BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

Work Integrated Learning Programs Division

Post Graduate Program in Artificial Intelligence and Machine Learning

ENSURING USER DATA PROTECTION IN MACHINE LEARNING MODELS

CAPSTONE PROJECT

Submitted in partial fulfillment of the requirements of the

Post Graduate Certification Program in Artificial Intelligence and Machine Learning

By

KARTHIK GN	**2022AIML047**
VIKAS VARDHAN SONI	**2022AIML010**
CHANCHAL SAHU	**2022AIML028**
NAMAN GUPTA	**2020AIML588**

Under the supervision of

PROF. SUDARDHAN S. DESHMUKH

Project work carried out at

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE Pilani (Rajasthan) INDIA

(7th OCT 2023)

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI SECOND SEMESTER 2022-23

PCAM ZC321 CAPSTONE PROJECT

Project Title: **ENSURING USER DATA PROTECTION IN MACHINE LEARNING MODELS**

Name of Mentor: SUDARSHAN S. DESHMUKH

Name of Student's:	**KARTHIK GN**
	VIKAS VARDHAN SONI
	CHANCHAL SAHU
	NAMAN GUPTA

ID No. of	:	**2022AIML047**
Student's		**2022AIML010**
		2022AIML028
		2020AIML588

ACKNOWLEDGEMENTS

We express our deep sense of gratitude to BITS, Pilani, for providing an opportunity to pursue our project on generative AI model successfully.

We thank our mentor, Prof. Sudarshan S. Deshmukh, for the constant support and guidance which resulted in the successful completion of the project within the specified time. His unflinching help and encouragement were a constant source of inspiration to us.

We would also like to thank all our professors, who taught us the basic and advanced concepts of artificial intelligence and machine learning in such a nuanced manner during this certification program.

We would also like to thank our respective families and organizations for supporting us.

KARTHIK GN	2022AIML047
VIKAS VARDHAN SONI	2022AIML010
CHANCHAL SAHU	2022AIML028
NAMAN GUPTA	2020AIML588

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

CERTIFICATE

This is to certify that the Capstone Project entitled "ENSURING USER DATA PROTECTION IN MACHINE LEARNING MODELS" and submitted by Mr. Karthik GN (2022AIML047), Mr. Vikas Vardhan Soni (2022AIML010), Mrs. Chanchal Sahu (2022AIML028), and Mr. Naman Gupta (2020AIML588) in partial fulfilment of the requirements of PCAM ZC321 Capstone Project, embodies the work done by him/her under my supervision.

Place : BITS Pilani, Hyderabad

Signature of the Mentor

[illegible]

Date : 7th. Oct. 2023

Name: Sudarshan S. Deshmukh

ABSTRACT

This project presents a methodology to credit risk assessment in banking that ensures privacy preservation using a combination of Generative Adversarial Network (GANs), Deterministic Encryption and Binary Cross Entropy classifier. The increasing digitization of banking services has led to an abundance of data, which, if utilized correctly, can significantly enhance credit risk assessment. However, the sensitive nature of banking data necessitates stringent privacy measures. Our method addresses this challenge by implementing a machine learning model that can learn from encrypted data, thereby maintaining the confidentiality of the information.

The proposed model employs Keras GAN to generate synthetic data that closely mimic the real-world banking data. This synthetic data is encrypted using deterministic encryption, allowing computations to be performed directly on this encrypted data. The machine learning model using a BCE classifier, is trained on this encrypted synthetic data, ensuring that the original sensitive data remains secure. Conditional GANs were also being explored to generate synthetic data with conditions as class labels which will closely mimic our original data set of 1000 records.

Our approach offers a solution that balances the need of advanced risk assessment techniques with the essential requirement of privacy preservation. The model is deployed as an API endpoint using Flask, providing a user-friendly interface for real-world application.

The proposed method’s efficacy is evaluated using various performance metrics like confusion matrix, classification report and loss calculations, demonstrating its potential to effectively assess credit risk without compromising data privacy.

This research contributes to the ongoing discourse on privacy-preserving machine learning in the banking sector. This also includes homomorphic encryption analysis using Seal, Tenseal which are fully homomorphic encryption and Pailler which is a partial homomorphic encryption. These encryption libraries are on-going research and only limited operations like addition and multiplications are allowed on this encrypted data.

Table of Contents

CHAPTER 1
INTRODUCTION

In today's digital age, data privacy and protection have become paramount concerns, especially when it comes to Machine Learning (ML) models used for sensitive tasks like credit risk assessment. Financial institutions gather vast amounts of customer data to assess creditworthiness, but ensuring the privacy of this data is critical. Privacy-preserving ML techniques aim to strike a balance between leveraging this valuable data for accurate risk assessment and safeguarding users' privacy.

User Data Protection and Privacy-Preserving ML for Credit Risk Assessment:

Financial institutions handle extensive customer data, including personal, financial, and behavioral information. Protecting this data is crucial to maintain customer trust and adhere to legal regulations like GDPR, CCPA, and various financial data protection laws.

Challenges in Data Privacy:

Data Sensitivity: Credit-related data is inherently sensitive, including income, debt, and spending habits. Any breach could lead to identity theft or financial fraud.

Regulatory Compliance: Financial institutions must comply with strict regulations governing data usage, making it essential to protect customer data and maintain legal compliance.

Data Sharing: Financial institutions often collaborate with credit bureaus and other agencies, requiring secure data sharing mechanisms.

Privacy-Preserving Techniques:

Differential Privacy: Adds noise to data, ensuring that the presence or absence of any individual's data doesn't significantly affect the model's output.

Federated Learning: Trains an ML model across multiple decentralized edge devices or servers holding local data samples without exchanging them. The global model is improved without raw data leaving user devices.

Homomorphic Encryption: Enables computations on encrypted data without decrypting it, ensuring that sensitive data remains encrypted during processing.

Secure Multi-Party Computation (SMPC): Allows parties to jointly compute a function over their inputs while keeping those inputs private. It ensures that no party learns anything beyond the output.

Tokenization and Encryption: Replaces sensitive data with unique tokens or encrypts the data, making it meaningless if intercepted without proper decryption keys.

<u>Benefits of Privacy-Preserving ML:</u>

Enhanced Customer Trust: Demonstrating a commitment to privacy can enhance customer trust, leading to stronger customer relationships.

Regulatory Compliance: Privacy-preserving techniques ensure compliance with data protection regulations, avoiding hefty fines and legal issues.

Innovation without Compromise: Institutions can innovate and improve credit risk models without compromising user privacy, fostering progress in financial technology.

1.1 PROJECT OBJECTIVE

The primary objective of this project is to create a privacy-preserving machine learning system tailored for secure utilization by financial institutions in the identification of credit risk associated with individuals. The focus will be on exploring encryption mechanisms implemented at the client end to ensure robust data security. Subsequently, the encrypted data will be utilized in conjunction with deep learning models, including neural networks, GANs, and autoencoders, to facilitate model training for classification and synthetic data generation purposes. The ultimate goal is to develop a seamlessly functioning model that maintains high accuracy, crucial for effectively managing potential credit risk exposure within the framework of encrypted data.

CHAPTER 2
LITERATURE REVIEW

In recent years, the use of machine learning models in banking for credit risk assessment has garnered significant attention, Studies like Khandani et al. (2010) and Lessmann et al. (2015) have demonstrated the potential of ML models in predicting credit risk with higher accuracy than traditional models. However, the sensitive nature of banking data has raised serious privacy concerns.

The concept of privacy-preserving machine learning has been introduced to address these concerns. Shokri and Shmatikov (2015) and Phong et al. (2018) have proposed different techniques to ensure privacy during the learning process. These methods, however often involve a. trade-off between privacy and model performance.

Generative Adversarial Network (GANs), introduced by Goodfellow et al. (2014), have been widely used to generate synthetic data that can mimic real-world data. Beauliu-Jones et al. (2017) demonstrated the application of GAN's in generating synthetic health record while preserving privacy. However, the use of GAN's in the banking sector for credit risk assessment is relatively unexplored.

Many types of homomorphic encryption were explored like Seal, Tenseal however due to limited support of operations like addition and multiplication which is yet to be extended to deep learning model, Deterministic Encryption (DE) was used. DE, a form of encryption uses a salt for encryption and decryption. DE maps identical input values to identical output values, has been used to ensure privacy in data storage and transmission. Popa et al. (2011) demonstrated the use of DE in building a practical and secure relational database.

Our research aims to bridge these gaps by proposing a novel method that combine GANs and DE along with BCE as classifier for privacy-preserving credit risk assessment. This method ensure privacy by training the ML model on synthetic data encrypted with DE, passing this to a BCE classifier for assessing good risk versus bad risk, there by preserving the confidentiality of the original sensitive data. We believe this approach will provide a practical solution to the privacy concerns in using ML for credit risk assessment in banking.

CHAPTER 3

PROJECT PREREQUISITES, CHALLENGES AND RISKS

3.1 PEOPLE, HARDWARE, AND SOFTWARE RESOURCES

People: This project was developed by a team of three members, including a project guide.

Hardware Resources: The hardware used for this project includes an Intel i5 7th gen processor, 16 GB of RAM, and an NVIDIA GeForce 940MX GPU.

Software Resources: The project was developed using Python programming language and the following libraries: PyTorch, torch Tensorflow, Keras, Flask, cryptography

3.2 PROJECT PREREQUISITES, POTENTIAL CHALLENGES AND RISKS IN THE PROJECT

3.2.1 PROJECT PREREQUISITES:

1. A good understanding of Python programming language.
2. Familiarity with: PyTorch, torch Tensorflow, Keras, Flask, cryptography
3. Experience with building APIs using Flask and deploying them.
4. Basic knowledge of GAN and Model building

3.2.2 CHALLENGES:

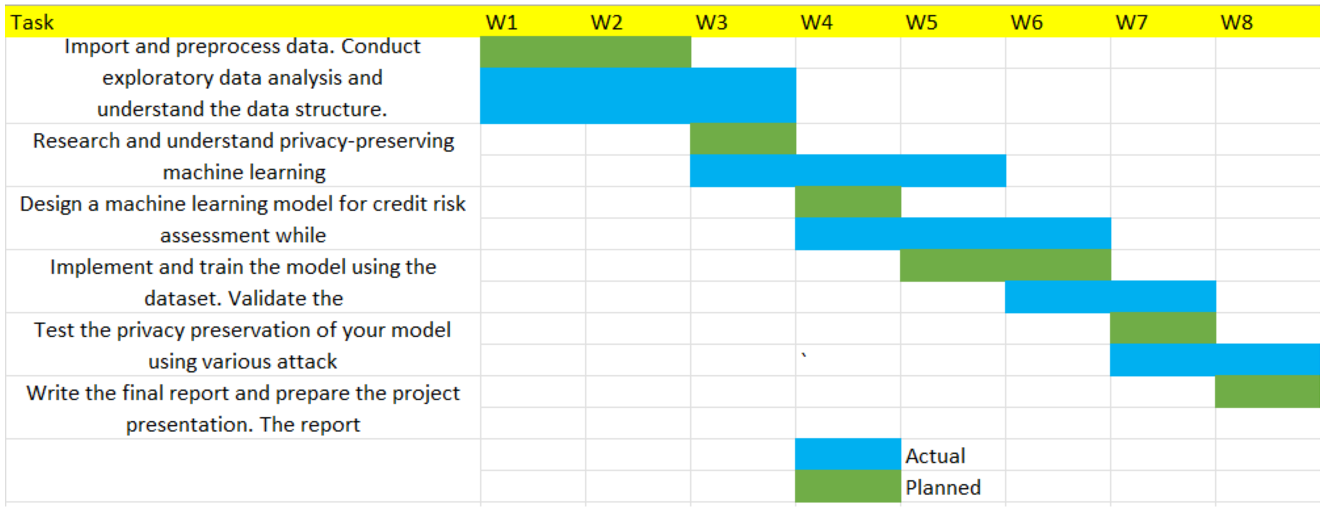
1. Training the model with small data.
2. Finding the right homomorphic encryption to feed it to the Deep Learning Model.
3. Potential class imbalance
4. Hyperparameter tuning for Deep Learning Model

3.2.3 RISK:

1. The training process may require a significant amount of time and computational resources, which can be expensive.
2. The model may not be able to generate synthetic data that are relevant to the real data.
3. The performance of the API may not scale well with the number of requests, leading to slower response time.

CHAPTER 4

Plan of Work



Task	W1	W2	W3	W4	W5	W6	W7	W8
Import and preprocess data. Conduct exploratory data analysis and understand the data structure.	🟩	🟩						
	🟦	🟦	🟦					
Research and understand privacy-preserving machine learning			🟩					
			🟦	🟦	🟦			
Design a machine learning model for credit risk assessment while				🟩				
				🟦	🟦	🟦		
Implement and train the model using the dataset. Validate the					🟩	🟩		
						🟦	🟦	
Test the privacy preservation of your model using various attack							🟩	
				`			🟦	🟦
Write the final report and prepare the project presentation. The report								🟩
				🟦	Actual			
				🟩	Planned			

CHAPTER 5

PRE-PROCESSING STEPS, MODELLING AND TECHNIQUES APPLIED

The project is implemented in the following steps:

1. Data Preparation.
2. Model Training.
3. Selecting a better model.
4. Develop API for the model.
5. Deployment.

Data Preparation:

In this step, we tried understanding the “German Credit” data by loading the data into a dataframe and then performing some basic operations. The data speaks about features that have sensitive data about credit profile of a person and the use-case under consideration involved training a machine learning model that can further help the organizations to understand credit risk against an individual.

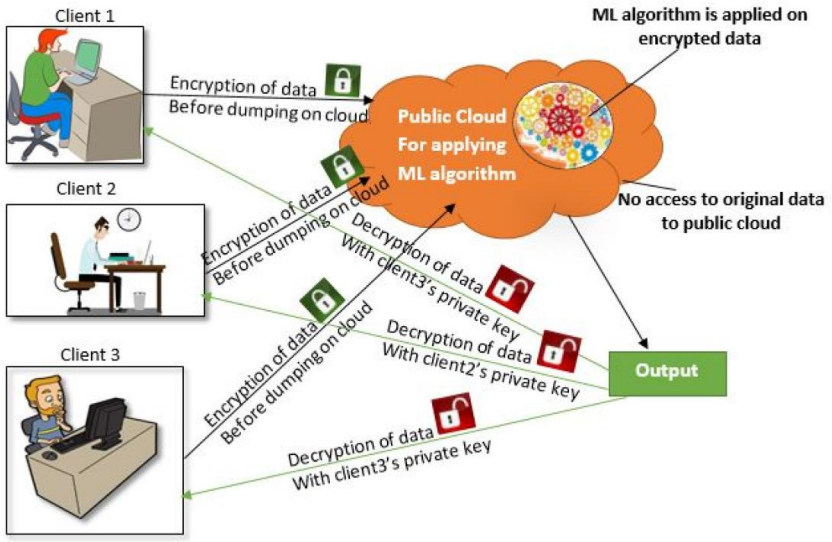
Since the data was partly categorical and partly numerical, we have to do a deep dive to understand data descriptions, ranges and ways to encode before the data can go into encryption. Following activity was done before any encryption:

- All categorical columns were encoded with Label encoder.

- Realized that standardization of numerical columns will not add value as we must take an encryption route followed by hot encoding. So, all numerical columns were kept as they are.

Model Training:

The objective here involves building a privacy-preserving ML driven model that can be used for credit risk assessment by banks and financial institutions. Below exhibit from Analytics Vidhya [1] captures the essence of exercise, where multiple clients can connect to the same model in a privacy preserving setup, passing encrypted data and get encrypted outputs which are decoded at the client end.


Client 1
Client 2
Client 3
Encryption of data Before dumping on cloud
Encryption of data Before dumping on cloud
Encryption of data Before dumping on cloud
Public Cloud For applying ML algorithm
ML algorithm is applied on encrypted data
No access to original data to public cloud
Decryption of data With client3's private key
Decryption of data With client2's private key
Decryption of data With client3's private key
Output

Few of the important steps in the entire scheme of things involved:

- Pre-processing the data to make it consistent.
- Encrypting Data using one of the encryption schemes.
- Separate test and train data
- Build deep learning classification model on train data.

- **Pre-processing the data**

  As mentioned above, the pre-processing involved label encoding the data so that all 21 columns have some numerical values which can be further used by encryption algorithms.

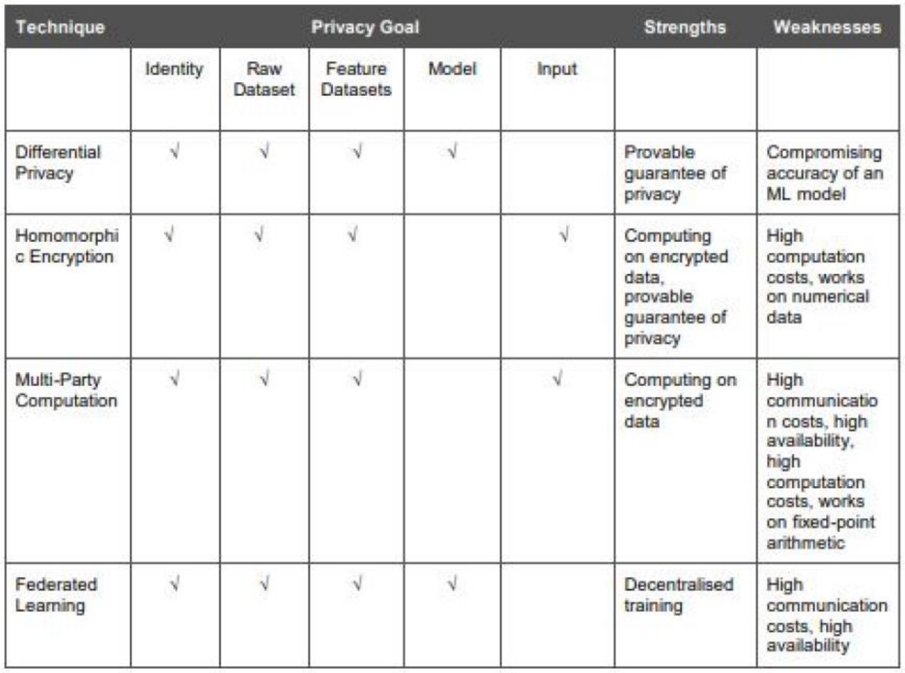
- **Data encryption approach**

  Data encryption is one of the most important steps in the whole process and we initially planned to use homomorphic encryption at client end. As per Duality tech blog [2], PPML (Privacy Preserving Machine Learning) involves two kinds of approaches:

- **Differential Privacy:** Differential Privacy is a data aggregation method that adds randomized “noise” to the data; data cannot be reverse engineered to understand the original inputs. While DP is used by Microsoft and open-source libraries to protect privacy in the creation and tuning of ML models, there is a distinct tradeoff when it comes to the data’s reliability.

- **Privacy Enhancing techniques,** which involves analyzing the data while its still encrypted. For ex, homomorphic encryption, and multi-party computation.

Below table captures various privacy preserving techniques:



Technique	Privacy Goal					Strengths	Weaknesses
	Identity	Raw Dataset	Feature Datasets	Model	Input		
Differential Privacy	√	√	√	√		Provable guarantee of privacy	Compromising accuracy of an ML model
Homomorphic Encryption	√	√	√		√	Computing on encrypted data, provable guarantee of privacy	High computation costs, works on numerical data
Multi-Party Computation	√	√	√		√	Computing on encrypted data	High communication costs, high availability, high computation costs, works on fixed-point arithmetic
Federated Learning	√	√	√	√		Decentralised training	High communication costs, high availability

Table Source: Overview of Privacy Preserving Techniques [7]

Approaches we tried for data encryption:

- Fully Homomorphic encryption using CKKS library
- Partial homomorphic encryption using Paillier library
- Fernet encryption
- Deterministic encryption with Cryptography library and hazmat layer

Key findings here are:

- Homomorphic encryption will not work for our case as we planned to use neural networks for classification. Output from both CKKS and Paillier encryption cannot be fed into Neural network.
- Fernet based encryption was not deterministic and thus cased different encryption strings to be generated. This increased complexity of the model for us as hot encoding increased the number of input columns.
- Finally, we agreed to use a deterministic encryption with Cryptography library which has a predefined initialization salt value so that encryption is always consistent.

- ## ML models built

We used torch library to build a classification model that trains on the German data train subset. The model was eventually tested upon test subset of German data.
Following other models and functions were built:

- GAN based model that is trained to generate real looking data. This was built to check the performance of the classifier as well as to corroborate the test data when any kind of class imbalance is there on the training data.
- We built a function (mapMyData), which will be used at the client end to encode and encrypt pure credit risk data to generate data which can be consumed at both GAN and classification function.

Classification function: Classification function is a torch based neural network which has following structure:

- Total 5 layers (Input + Output + 3 hidden layers)
- Activation function, Relu for input and hidden layers, Sigmoid for the output layer
- Output layer as only one node, because it generated a binary credit risk with sigmoid activation function.
- Loss function used is binary cross entropy loss as we have binary classification of the output label.
- We are using stochastic gradient descent as an optimizer with a learning rate of .05. We started with a learning rate of .001 but

finally got best value at .05 with maximum loss minimization. SGD is used because it provides good performance with deep neural networks.

The model was trained for some 1000 epoch on the training data generating around 75% plus accuracy.

Develop and Test API for the Model:

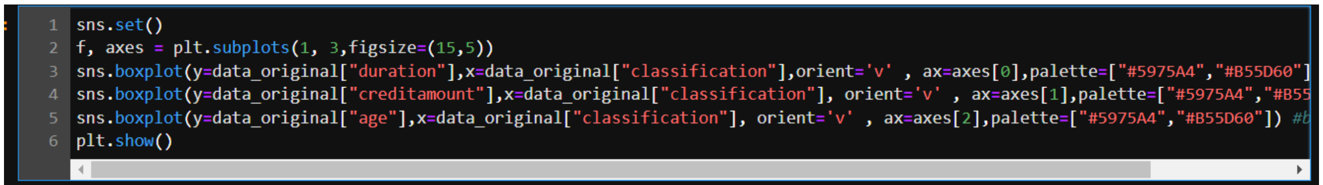
In this step, we deployed the trained model using Flask, a modern web framework for building APIs with Python. Flask is a lightweight, high-performance framework well-suited for deploying machine learning models.

To test the model, we created a Flask API and added an endpoint which accepts credit risk data via PostMan.

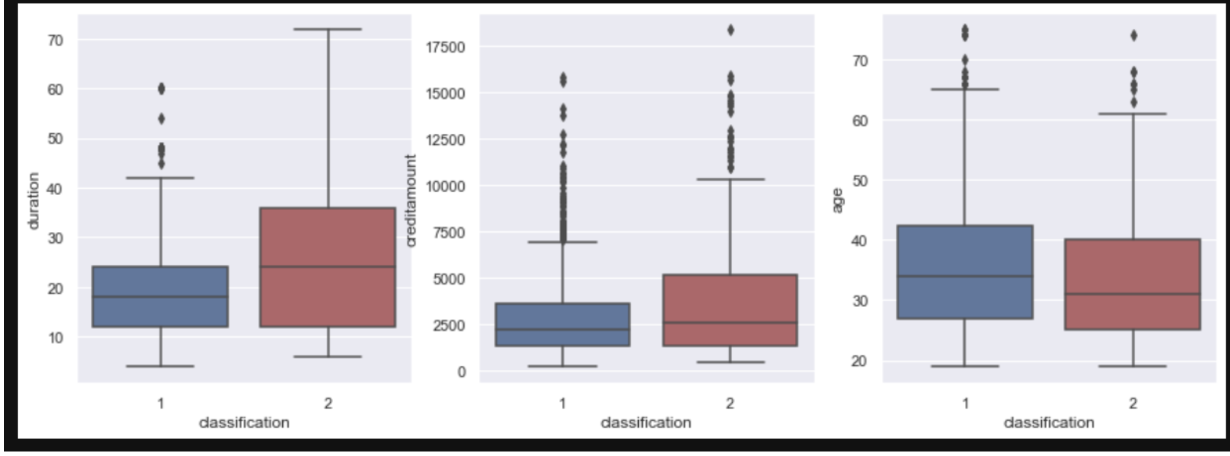
CHAPTER 6

METRICS

Boxplot on 3 numerical columns to understand the data distribution and revealing the presence of outliers

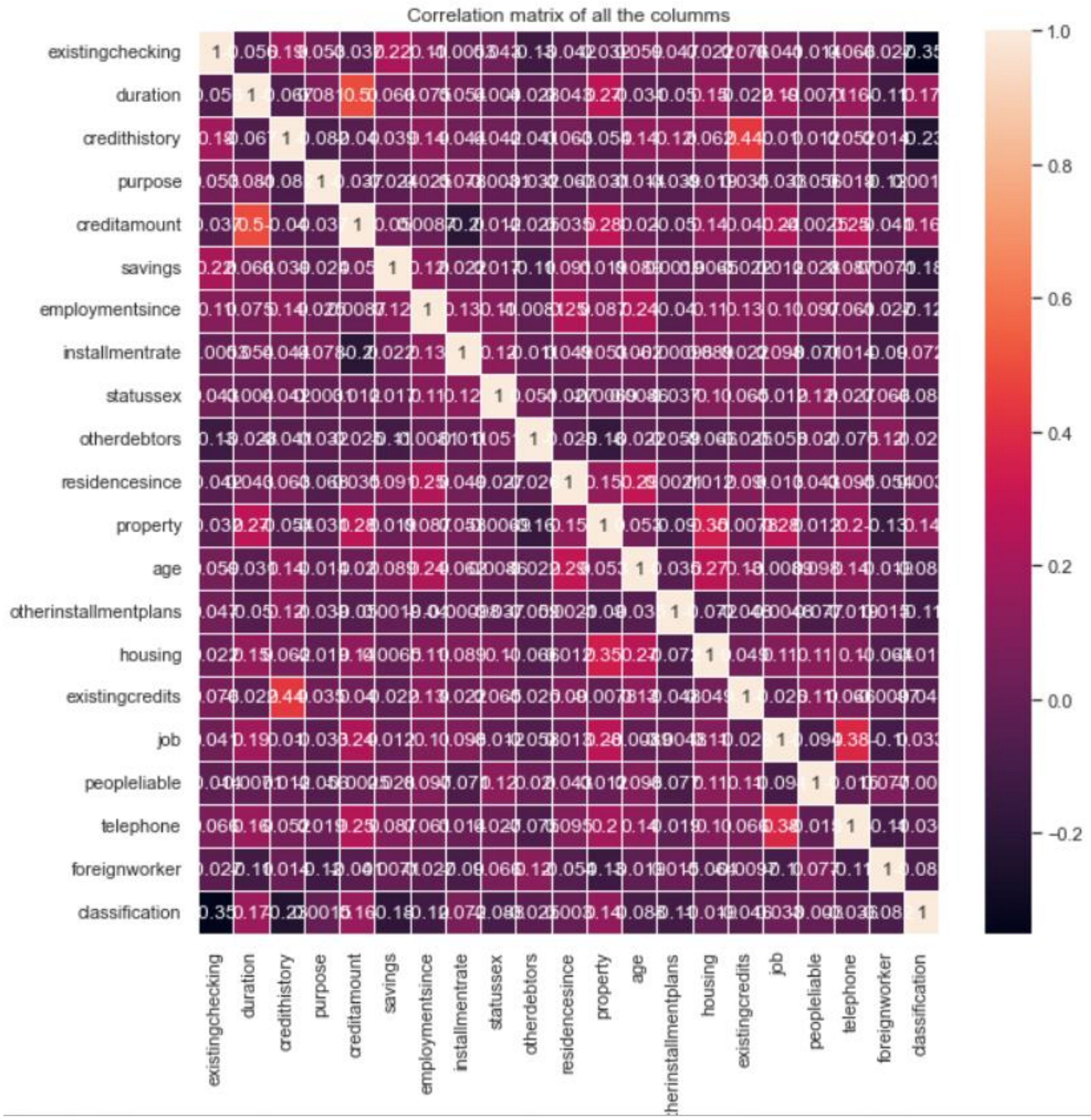


```
sns.set()
f, axes = plt.subplots(1, 3,figsize=(15,5))
sns.boxplot(y=data_original["duration"],x=data_original["classification"],orient='v' , ax=axes[0],palette=["#5975A4","#B55D60"]
sns.boxplot(y=data_original["creditamount"],x=data_original["classification"], orient='v' , ax=axes[1],palette=["#5975A4","#B55
sns.boxplot(y=data_original["age"],x=data_original["classification"], orient='v' , ax=axes[2],palette=["#5975A4","#B55D60"]) #b
plt.show()
```



Correlation matrix of all the columns after performing Label Encoding on categorical features and Binning on numerical features

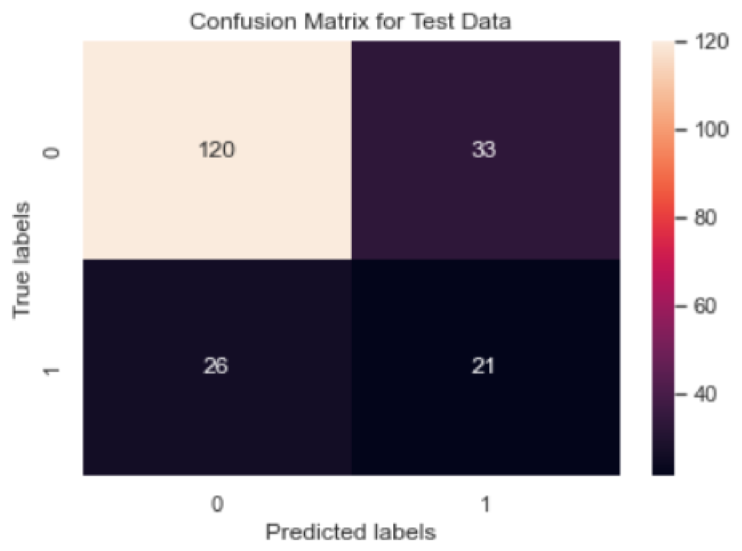
```
corr = data.corr()

fig, ax = plt.subplots(figsize=(10,10))
ax = sns.heatmap(corr, annot=True, linewidth=0.5, xticklabels=data.columns)
ax.set_yticklabels(labels=data.columns, rotation=0)
ax.set_title("Correlation matrix of all the columms")
plt.show()
```



Classification report of Simple Encoder without Encryption:

We evaluated the model's performance using classification_report. classification_report is a function in the scikit-learn library in Python that generates a comprehensive report summarizing the performance of a classification model. This report includes various metrics such as precision, recall, F1-score, and support for each class present in the target variable. It is particularly useful for evaluating the performance of a classification algorithm across multiple classes.

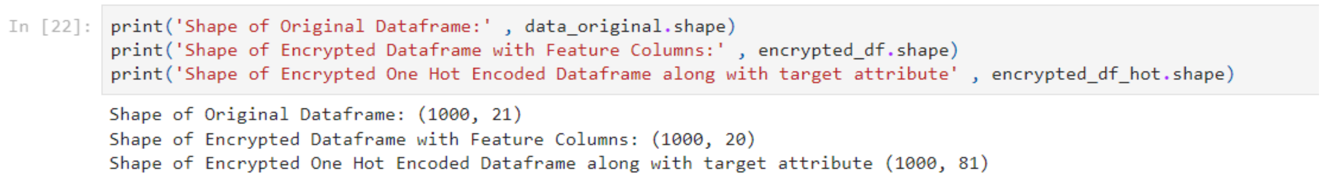
```
[Text(0, 0.5, '0'), Text(0, 1.5, '1')]
```



```
print(classification_report(y_test, y_pred))
```

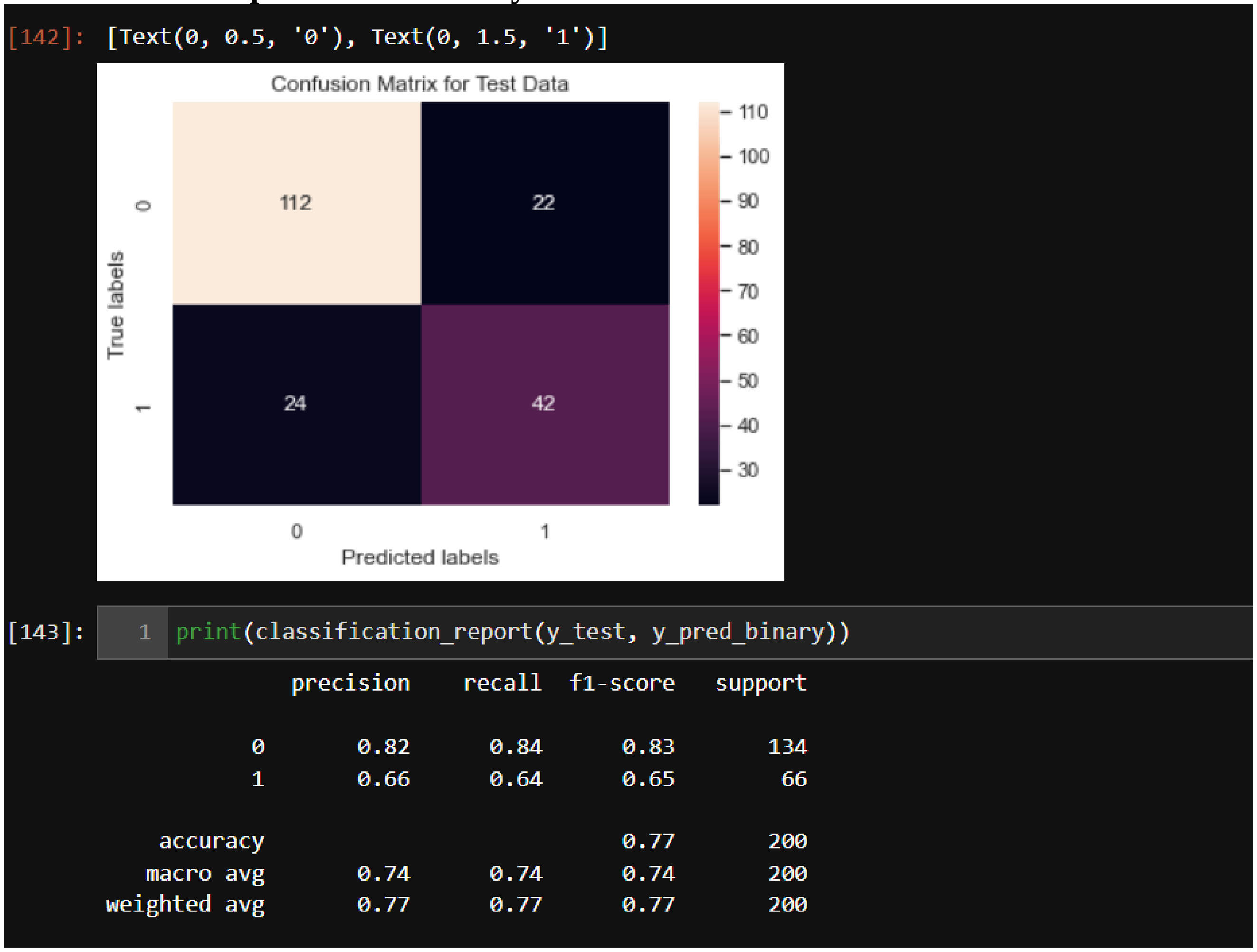
```
              precision    recall  f1-score   support

           0       0.82      0.78      0.80       153
           1       0.39      0.45      0.42        47

    accuracy                           0.70       200
   macro avg       0.61      0.62      0.61       200
weighted avg       0.72      0.70      0.71       200
```

Below screenshot show the transformation of data shape before and after encryption



```
In [22]: print('Shape of Original Dataframe:' , data_original.shape)
         print('Shape of Encrypted Dataframe with Feature Columns:' , encrypted_df.shape)
         print('Shape of Encrypted One Hot Encoded Dataframe along with target attribute' , encrypted_df_hot.shape)

         Shape of Original Dataframe: (1000, 21)
         Shape of Encrypted Dataframe with Feature Columns: (1000, 20)
         Shape of Encrypted One Hot Encoded Dataframe along with target attribute (1000, 81)
```

Classification report of our BinaryClassifier



```
[142]:  [Text(0, 0.5, '0'), Text(0, 1.5, '1')]
```

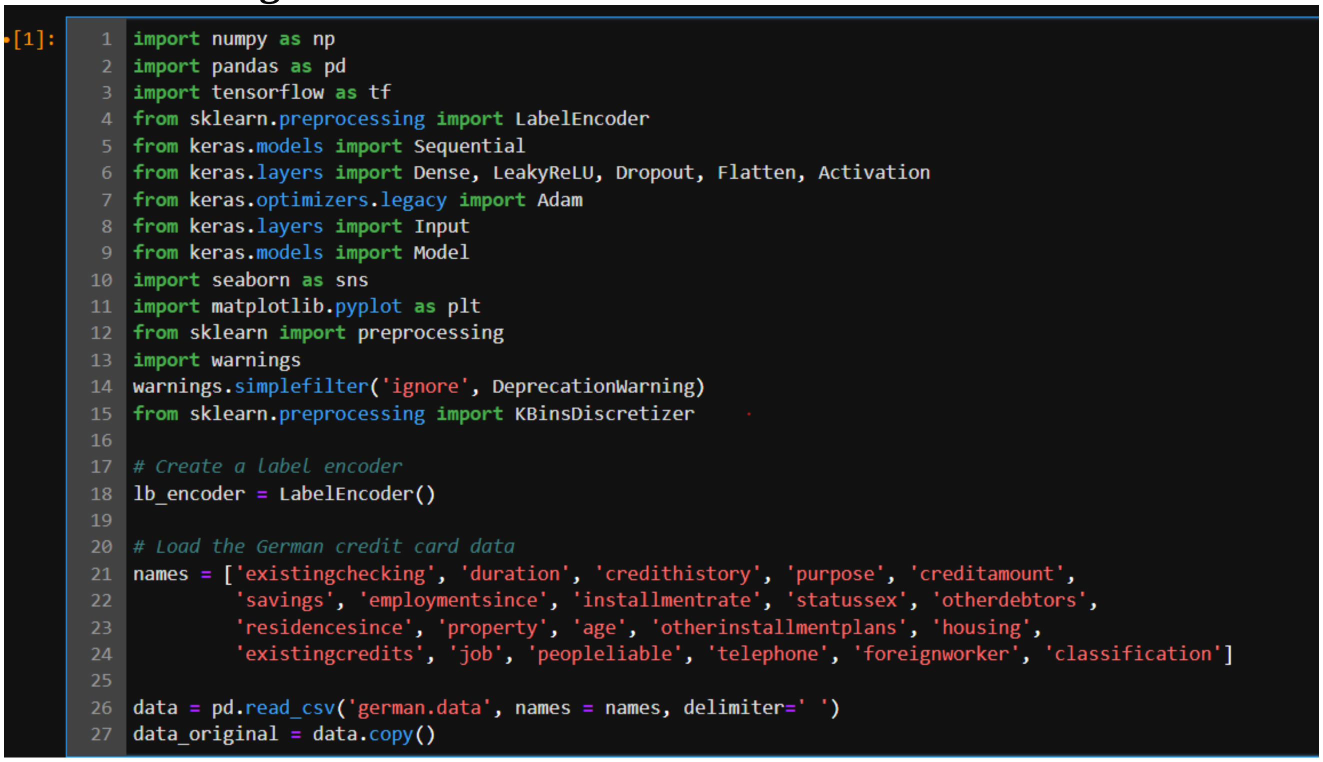
```
[143]:  print(classification_report(y_test, y_pred_binary))
```

```
              precision    recall  f1-score   support

           0       0.82      0.84      0.83       134
           1       0.66      0.64      0.65        66

    accuracy                           0.77       200
   macro avg       0.74      0.74      0.74       200
weighted avg       0.77      0.77      0.77       200
```

CHAPTER 7
CODE AND SCREENSHOTS

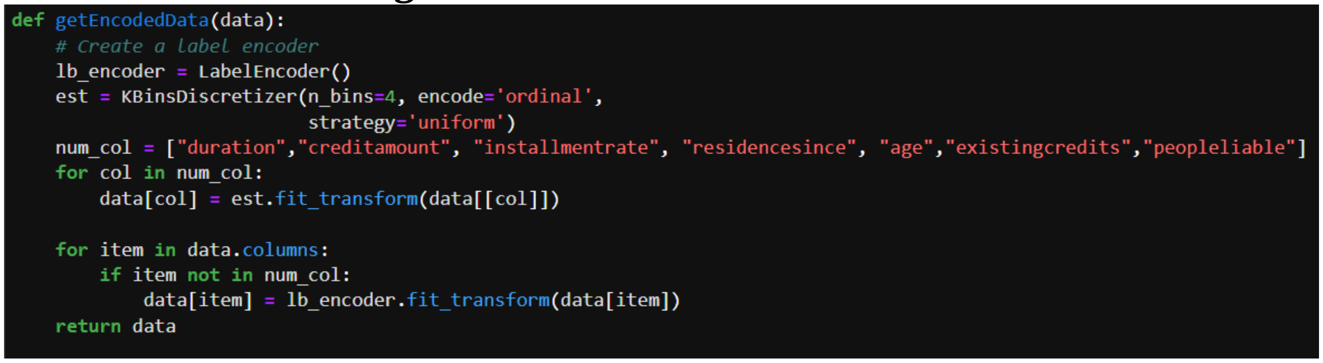
Below are the screenshots of the code and executions of the implementation.

1. Loading the data

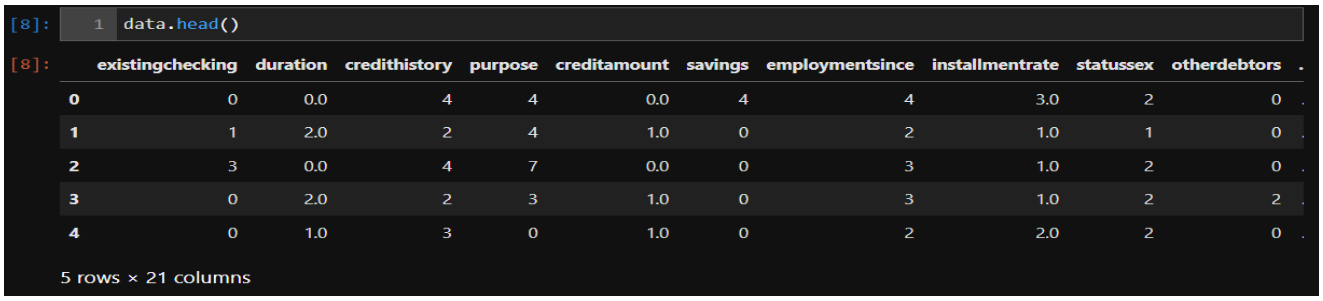


```
[1]:
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.preprocessing import LabelEncoder
from keras.models import Sequential
from keras.layers import Dense, LeakyReLU, Dropout, Flatten, Activation
from keras.optimizers.legacy import Adam
from keras.layers import Input
from keras.models import Model
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import preprocessing
import warnings
warnings.simplefilter('ignore', DeprecationWarning)
from sklearn.preprocessing import KBinsDiscretizer

# Create a label encoder
lb_encoder = LabelEncoder()

# Load the German credit card data
names = ['existingchecking', 'duration', 'credithistory', 'purpose', 'creditamount',
         'savings', 'employmentsince', 'installmentrate', 'statussex', 'otherdebtors',
         'residencesince', 'property', 'age', 'otherinstallmentplans', 'housing',
         'existingcredits', 'job', 'peopleliable', 'telephone', 'foreignworker', 'classification']

data = pd.read_csv('german.data', names = names, delimiter=' ')
data_original = data.copy()
```

2. Label Encoding and Discretization

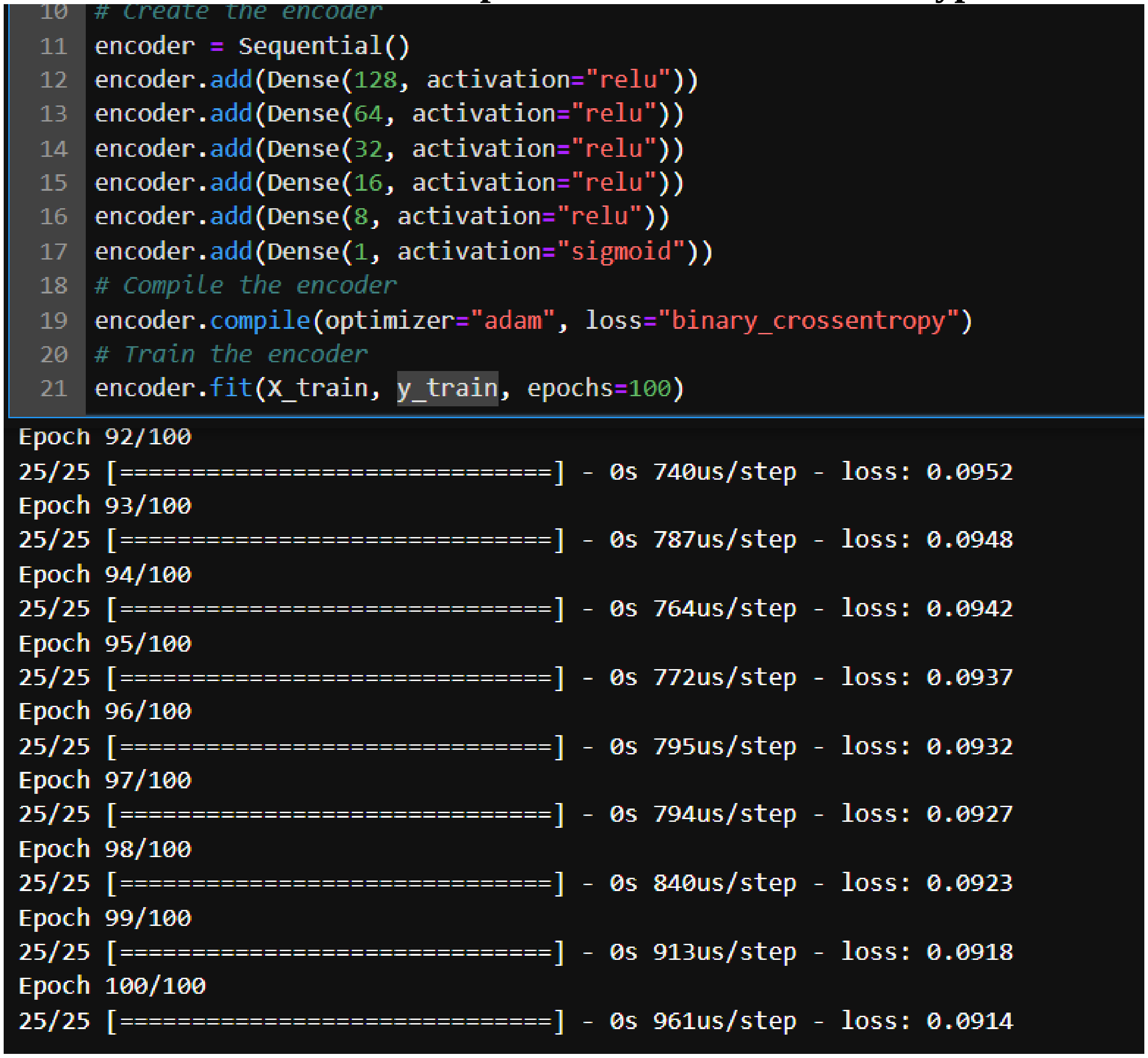


```
def getEncodedData(data):
    # Create a label encoder
    lb_encoder = LabelEncoder()
    est = KBinsDiscretizer(n_bins=4, encode='ordinal',
                           strategy='uniform')
    num_col = ["duration","creditamount", "installmentrate", "residencesince", "age","existingcredits","peopleliable"]
    for col in num_col:
        data[col] = est.fit_transform(data[[col]])

    for item in data.columns:
        if item not in num_col:
            data[item] = lb_encoder.fit_transform(data[item])
    return data
```



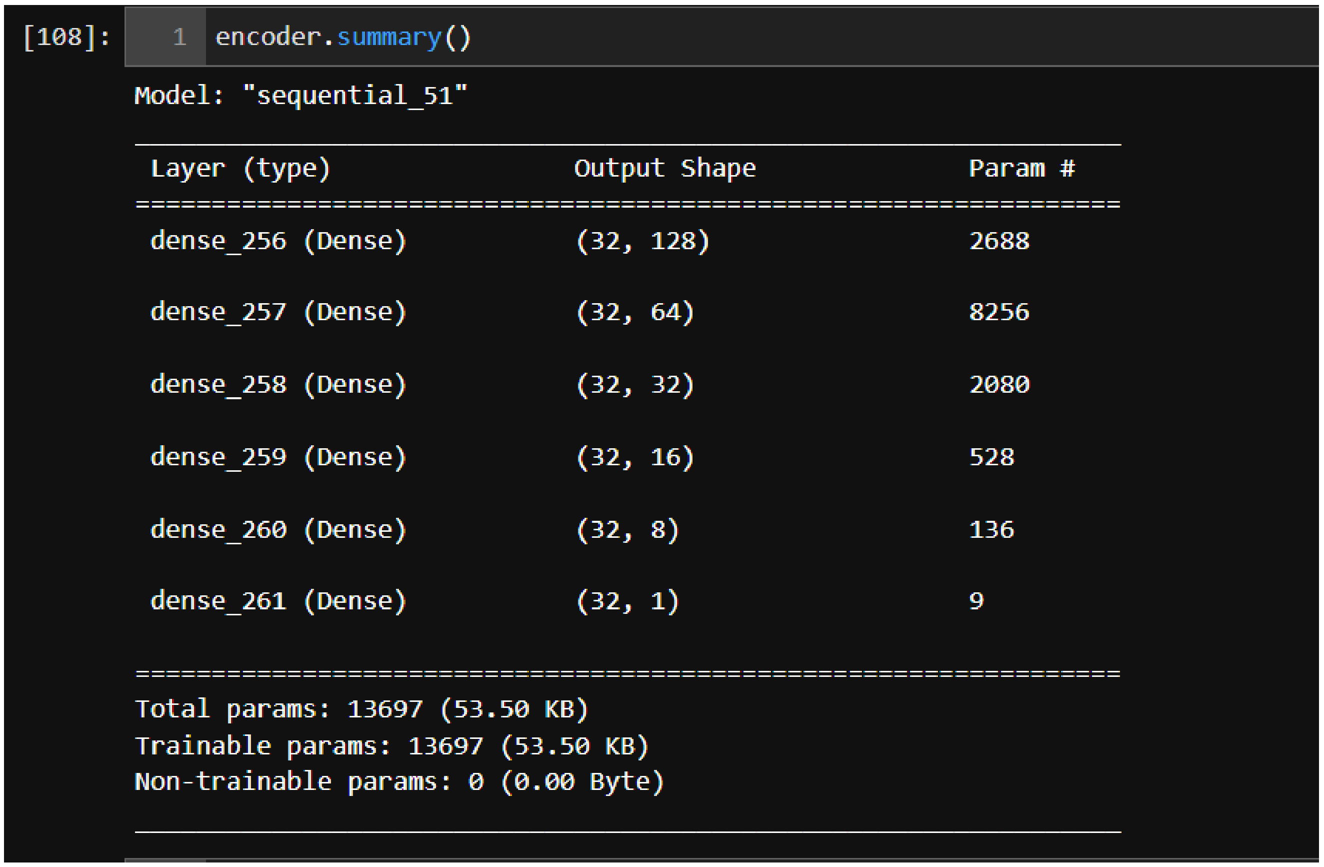
```
[8]: data.head()

[8]:
   existingchecking  duration  credithistory  purpose  creditamount  savings  employmentsince  installmentrate  statussex  otherdebtors  ...
0                 0       0.0              4        4           0.0        4                4              3.0          2             0  ...
1                 1       2.0              2        4           1.0        0                2              1.0          1             0  ...
2                 3       0.0              4        7           0.0        0                3              1.0          2             0  ...
3                 0       2.0              2        3           1.0        0                3              1.0          2             2  ...
4                 0       1.0              3        0           1.0        0                2              2.0          2             0  ...

5 rows × 21 columns
```

3. NN Model trained on plain data which is not encrypted



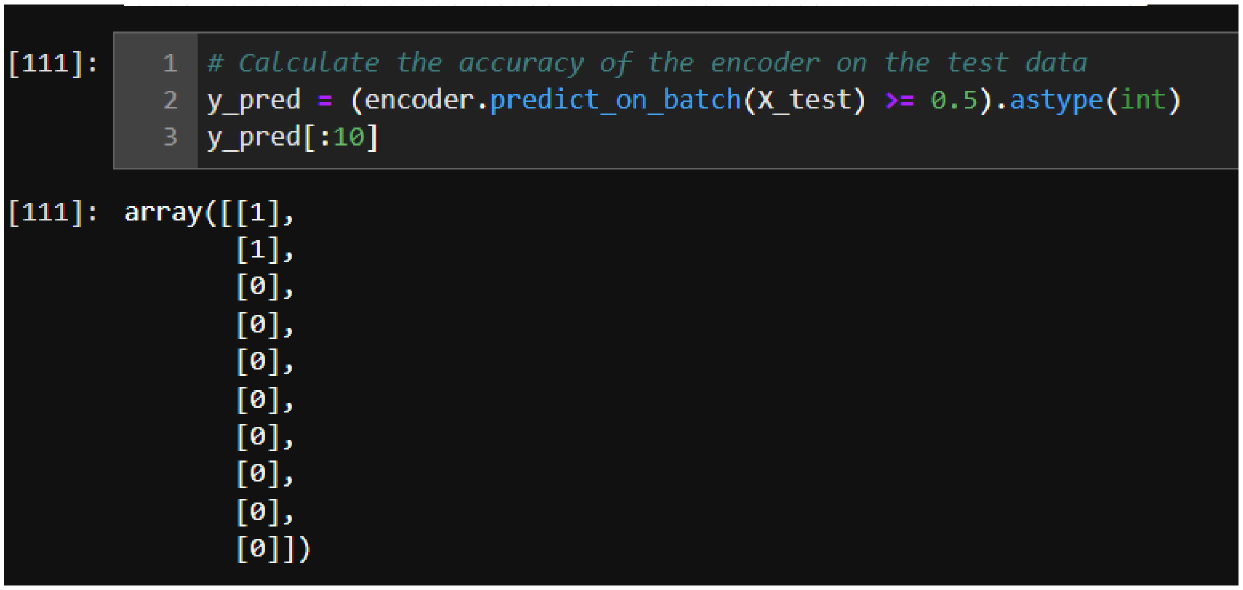
```
# Create the encoder
encoder = Sequential()
encoder.add(Dense(128, activation="relu"))
encoder.add(Dense(64, activation="relu"))
encoder.add(Dense(32, activation="relu"))
encoder.add(Dense(16, activation="relu"))
encoder.add(Dense(8, activation="relu"))
encoder.add(Dense(1, activation="sigmoid"))
# Compile the encoder
encoder.compile(optimizer="adam", loss="binary_crossentropy")
# Train the encoder
encoder.fit(X_train, y_train, epochs=100)
```

```
Epoch 92/100
25/25 [==============================] - 0s 740us/step - loss: 0.0952
Epoch 93/100
25/25 [==============================] - 0s 787us/step - loss: 0.0948
Epoch 94/100
25/25 [==============================] - 0s 764us/step - loss: 0.0942
Epoch 95/100
25/25 [==============================] - 0s 772us/step - loss: 0.0937
Epoch 96/100
25/25 [==============================] - 0s 795us/step - loss: 0.0932
Epoch 97/100
25/25 [==============================] - 0s 794us/step - loss: 0.0927
Epoch 98/100
25/25 [==============================] - 0s 840us/step - loss: 0.0923
Epoch 99/100
25/25 [==============================] - 0s 913us/step - loss: 0.0918
Epoch 100/100
25/25 [==============================] - 0s 961us/step - loss: 0.0914
```



```
[108]: encoder.summary()
```

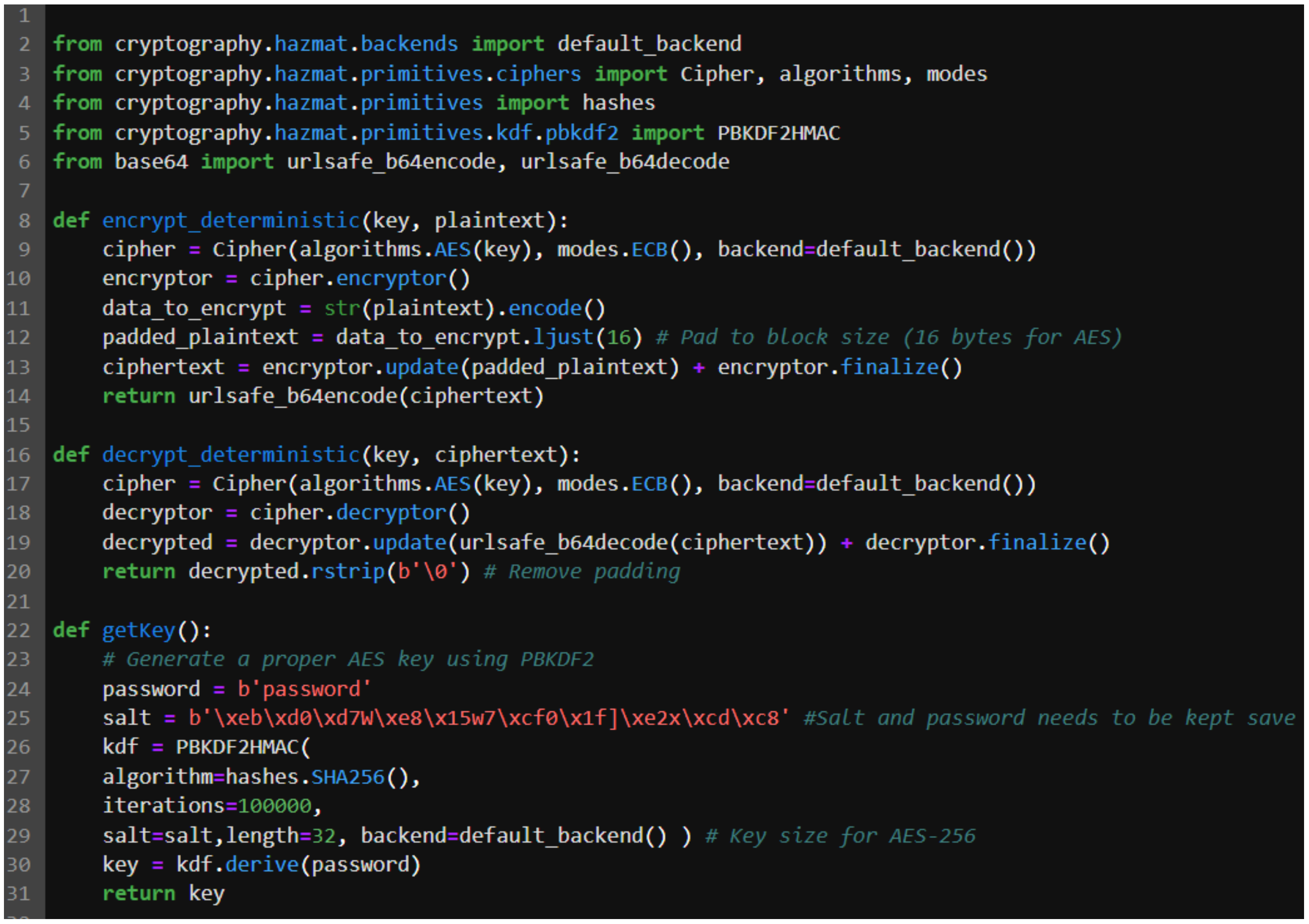
```
Model: "sequential_51"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_256 (Dense)           (32, 128)                 2688

 dense_257 (Dense)           (32, 64)                  8256

 dense_258 (Dense)           (32, 32)                  2080

 dense_259 (Dense)           (32, 16)                  528

 dense_260 (Dense)           (32, 8)                   136

 dense_261 (Dense)           (32, 1)                   9

=================================================================
Total params: 13697 (53.50 KB)
Trainable params: 13697 (53.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```



```
[111]: # Calculate the accuracy of the encoder on the test data
       y_pred = (encoder.predict_on_batch(X_test) >= 0.5).astype(int)
       y_pred[:10]
```

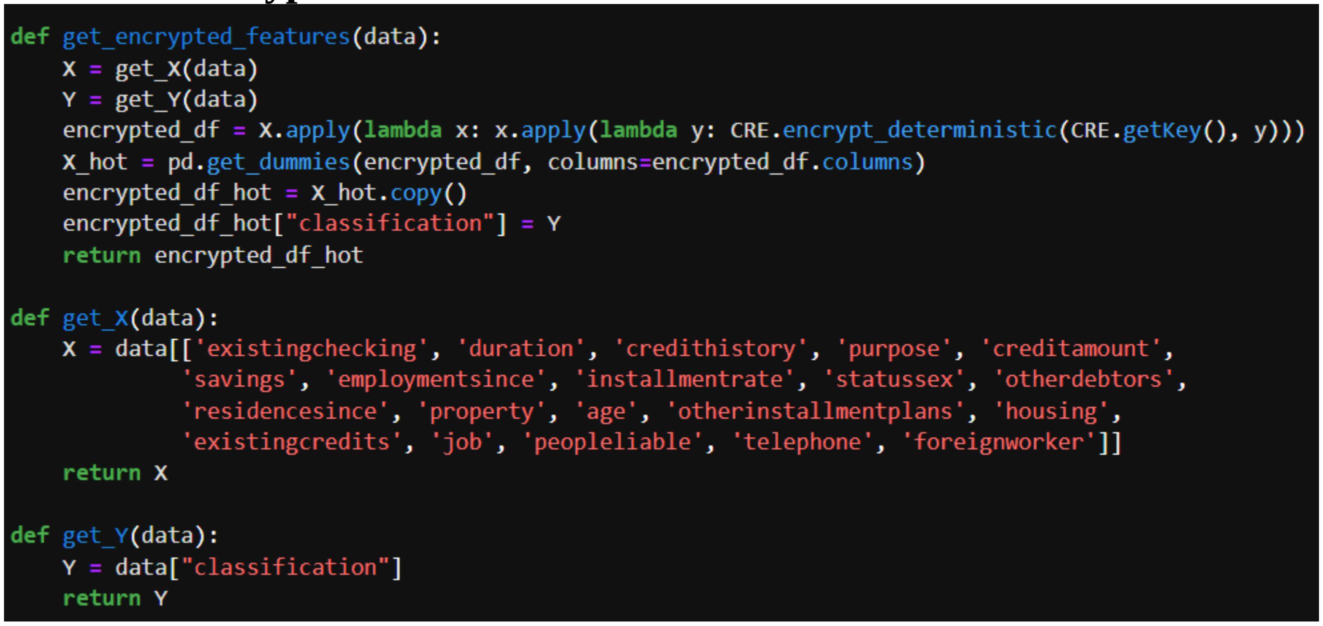
```
[111]: array([[1],
              [1],
              [0],
              [0],
              [0],
              [0],
              [0],
              [0],
              [0],
              [0]])
```

4. Client End Data Encryption Method



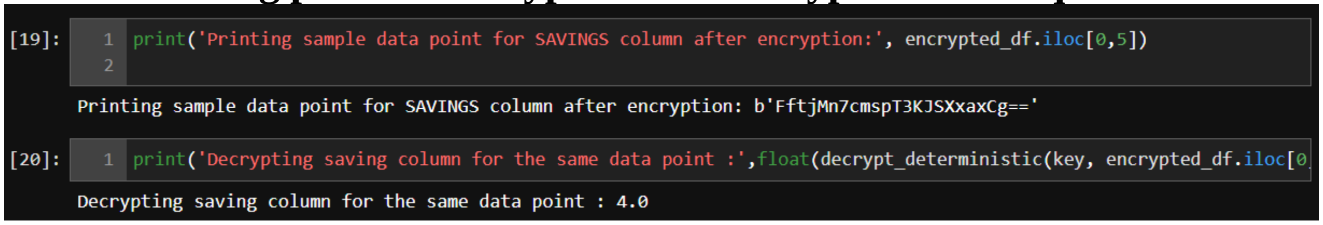
```python

from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
from base64 import urlsafe_b64encode, urlsafe_b64decode

def encrypt_deterministic(key, plaintext):
    cipher = Cipher(algorithms.AES(key), modes.ECB(), backend=default_backend())
    encryptor = cipher.encryptor()
    data_to_encrypt = str(plaintext).encode()
    padded_plaintext = data_to_encrypt.ljust(16) # Pad to block size (16 bytes for AES)
    ciphertext = encryptor.update(padded_plaintext) + encryptor.finalize()
    return urlsafe_b64encode(ciphertext)

def decrypt_deterministic(key, ciphertext):
    cipher = Cipher(algorithms.AES(key), modes.ECB(), backend=default_backend())
    decryptor = cipher.decryptor()
    decrypted = decryptor.update(urlsafe_b64decode(ciphertext)) + decryptor.finalize()
    return decrypted.rstrip(b'\0') # Remove padding

def getKey():
    # Generate a proper AES key using PBKDF2
    password = b'password'
    salt = b'\xeb\xd0\xd7W\xe8\x15w7\xcf0\x1f]\xe2x\xcd\xc8' #Salt and password needs to be kept save
    kdf = PBKDF2HMAC(
    algorithm=hashes.SHA256(),
    iterations=100000,
    salt=salt,length=32, backend=default_backend() ) # Key size for AES-256
    key = kdf.derive(password)
    return key
```

5. Code to Encrypt Encoded data and performing One-Hot encoding on encrypted data



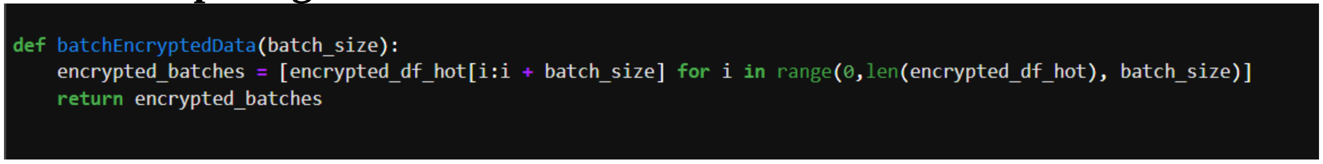
```python
def get_encrypted_features(data):
    X = get_X(data)
    Y = get_Y(data)
    encrypted_df = X.apply(lambda x: x.apply(lambda y: CRE.encrypt_deterministic(CRE.getKey(), y)))
    X_hot = pd.get_dummies(encrypted_df, columns=encrypted_df.columns)
    encrypted_df_hot = X_hot.copy()
    encrypted_df_hot["classification"] = Y
    return encrypted_df_hot

def get_X(data):
    X = data[['existingchecking', 'duration', 'credithistory', 'purpose', 'creditamount',
             'savings', 'employmentsince', 'installmentrate', 'statussex', 'otherdebtors',
             'residencesince', 'property', 'age', 'otherinstallmentplans', 'housing',
             'existingcredits', 'job', 'peopleliable', 'telephone', 'foreignworker']]
    return X

def get_Y(data):
    Y = data["classification"]
    return Y
```

6. Printing proof of encryption and decryption technique



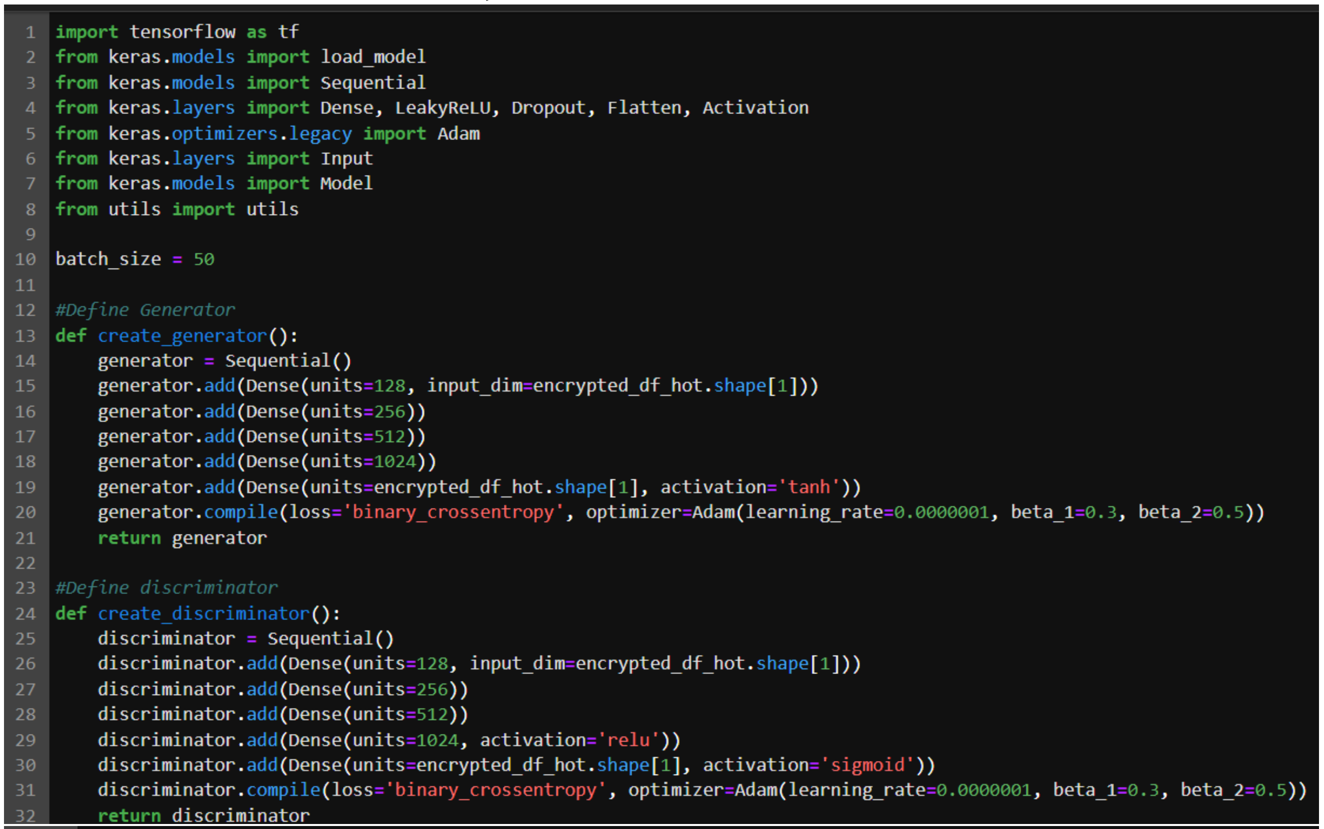
```
[19]: print('Printing sample data point for SAVINGS column after encryption:', encrypted_df.iloc[0,5])

      Printing sample data point for SAVINGS column after encryption: b'FftjMn7cmspT3KJSXxaxCg=='

[20]: print('Decrypting saving column for the same data point :',float(decrypt_deterministic(key, encrypted_df.iloc[0

      Decrypting saving column for the same data point : 4.0
```

7. Preparing batched data for GAN Model

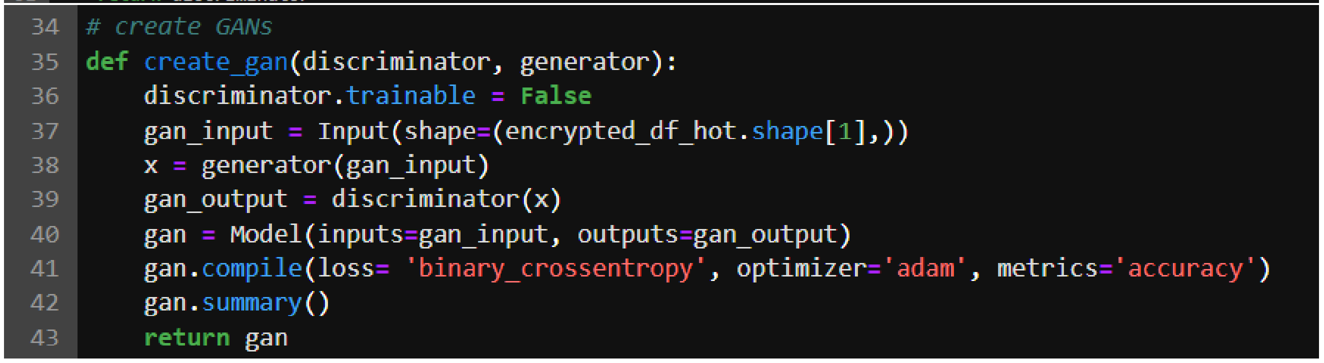


```python
def batchEncryptedData(batch_size):
    encrypted_batches = [encrypted_df_hot[i:i + batch_size] for i in range(0,len(encrypted_df_hot), batch_size)]
    return encrypted_batches
```

8. Code for Generator, Discriminator and GAN

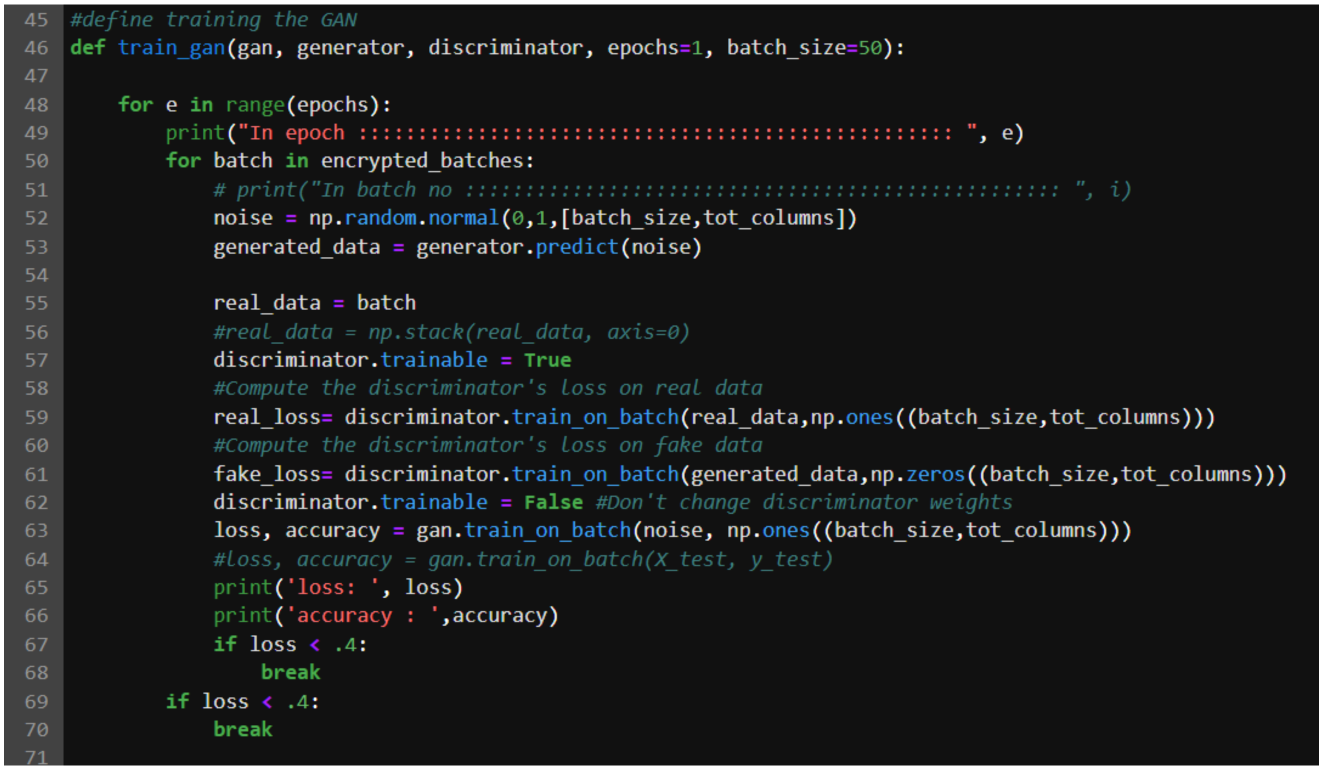


```python
import tensorflow as tf
from keras.models import load_model
from keras.models import Sequential
from keras.layers import Dense, LeakyReLU, Dropout, Flatten, Activation
from keras.optimizers.legacy import Adam
from keras.layers import Input
from keras.models import Model
from utils import utils

batch_size = 50

#Define Generator
def create_generator():
    generator = Sequential()
    generator.add(Dense(units=128, input_dim=encrypted_df_hot.shape[1]))
    generator.add(Dense(units=256))
    generator.add(Dense(units=512))
    generator.add(Dense(units=1024))
    generator.add(Dense(units=encrypted_df_hot.shape[1], activation='tanh'))
    generator.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.0000001, beta_1=0.3, beta_2=0.5))
    return generator

#Define discriminator
def create_discriminator():
    discriminator = Sequential()
    discriminator.add(Dense(units=128, input_dim=encrypted_df_hot.shape[1]))
    discriminator.add(Dense(units=256))
    discriminator.add(Dense(units=512))
    discriminator.add(Dense(units=1024, activation='relu'))
    discriminator.add(Dense(units=encrypted_df_hot.shape[1], activation='sigmoid'))
    discriminator.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.0000001, beta_1=0.3, beta_2=0.5))
    return discriminator
```



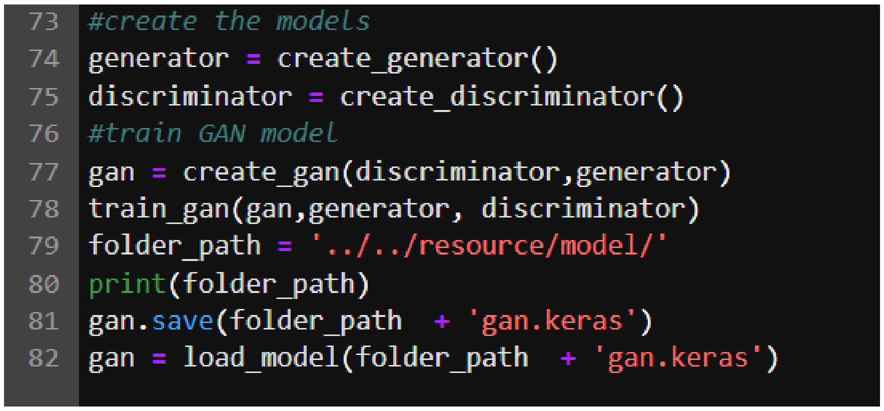
```python
# create GANs
def create_gan(discriminator, generator):
    discriminator.trainable = False
    gan_input = Input(shape=(encrypted_df_hot.shape[1],))
    x = generator(gan_input)
    gan_output = discriminator(x)
    gan = Model(inputs=gan_input, outputs=gan_output)
    gan.compile(loss= 'binary_crossentropy', optimizer='adam', metrics='accuracy')
    gan.summary()
    return gan
```

9. Training the GAN



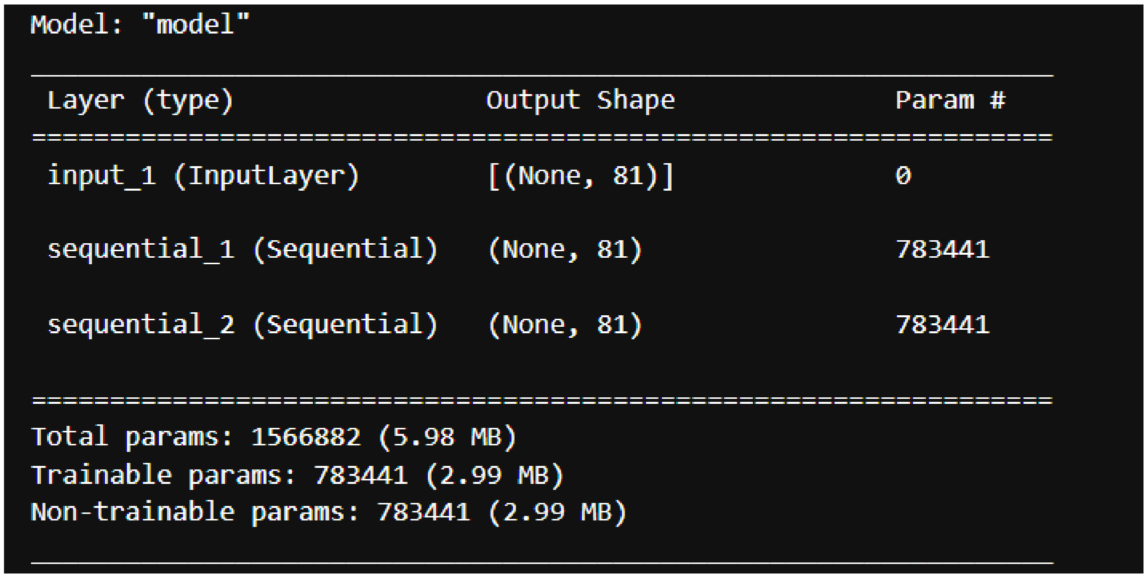
```python
#define training the GAN
def train_gan(gan, generator, discriminator, epochs=1, batch_size=50):

    for e in range(epochs):
        print("In epoch :::::::::::::::::::::::::::::::::::::::::::::::::: ", e)
        for batch in encrypted_batches:
            # print("In batch no :::::::::::::::::::::::::::::::::::::::::::::::::: ", i)
            noise = np.random.normal(0,1,[batch_size,tot_columns])
            generated_data = generator.predict(noise)

            real_data = batch
            #real_data = np.stack(real_data, axis=0)
            discriminator.trainable = True
            #Compute the discriminator's loss on real data
            real_loss= discriminator.train_on_batch(real_data,np.ones((batch_size,tot_columns)))
            #Compute the discriminator's loss on fake data
            fake_loss= discriminator.train_on_batch(generated_data,np.zeros((batch_size,tot_columns)))
            discriminator.trainable = False #Don't change discriminator weights
            loss, accuracy = gan.train_on_batch(noise, np.ones((batch_size,tot_columns)))
            #loss, accuracy = gan.train_on_batch(X_test, y_test)
            print('loss: ', loss)
            print('accuracy : ',accuracy)
            if loss < .4:
                break
        if loss < .4:
            break

```

10. Creating the Model and saving under resources

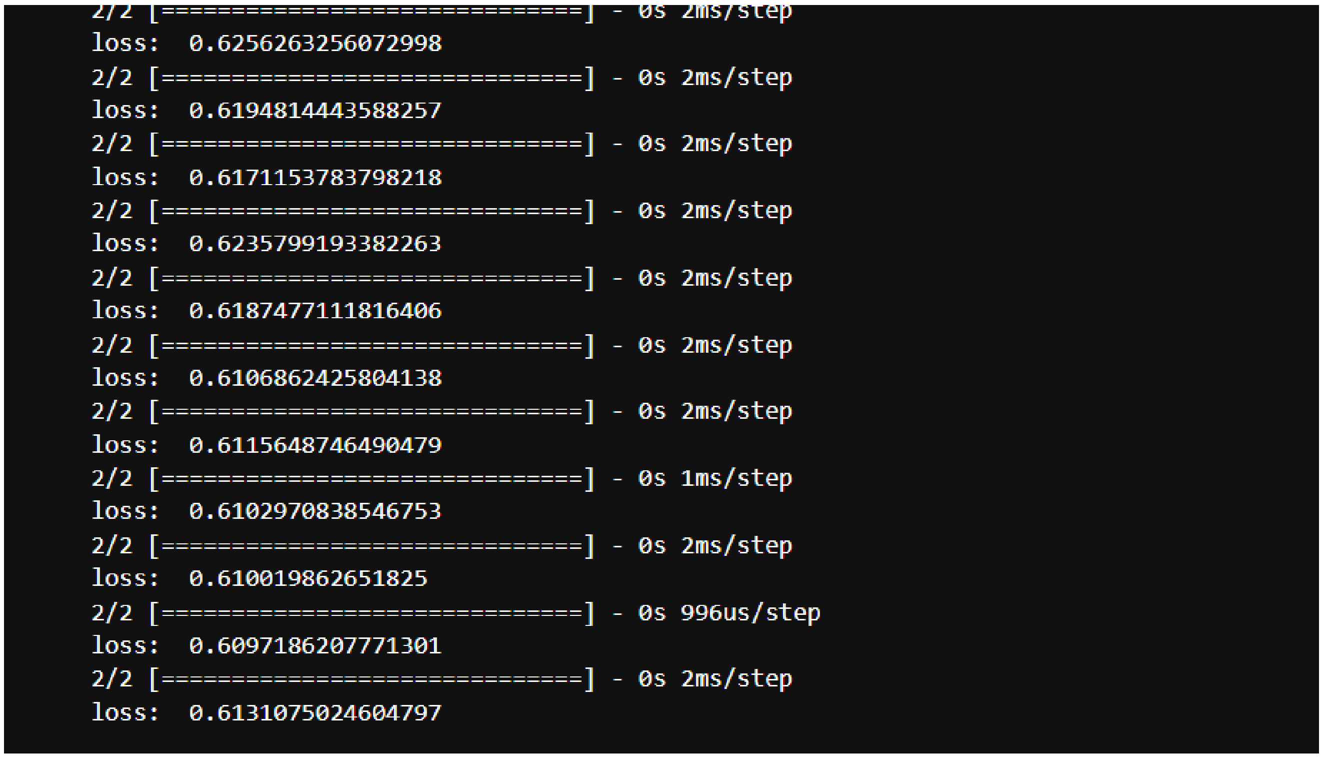


```python
#create the models
generator = create_generator()
discriminator = create_discriminator()
#train GAN model
gan = create_gan(discriminator,generator)
train_gan(gan,generator, discriminator)
folder_path = '../../resource/model/'
print(folder_path)
gan.save(folder_path  + 'gan.keras')
gan = load_model(folder_path  + 'gan.keras')
```

11. Printing Summary and loss

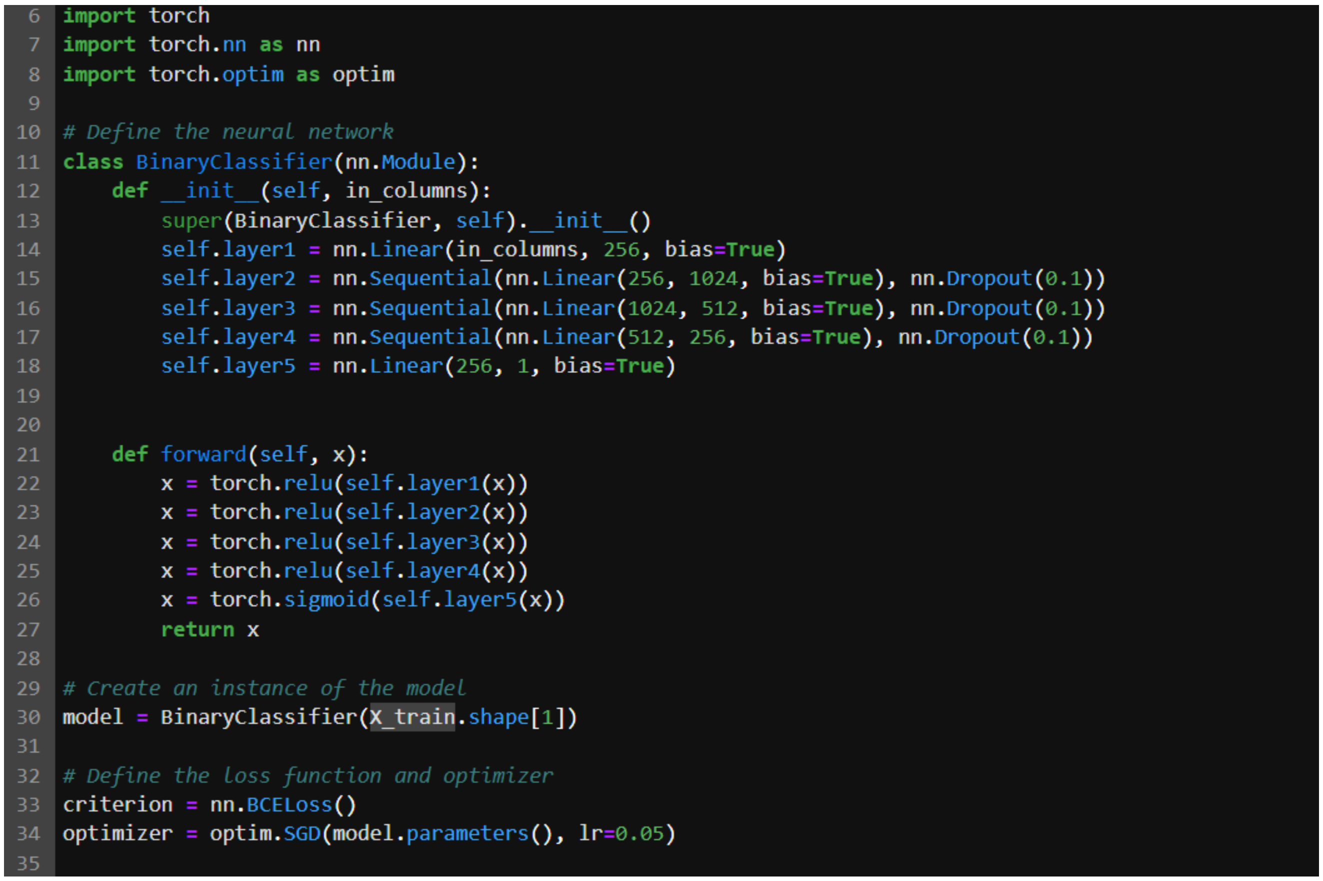


```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 81)]              0

 sequential_1 (Sequential)   (None, 81)                783441

 sequential_2 (Sequential)   (None, 81)                783441

=================================================================
Total params: 1566882 (5.98 MB)
Trainable params: 783441 (2.99 MB)
Non-trainable params: 783441 (2.99 MB)
_________________________________________________________________
```

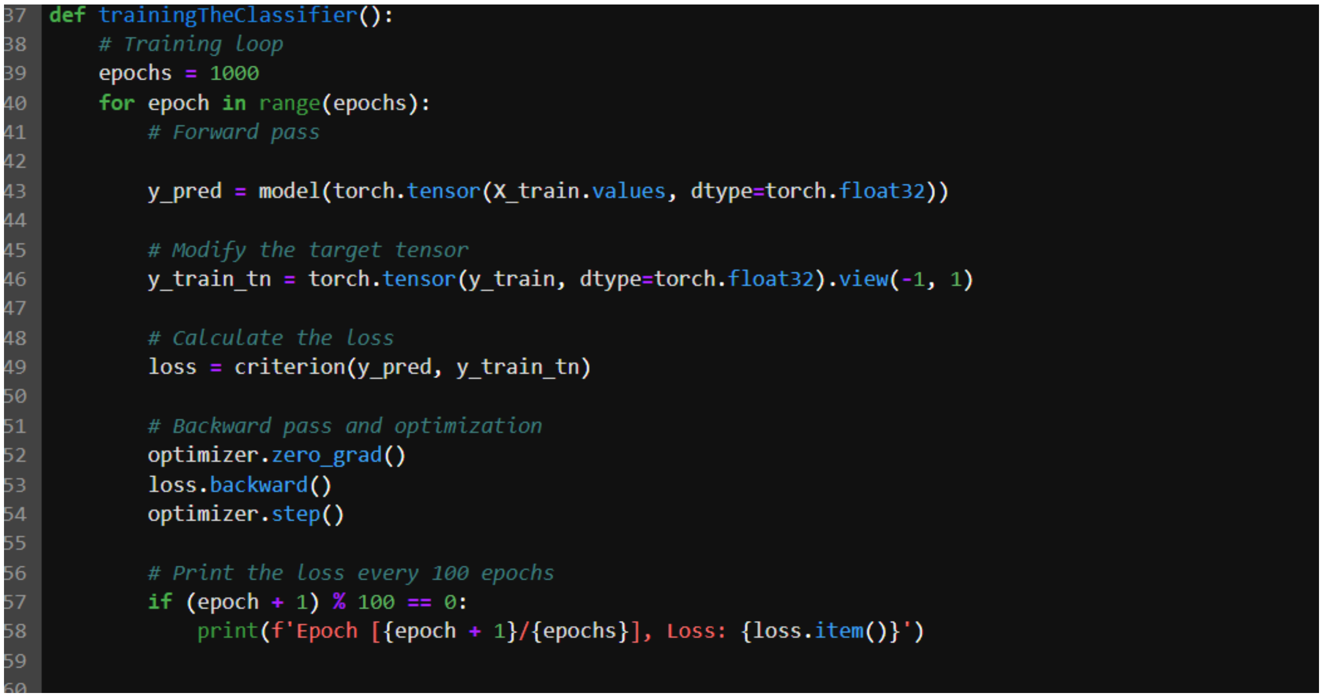


```
    2/2 [==============================] - 0s 2ms/step
    loss:  0.6256263256072998
    2/2 [==============================] - 0s 2ms/step
    loss:  0.6194814443588257
    2/2 [==============================] - 0s 2ms/step
    loss:  0.6171153783798218
    2/2 [==============================] - 0s 2ms/step
    loss:  0.6235799193382263
    2/2 [==============================] - 0s 2ms/step
    loss:  0.6187477111816406
    2/2 [==============================] - 0s 2ms/step
    loss:  0.6106862425804138
    2/2 [==============================] - 0s 2ms/step
    loss:  0.6115648746490479
    2/2 [==============================] - 0s 1ms/step
    loss:  0.6102970838546753
    2/2 [==============================] - 0s 2ms/step
    loss:  0.610019862651825
    2/2 [==============================] - 0s 996us/step
    loss:  0.6097186207771301
    2/2 [==============================] - 0s 2ms/step
    loss:  0.6131075024604797
```

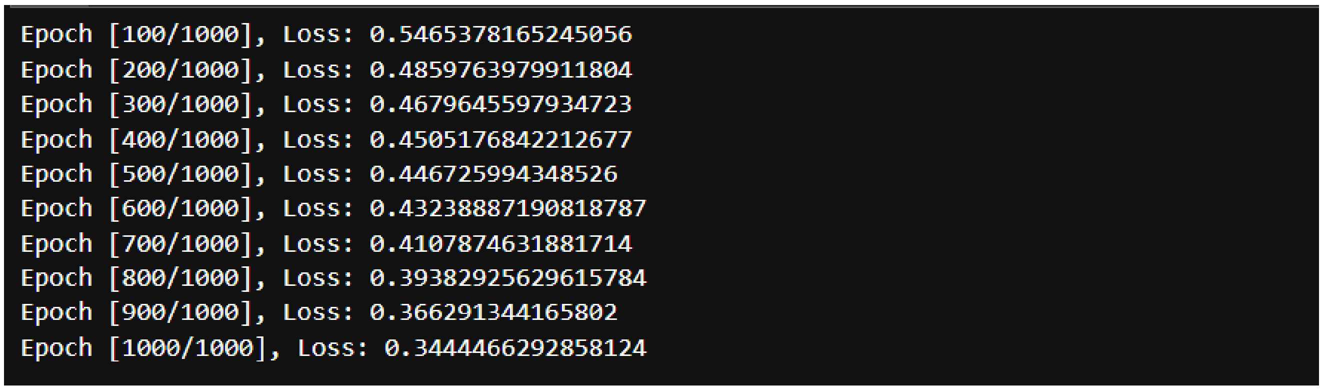
12. Generating Binary Classifier and Training



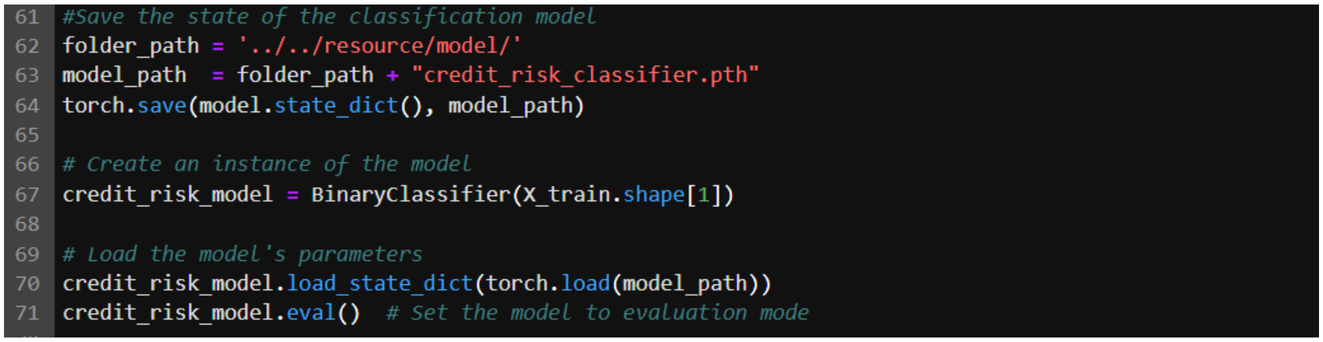
```python
import torch
import torch.nn as nn
import torch.optim as optim

# Define the neural network
class BinaryClassifier(nn.Module):
    def __init__(self, in_columns):
        super(BinaryClassifier, self).__init__()
        self.layer1 = nn.Linear(in_columns, 256, bias=True)
        self.layer2 = nn.Sequential(nn.Linear(256, 1024, bias=True), nn.Dropout(0.1))
        self.layer3 = nn.Sequential(nn.Linear(1024, 512, bias=True), nn.Dropout(0.1))
        self.layer4 = nn.Sequential(nn.Linear(512, 256, bias=True), nn.Dropout(0.1))
        self.layer5 = nn.Linear(256, 1, bias=True)


    def forward(self, x):
        x = torch.relu(self.layer1(x))
        x = torch.relu(self.layer2(x))
        x = torch.relu(self.layer3(x))
        x = torch.relu(self.layer4(x))
        x = torch.sigmoid(self.layer5(x))
        return x

# Create an instance of the model
model = BinaryClassifier(X_train.shape[1])

# Define the loss function and optimizer
criterion = nn.BCELoss()
optimizer = optim.SGD(model.parameters(), lr=0.05)

```

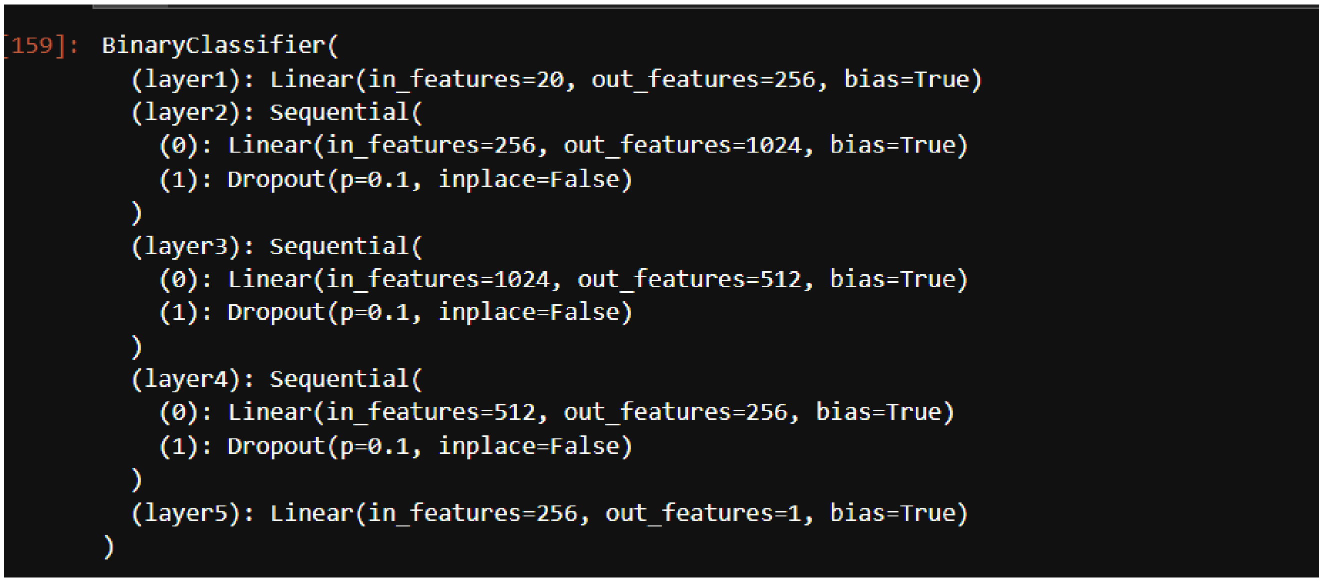


```python
def trainingTheClassifier():
    # Training loop
    epochs = 1000
    for epoch in range(epochs):
        # Forward pass

        y_pred = model(torch.tensor(X_train.values, dtype=torch.float32))

        # Modify the target tensor
        y_train_tn = torch.tensor(y_train, dtype=torch.float32).view(-1, 1)

        # Calculate the loss
        loss = criterion(y_pred, y_train_tn)

        # Backward pass and optimization
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # Print the loss every 100 epochs
        if (epoch + 1) % 100 == 0:
            print(f'Epoch [{epoch + 1}/{epochs}], Loss: {loss.item()}')

```

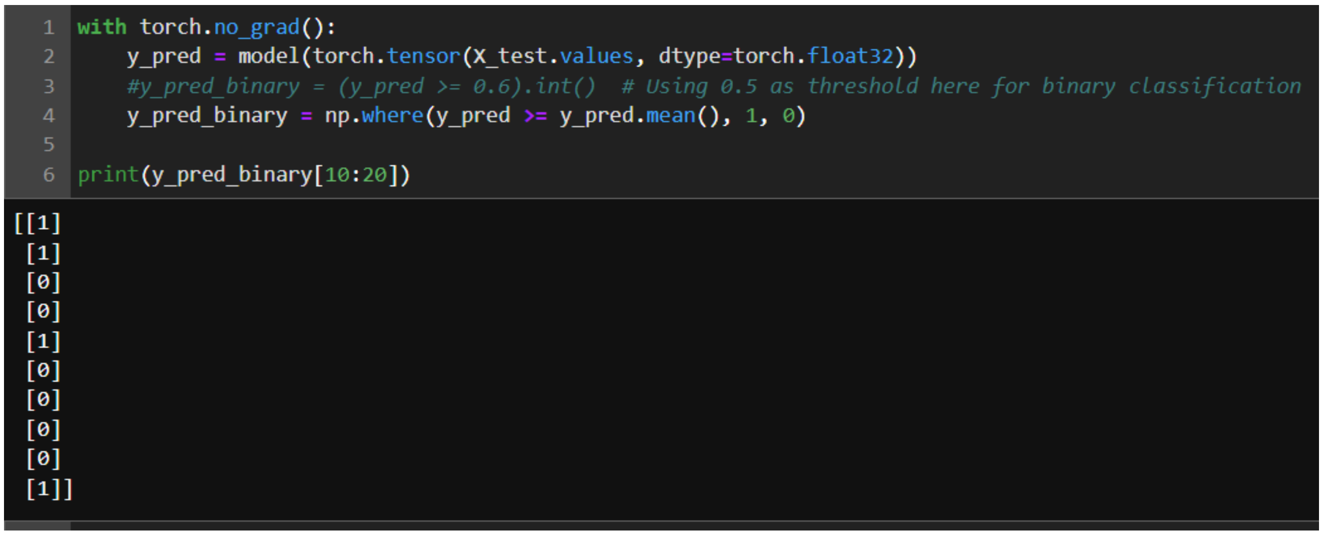

```
Epoch [100/1000], Loss: 0.5465378165245056
Epoch [200/1000], Loss: 0.4859763979911804
Epoch [300/1000], Loss: 0.4679645597934723
Epoch [400/1000], Loss: 0.4505176842212677
Epoch [500/1000], Loss: 0.446725994348526
Epoch [600/1000], Loss: 0.4323888719081878
Epoch [700/1000], Loss: 0.4107874631881714
Epoch [800/1000], Loss: 0.3938292562961578
Epoch [900/1000], Loss: 0.366291344165802
Epoch [1000/1000], Loss: 0.3444466292858124
```

13. Save the Binary Classifier Model


```python
#Save the state of the classification model
folder_path = '../../resource/model/'
model_path  = folder_path + "credit_risk_classifier.pth"
torch.save(model.state_dict(), model_path)

# Create an instance of the model
credit_risk_model = BinaryClassifier(X_train.shape[1])

# Load the model's parameters
credit_risk_model.load_state_dict(torch.load(model_path))
credit_risk_model.eval()  # Set the model to evaluation mode
```


```
[159]: BinaryClassifier(
         (layer1): Linear(in_features=20, out_features=256, bias=True)
         (layer2): Sequential(
           (0): Linear(in_features=256, out_features=1024, bias=True)
           (1): Dropout(p=0.1, inplace=False)
         )
         (layer3): Sequential(
           (0): Linear(in_features=1024, out_features=512, bias=True)
           (1): Dropout(p=0.1, inplace=False)
         )
         (layer4): Sequential(
           (0): Linear(in_features=512, out_features=256, bias=True)
           (1): Dropout(p=0.1, inplace=False)
         )
         (layer5): Linear(in_features=256, out_features=1, bias=True)
       )
```

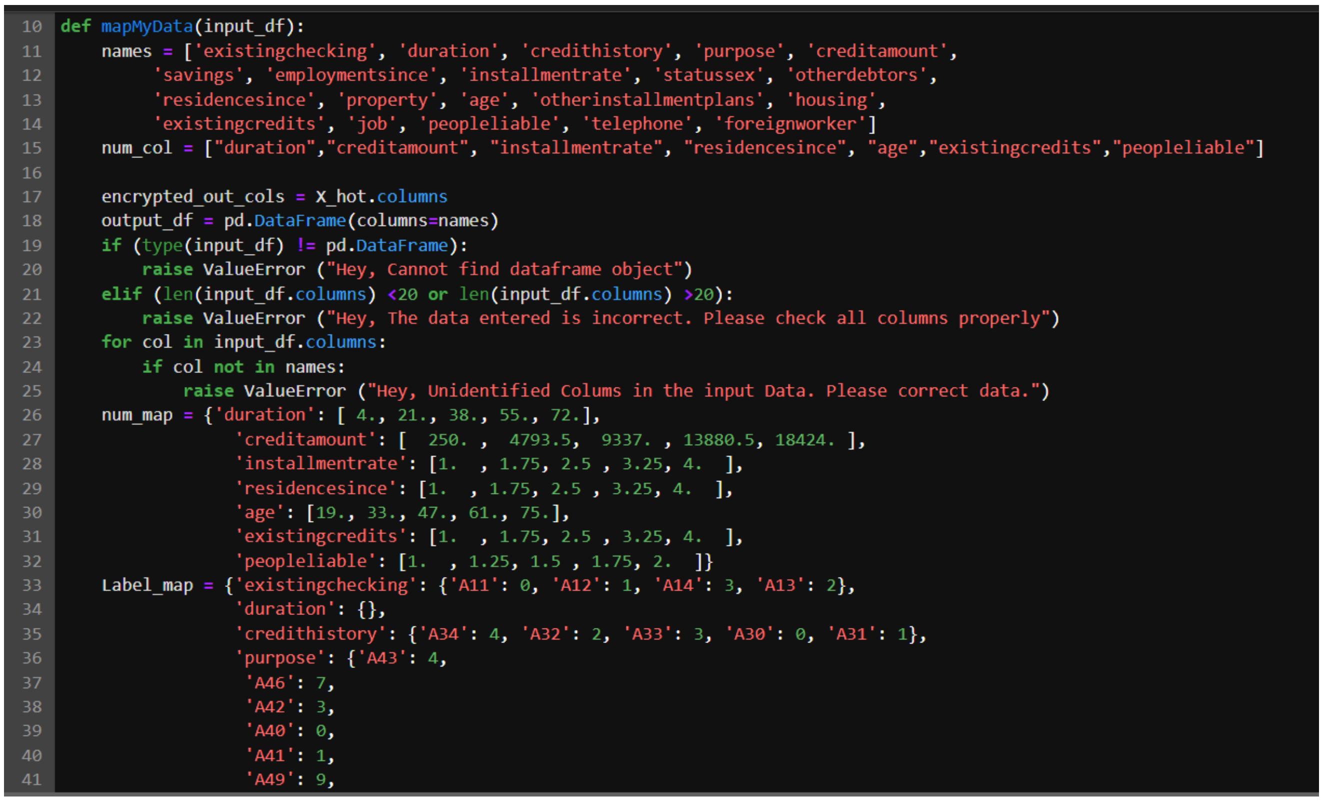
14. Testing Binary Classifier Model on test data



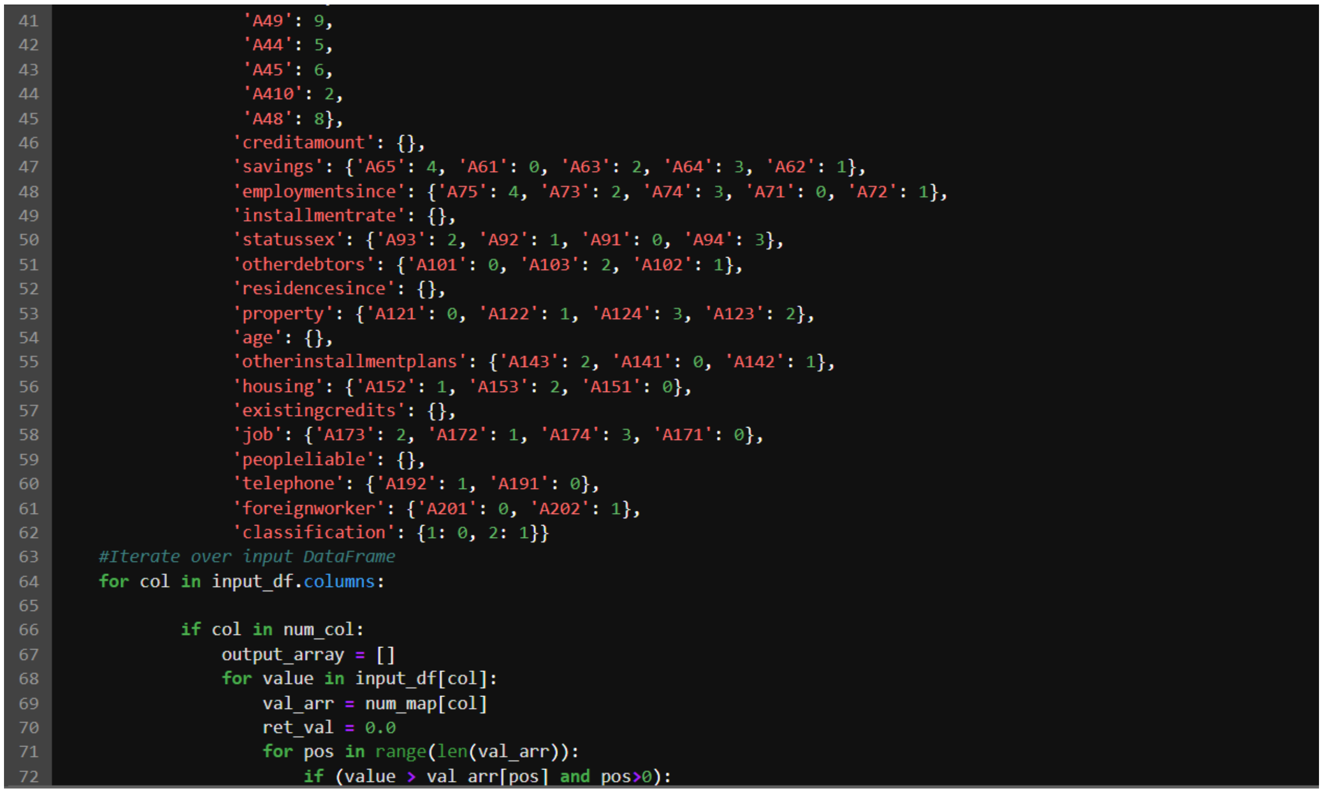
```python
with torch.no_grad():
    y_pred = model(torch.tensor(X_test.values, dtype=torch.float32))
    #y_pred_binary = (y_pred >= 0.6).int()  # Using 0.5 as threshold here for binary classification
    y_pred_binary = np.where(y_pred >= y_pred.mean(), 1, 0)

print(y_pred_binary[10:20])
```

```
[[1]
 [1]
 [0]
 [0]
 [1]
 [0]
 [0]
 [0]
 [0]
 [1]]
```

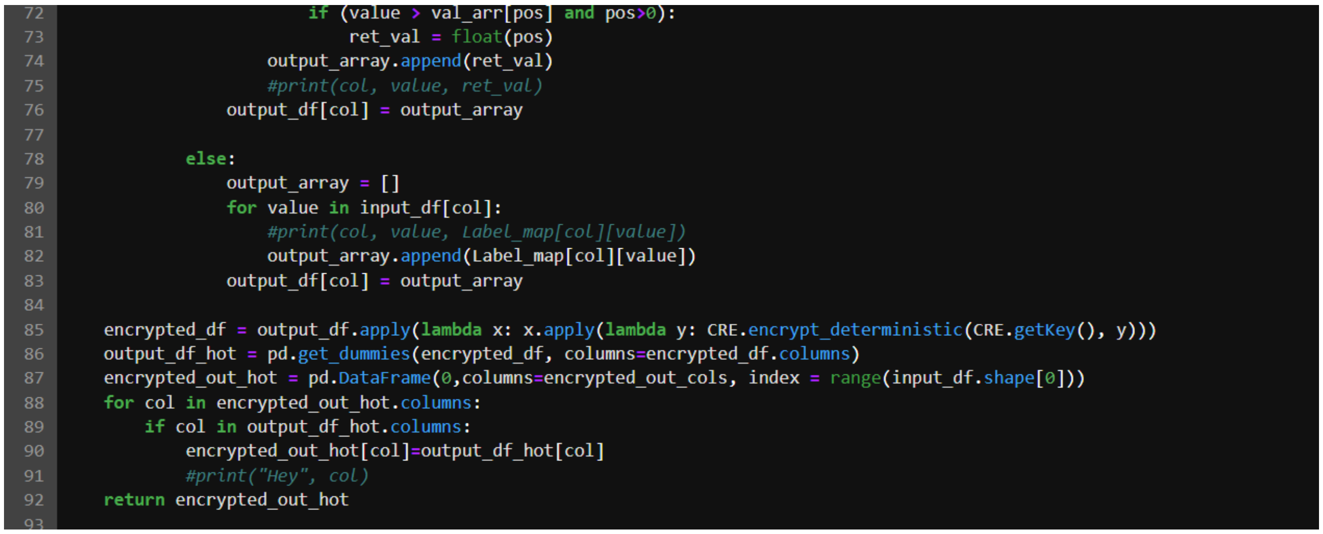
15. Function to convert input data from API to be consumed by model



```python
def mapMyData(input_df):
    names = ['existingchecking', 'duration', 'credithistory', 'purpose', 'creditamount',
         'savings', 'employmentsince', 'installmentrate', 'statussex', 'otherdebtors',
         'residencesince', 'property', 'age', 'otherinstallmentplans', 'housing',
         'existingcredits', 'job', 'peopleliable', 'telephone', 'foreignworker']
    num_col = ["duration","creditamount", "installmentrate", "residencesince", "age","existingcredits","peopleliable"]

    encrypted_out_cols = X_hot.columns
    output_df = pd.DataFrame(columns=names)
    if (type(input_df) != pd.DataFrame):
        raise ValueError ("Hey, Cannot find dataframe object")
    elif (len(input_df.columns) <20 or len(input_df.columns) >20):
        raise ValueError ("Hey, The data entered is incorrect. Please check all columns properly")
    for col in input_df.columns:
        if col not in names:
            raise ValueError ("Hey, Unidentified Colums in the input Data. Please correct data.")
    num_map = {'duration': [ 4., 21., 38., 55., 72.],
                 'creditamount': [  250. ,  4793.5,  9337. , 13880.5, 18424. ],
                 'installmentrate': [1.  , 1.75, 2.5 , 3.25, 4.  ],
                 'residencesince': [1.  , 1.75, 2.5 , 3.25, 4.  ],
                 'age': [19., 33., 47., 61., 75.],
                 'existingcredits': [1.  , 1.75, 2.5 , 3.25, 4.  ],
                 'peopleliable': [1.  , 1.25, 1.5 , 1.75, 2.  ]}
    Label_map = {'existingchecking': {'A11': 0, 'A12': 1, 'A14': 3, 'A13': 2},
                 'duration': {},
                 'credithistory': {'A34': 4, 'A32': 2, 'A33': 3, 'A30': 0, 'A31': 1},
                 'purpose': {'A43': 4,
                  'A46': 7,
                  'A42': 3,
                  'A40': 0,
                  'A41': 1,
                  'A49': 9,
```



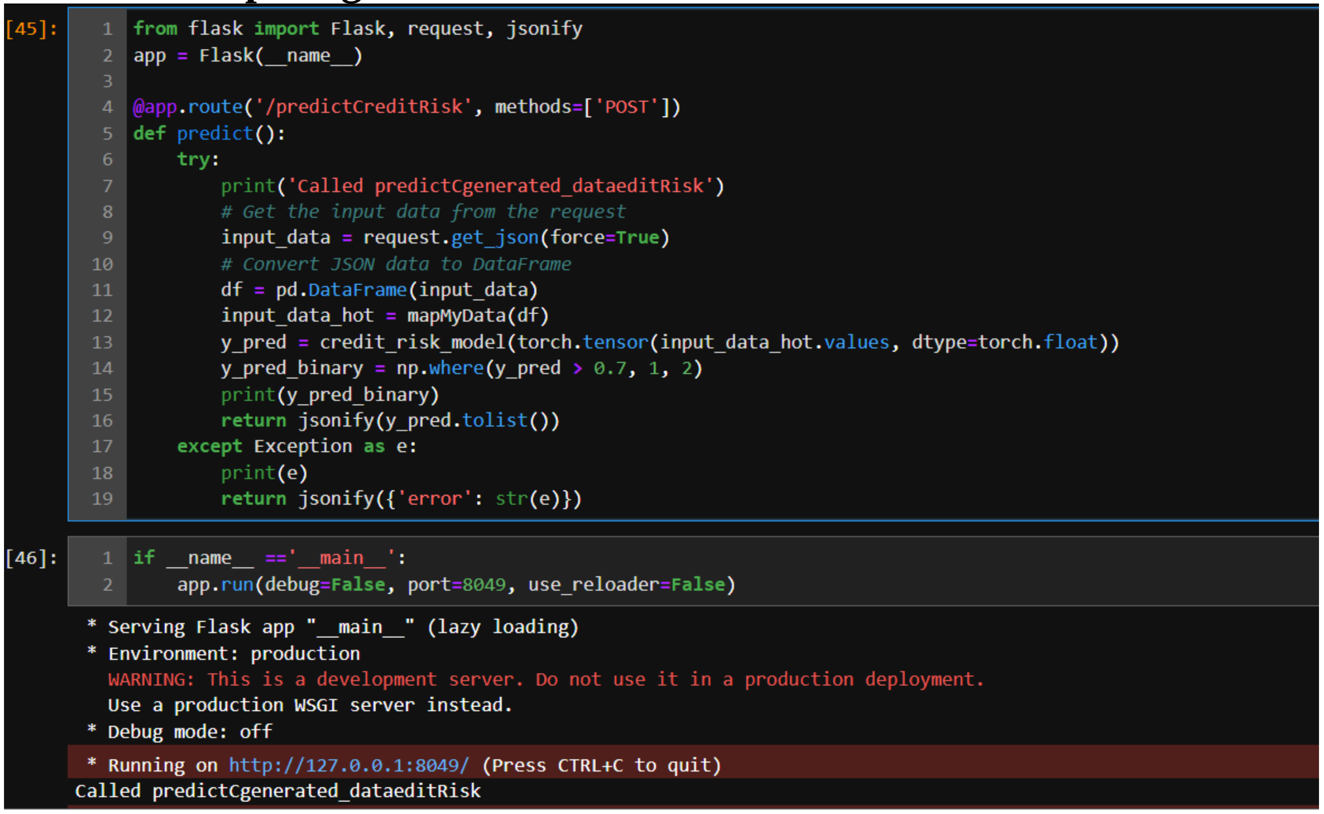
```python
                  'A49': 9,
                  'A44': 5,
                  'A45': 6,
                  'A410': 2,
                  'A48': 8},
                 'creditamount': {},
                 'savings': {'A65': 4, 'A61': 0, 'A63': 2, 'A64': 3, 'A62': 1},
                 'employmentsince': {'A75': 4, 'A73': 2, 'A74': 3, 'A71': 0, 'A72': 1},
                 'installmentrate': {},
                 'statussex': {'A93': 2, 'A92': 1, 'A91': 0, 'A94': 3},
                 'otherdebtors': {'A101': 0, 'A103': 2, 'A102': 1},
                 'residencesince': {},
                 'property': {'A121': 0, 'A122': 1, 'A124': 3, 'A123': 2},
                 'age': {},
                 'otherinstallmentplans': {'A143': 2, 'A141': 0, 'A142': 1},
                 'housing': {'A152': 1, 'A153': 2, 'A151': 0},
                 'existingcredits': {},
                 'job': {'A173': 2, 'A172': 1, 'A174': 3, 'A171': 0},
                 'peopleliable': {},
                 'telephone': {'A192': 1, 'A191': 0},
                 'foreignworker': {'A201': 0, 'A202': 1},
                 'classification': {1: 0, 2: 1}}
    #Iterate over input DataFrame
    for col in input_df.columns:

            if col in num_col:
                output_array = []
                for value in input_df[col]:
                    val_arr = num_map[col]
                    ret_val = 0.0
                    for pos in range(len(val_arr)):
                        if (value > val_arr[pos] and pos>0):
```



```python
                        if (value > val_arr[pos] and pos>0):
```

```python
                            ret_val = float(pos)
                    output_array.append(ret_val)
                    #print(col, value, ret_val)
                output_df[col] = output_array

            else:
                output_array = []
                for value in input_df[col]:
                    #print(col, value, Label_map[col][value])
                    output_array.append(Label_map[col][value])
                output_df[col] = output_array

    encrypted_df = output_df.apply(lambda x: x.apply(lambda y: CRE.encrypt_deterministic(CRE.getKey(), y)))
    output_df_hot = pd.get_dummies(encrypted_df, columns=encrypted_df.columns)
    encrypted_out_hot = pd.DataFrame(0,columns=encrypted_out_cols, index = range(input_df.shape[0]))
    for col in encrypted_out_hot.columns:
        if col in output_df_hot.columns:
            encrypted_out_hot[col]=output_df_hot[col]
            #print("Hey", col)
    return encrypted_out_hot
```

16. Exposing an API for Credit Risk Data Prediction

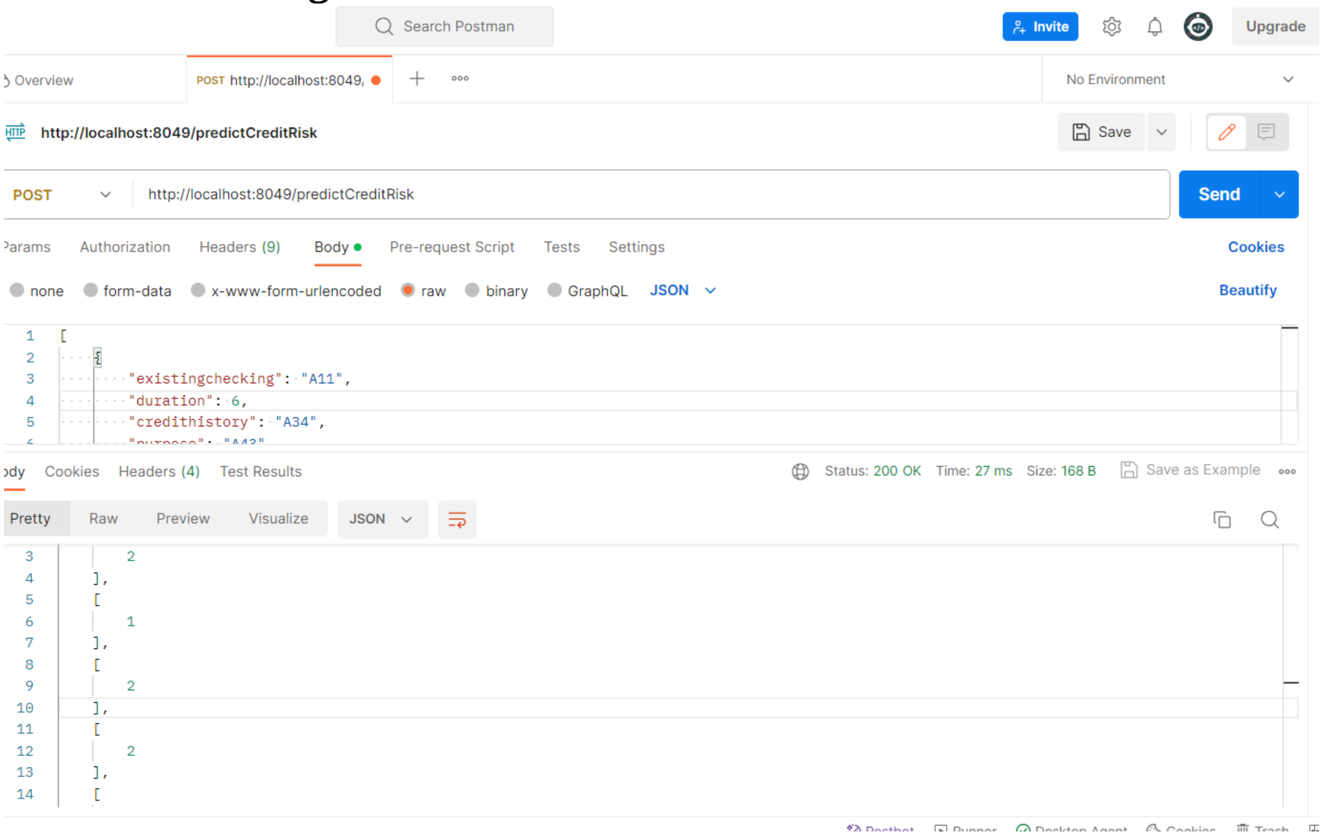


```
[45]:
from flask import Flask, request, jsonify
app = Flask(__name__)

@app.route('/predictCreditRisk', methods=['POST'])
def predict():
    try:
        print('Called predictCgenerated_dataeditRisk')
        # Get the input data from the request
        input_data = request.get_json(force=True)
        # Convert JSON data to DataFrame
        df = pd.DataFrame(input_data)
        input_data_hot = mapMyData(df)
        y_pred = credit_risk_model(torch.tensor(input_data_hot.values, dtype=torch.float))
        y_pred_binary = np.where(y_pred > 0.7, 1, 2)
        print(y_pred_binary)
        return jsonify(y_pred.tolist())
    except Exception as e:
        print(e)
        return jsonify({'error': str(e)})
```

```
[46]:
if __name__ =='__main__':
    app.run(debug=False, port=8049, use_reloader=False)
```

```
 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:8049/ (Press CTRL+C to quit)
Called predictCgenerated_dataeditRisk
```

17. Testing API Via Postman



http://localhost:8049/predictCreditRisk

POST http://localhost:8049/predictCreditRisk

```
[
    {
        "existingchecking": "A11",
        "duration": 6,
        "credithistory": "A34",
        "purpose": "A43"
```

Status: 200 OK Time: 27 ms Size: 168 B

Pretty JSON

```
        2
    ],
    [
        1
    ],
    [
        2
    ],
    [
        2
    ],
    [
```

CHAPTER 8

CONCLUSION

In this project, we used different deep learning approaches like deep neural networks, generative adversarial networks and privacy preserving data encryption techniques.

This project work highlights that homomorphic encryption approaches such as CKKS and Paillier encryption can be used for simple mathematical tasks but cannot be used with deep learning algorithms and this is an area of active research.

We were able to create a simple setup where credit data can be converted into an encrypted form and when fed into a neural network, we were able to classify good and bad credit.

On top of it, we were able to create GAN which was trained to generate synthetic data. In case of class imbalance issues, clients should be able to generate synthetic data which will improve overall training and prediction.

Overall loss and accuracy metrics were used to gauge model performance. We saw GAN loss decreasing as we increased the number of epochs. Similarly, with changing learning rates we saw classification performance improving and we got the best classification at the learning rate of .05.

CHAPTER 9

FUTURE WORK AND SCOPE OF IMPROVEMENT

The current project provides a proof-of-concept for using privacy preserving measures that can be used in the field of banking finance, healthcare, national security and other areas where leakage of private information can cause great risk to individual identity and threat organization's credibility.

We have used a very basic encryption algorithm as complex homomorphic encryptors like CKKS and Paillier caused issues with deep neural networks. This is an area of active research and any breakthrough in the same will bring great value for privacy preservation methods.

Secondly, the number of layers and nodes on the deep neural network have been minimal. However, this could be increased so as to improve the classification ability of the model. Also the model has been trained on some 800 data points and this can be very biased. More data added to the network will help to get better results.

We experimented with a limited range of learning rate and optimizer choice in paucity of time. A wider test on learning rates and optimizer (Adam, RMSprop, Adagrad, adadelta) could have helped us further improve performance of the model.

Adding regularization through drop-outs could have avoided any overfitting seen. We did not do the same as we did not get very high accuracy values.
Also, we had created GANs which could still be trained and evolved over larger data set to generate real looking data. Thus, GANs can further be explored to improve classification capacity of the binary classifier.

We believe that privacy preserving machine learning has the potential to revolutionize the way we use machine learning. By protecting the privacy of individuals, PPML can enable us to train and deploy machine learning models on sensitive data without compromising privacy. This will open up new possibilities for using machine learning to solve real-world problems.

[Available Online]

REFERENCES

[1].https://www.analyticsvidhya.com/blog/2022/02/privacy-preserving-in-machine-learning-ppml/

[2].https://dualitytech.com/blog/privacy-preserving-machine-learning

[3].https://tspace.library.utoronto.ca/bitstream/1807/125202/1/Atkins_Samuel_202211_MAS_thesis.pdf

[4].Welcome to PyTorch Tutorials — PyTorch Tutorials 1.13.1+cu117 documentation

[5].NumPy fundamentals — NumPy v1.24 Manual

[6]. https://link.springer.com/article/10.1007/s12083-021-01076-8/

[7].https://www.alexandra.dk/wp-content/uploads/2020/10/Alexandra-Instituttet-whitepaper-Privacy-Preserving-Machine-Learning-A-Practical-Guide.pdf

[8]. https://stackabuse.com/gpt-style-text-generation-in-python-with-tensorflowkeras/

[9]. Khandani, Amir E., et al. (2010). Consumer Credit-Risk Models via Machine-Learning Algorithms. https://www.sciencedirect.com/science/article/abs/pii/S0378426610002372

[10]. Lessmann, Stefan, et al. (2015). Benchmarking Classification Models for Credit Scoring: A Plea for Non-Parametric Methods.
https://www.sciencedirect.com/science/article/abs/pii/S0377221715004208

[11]. Shokri, Reza, and Vitaly Shmatikov. (2015). Privacy-Preserving Deep Learning.
https://www.cs.cornell.edu/~shmat/shmat_ccs15.pdf

[12]. Phong, Le Trieu, et al. (2018). Privacy-preserving Deep Learning with Application to Human Activity Recognition. https://www.researchgate.net/publication/318154664_Privacy-Preserving_Deep_Learning_Revisited_and_Enhanced

[13]. Goodfellow, Ian, et al. (2014). Generative Adversarial Nets.
https://doi.org/10.48550/arXiv.1406.2661

[14]. Popa, Raluca Ada, et al. (2011). CryptDB: Protecting Confidentiality with Encrypted Query Processing. https://web.cs.ucdavis.edu/~franklin/ecs228/2013/popa_etal_sosp_2011.pdf