2024)

Amazon.com. Available at: https://docs.aws.amazon.com/sagemaker/latest/dg/gs-setup-working-env.html (Accessed: August 14, 2024).

(*Amazon.com*. Available at: https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html (Accessed: August 14, 2024).