INM427 Neural Computing

Group work

Supanut Sookkho (MSc Data Science / 230024841) & Yumi Heo (Msc Data Science / 230003122)

In [1]:
```
# Import the relevant libraries.

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import torch
import torch.nn as nn
import random
```

In [2]:
```
# Generate random numbers.


random.seed(123)
```

In [3]:
```
# The dataset is from Kaggle, and this dataset is about whether a customer will keep their bank account or not.
# Dataset URL: https://www.kaggle.com/competitions/playground-series-s4e1

# We use only the train set in Kaggle
# since the test set does not have their target data(the column name with 'Exited') due to the competition.
# We renamed the file of the train set into 'bank_churn_data.csv'.


df = pd.read_csv('bank_churn_data.csv')
df
```

Out[3]:

	id	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	0	15674932	Okwudilichukwu	668	France	Male	33.0	3	0.00	2	1	0	181449.97	0
1	1	15749177	Okwudiliolisa	627	France	Male	33.0	1	0.00	2	1	1	49503.50	0
2	2	15694510	Hsueh	678	France	Male	40.0	10	0.00	2	1	0	184866.69	0
3	3	15741417	Kao	581	France	Male	34.0	2	148882.54	1	1	1	84560.88	0
4	4	15766172	Chiemenam	716	Spain	Male	33.0	5	0.00	2	1	1	15068.83	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
165029	165029	15667085	Meng	667	Spain	Female	33.0	2	0.00	1	1	1	131834.75	0
165030	165030	15665521	Okechukwu	792	France	Male	35.0	3	0.00	1	0	0	131834.45	0
165031	165031	15664752	Hsia	565	France	Male	31.0	5	0.00	1	1	1	127429.56	0
165032	165032	15689614	Hsiung	554	Spain	Female	30.0	7	161533.00	1	0	1	71173.03	0
165033	165033	15732798	Ulyanov	850	France	Male	31.0	1	0.00	1	1	0	61581.79	1

165034 rows × 14 columns

Data wrangling

In [4]:
```
df.dtypes
```

Out[4]:
```
id                 int64
CustomerId         int64
Surname           object
CreditScore        int64
Geography         object
Gender            object
Age              float64
Tenure             int64
Balance          float64
NumOfProducts      int64
HasCrCard          int64
IsActiveMember     int64
EstimatedSalary  float64
Exited             int64
dtype: object
```

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
id                 0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

In [6]:
```
# Check the number of values in 'Gender' and 'Geography'.

print(df['Gender'].nunique())
print(df['Geography'].nunique())
```

```
2
3
```

In [7]:
```
# Remove irrelevant features for a model.

df = df.drop(['id', 'CustomerId', 'Surname'], axis=1)
df
```

Out[7]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	668	France	Male	33.0	3	0.00	2	1	0	181449.97	0
1	627	France	Male	33.0	1	0.00	2	1	1	49503.50	0
2	678	France	Male	40.0	10	0.00	2	1	0	184866.69	0
3	581	France	Male	34.0	2	148882.54	1	1	1	84560.88	0
4	716	Spain	Male	33.0	5	0.00	2	1	1	15068.83	0
...	...	...	...	...	...	...	...	...	...	...	...
165029	667	Spain	Female	33.0	2	0.00	1	1	1	131834.75	0
165030	792	France	Male	35.0	3	0.00	1	0	0	131834.45	0
165031	565	France	Male	31.0	5	0.00	1	1	1	127429.56	0
165032	554	Spain	Female	30.0	7	161533.00	1	0	1	71173.03	0
165033	850	France	Male	31.0	1	0.00	1	1	0	61581.79	1

165034 rows × 11 columns

In [8]:
```
# Change the string data to numeric one in 'Gender'.

map_dict = {'Female' : 0, 'Male' : 1}
df['Gender'] = df['Gender'].map(map_dict).astype(int)
```

In [9]:
```
df['Gender'].value_counts()
```

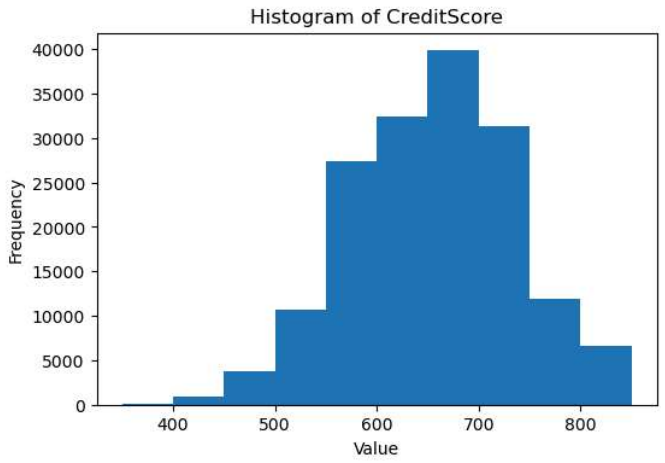
Out[9]:
```
1    93150
0    71884
Name: Gender, dtype: int64
```

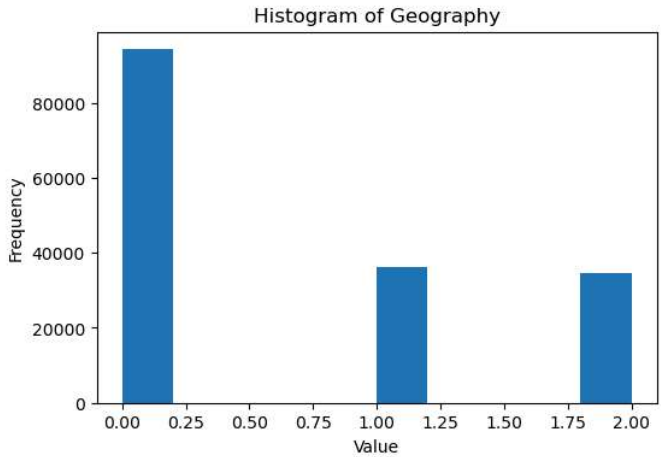
In [10]:
```
df['Geography'].value_counts()
```

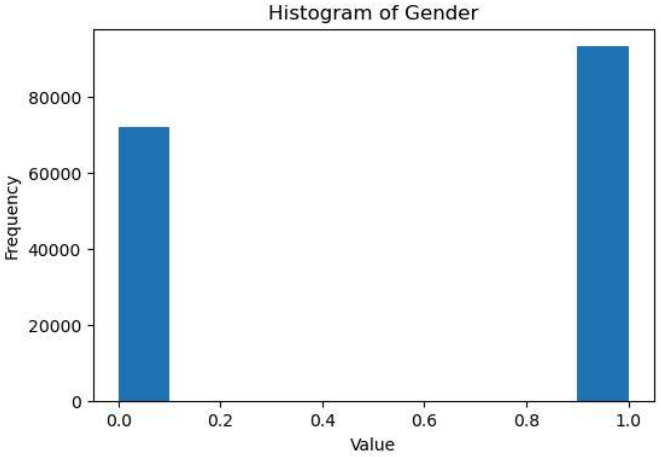
Out[10]:
```
France     94215
Spain      36213
Germany    34606
Name: Geography, dtype: int64
```

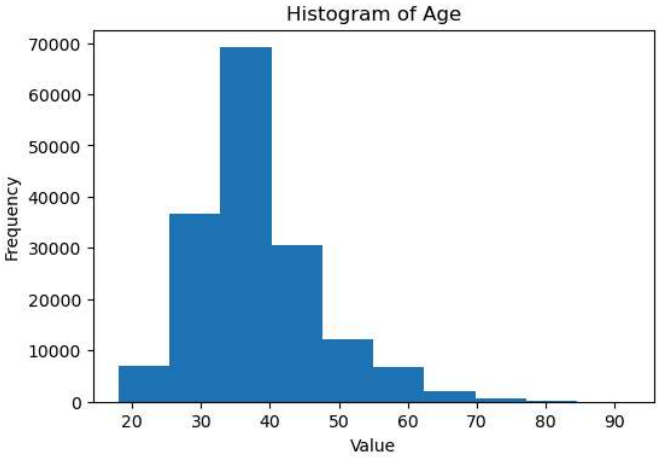
In [11]:
```
# Change the string data to numeric one in 'Geography'.

map_dict = {'France' : 0, 'Spain' : 1, 'Germany' : 2}
df['Geography'] = df['Geography'].map(map_dict).astype(int)
```

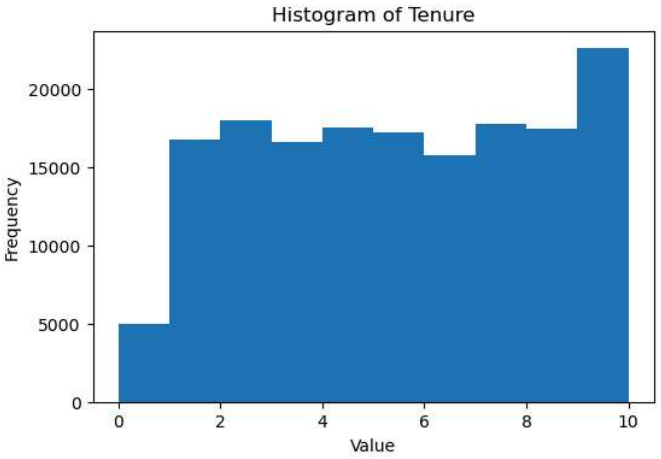
In [12]:
```
# Check each distribution.

for column in df.columns:
    plt.figure(figsize=(6, 4))
    plt.hist(df[column], bins=10)
    plt.title('Histogram of {}'.format(column))
    plt.xlabel('Value')
    plt.ylabel('Frequency')
    plt.show()
```

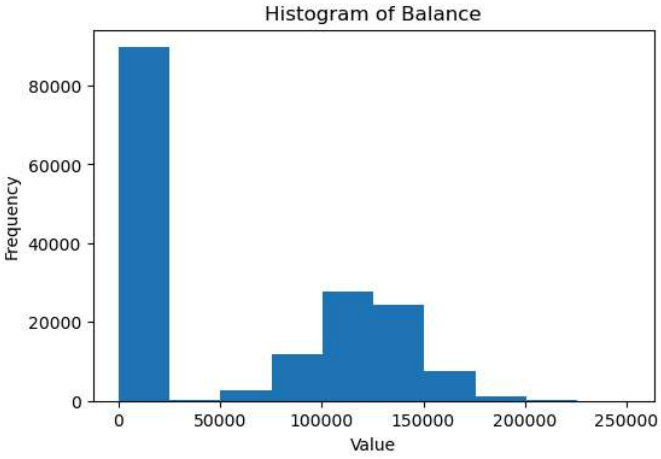


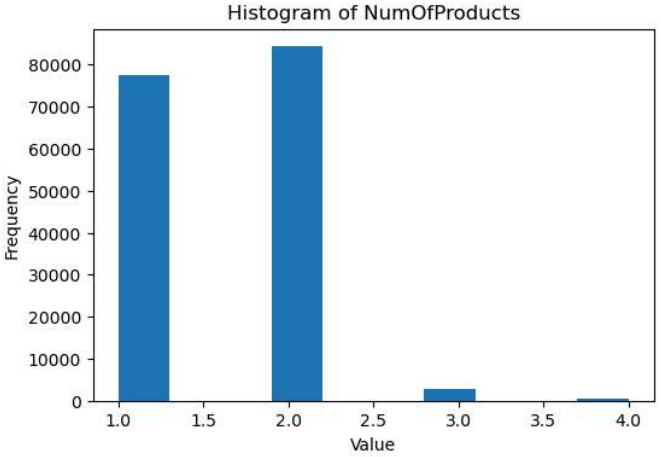


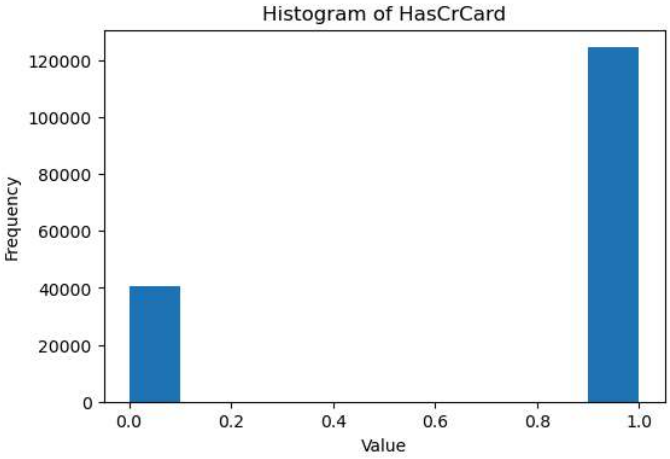

Histogram of Gender
Frequency
Value
0
20000
40000
60000
80000
0.0
0.2
0.4
0.6
0.8
1.0

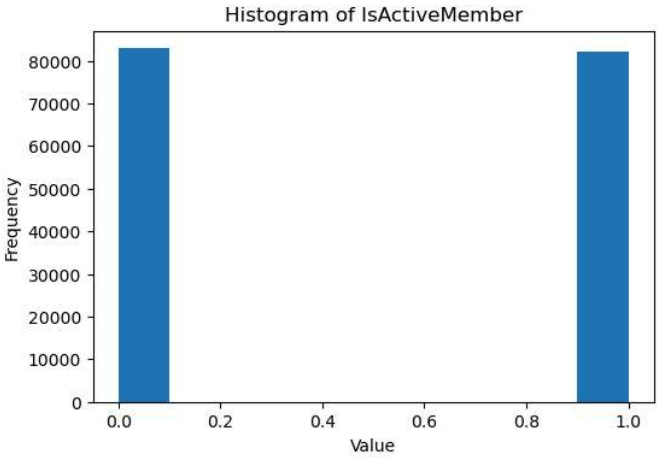

Histogram of Age
Frequency
Value
0
10000
20000
30000
40000
50000
60000
70000
20
30
40
50
60
70
80
90

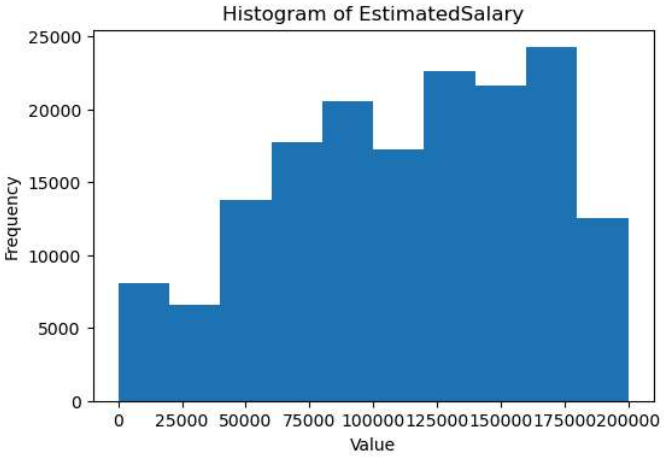

Histogram of Tenure
Frequency
Value
0
5000
10000
15000
20000
0
2
4
6
8
10

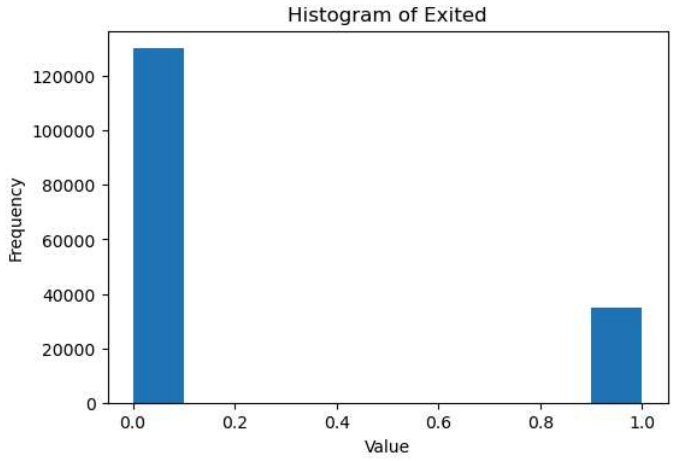

Histogram of Balance
Frequency
Value
0
20000
40000
60000
80000
0
50000
100000
150000
200000
250000


Histogram of NumOfProducts
Frequency
Value
0
10000
20000
30000
40000
50000
60000
70000
80000
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Histogram of HasCrCard
Frequency
Value
0
20000
40000
60000
80000
100000
120000
0.0
0.2
0.4
0.6
0.8
1.0


Histogram of IsActiveMember
Frequency
Value
0
10000
20000
30000
40000
50000
60000
70000
80000
0.0
0.2
0.4
0.6
0.8
1.0


Histogram of EstimatedSalary
Frequency
Value
0
5000
10000
15000
20000
25000
0
25000
50000
75000
100000
125000
150000
175000
200000



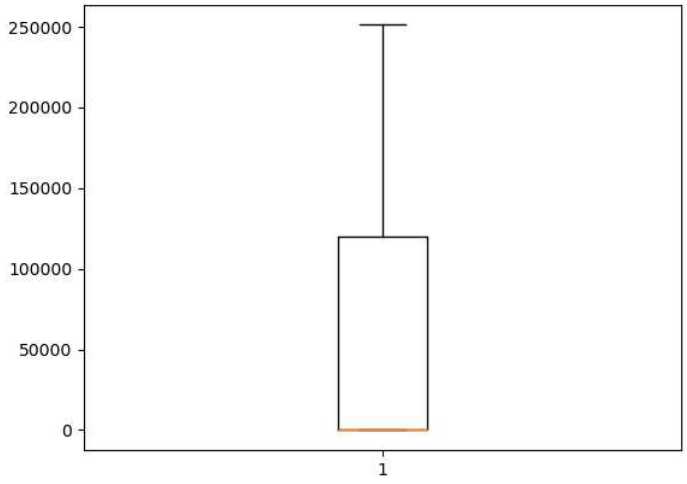
In [13]:
```
# There is a large proportion of the 'Balance' being 0.
# We will determine whether to remove the data with 0 in 'Balance'
# by conducting statistical analysis on the 'Balance'
# and rechecking the distribution of the other features and 'Exited'(target) that appear when the balance is 0.

df['Balance'].value_counts()
```

Out[13]:
```
0.00         89648
124577.33       88
127864.40       64
122314.50       63
129855.32       59
             ...
125824.21        1
158741.56        1
126815.52        1
61172.57         1
110993.29        1
Name: Balance, Length: 30075, dtype: int64
```

In [14]:
```
df['Balance'].describe()
```

Out[14]:
```
count    165034.000000
mean      55478.086689
std       62817.663278
min           0.000000
25%           0.000000
50%           0.000000
75%      119939.517500
max      250898.090000
Name: Balance, dtype: float64
```

In [15]:
```
plt.boxplot(df['Balance']);
```



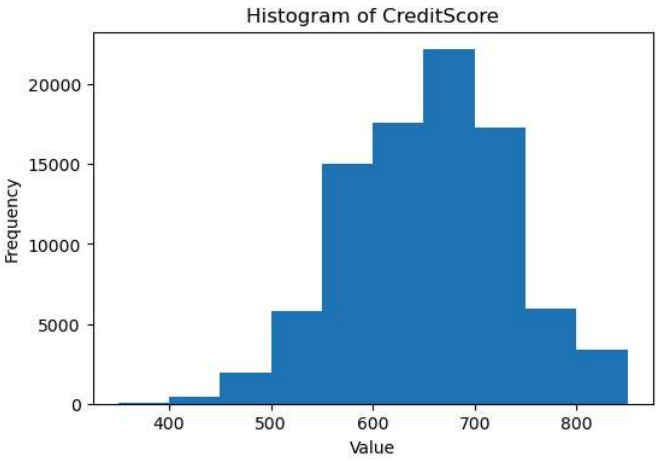
In [16]:
```
# Select the data with 0 in 'Balance'.

df_0_balance = df[df['Balance'] == 0]
df_0_balance
```

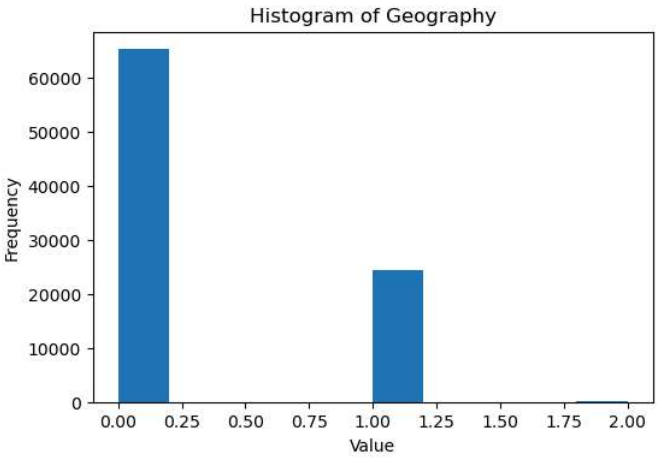
Out[16]:

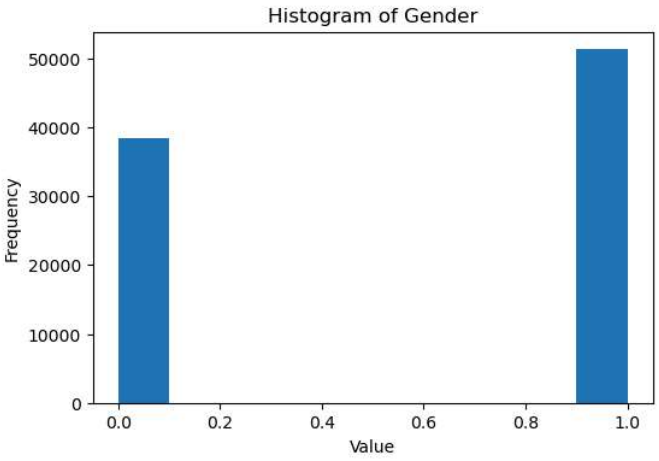
	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	668	0	1	33.0	3	0.0	2	1	0	181449.97	0
1	627	0	1	33.0	1	0.0	2	1	1	49503.50	0
2	678	0	1	40.0	10	0.0	2	1	0	184866.69	0
4	716	1	1	33.0	5	0.0	2	1	1	15068.83	0
8	676	0	1	43.0	4	0.0	2	1	0	142917.13	0
...	...	...	...	...	...	...	...	...	...	...	...
165028	630	0	1	50.0	8	0.0	2	1	1	5962.50	0
165029	667	1	0	33.0	2	0.0	1	1	1	131834.75	0
165030	792	0	1	35.0	3	0.0	1	0	0	131834.45	0
165031	565	0	1	31.0	5	0.0	1	1	1	127429.56	0
165033	850	0	1	31.0	1	0.0	1	1	0	61581.79	1

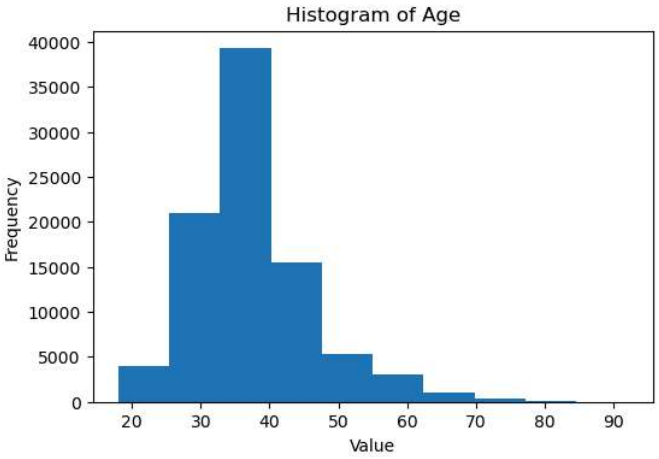
89648 rows × 11 columns

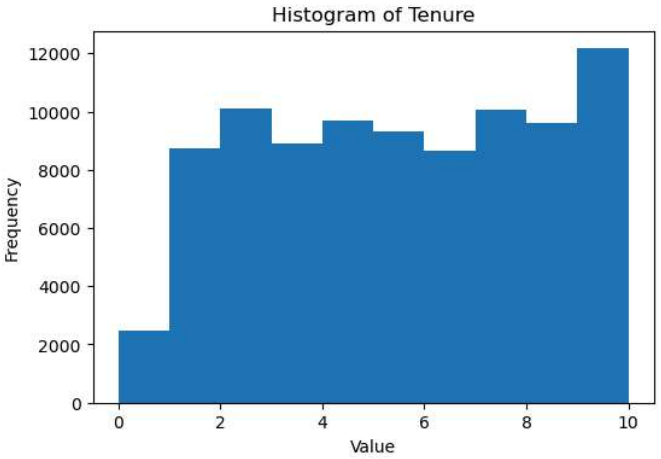
In [17]:
```
# Check each distribution of the features in 'df_0_balance'.

for column in df_0_balance.columns:
    plt.figure(figsize=(6, 4))
    plt.hist(df_0_balance[column], bins=10)
    plt.title('Histogram of {}'.format(column))
    plt.xlabel('Value')
    plt.ylabel('Frequency')
    plt.show()
```

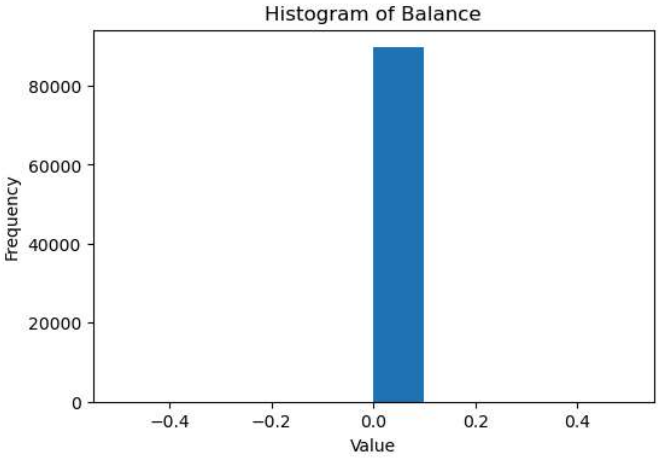


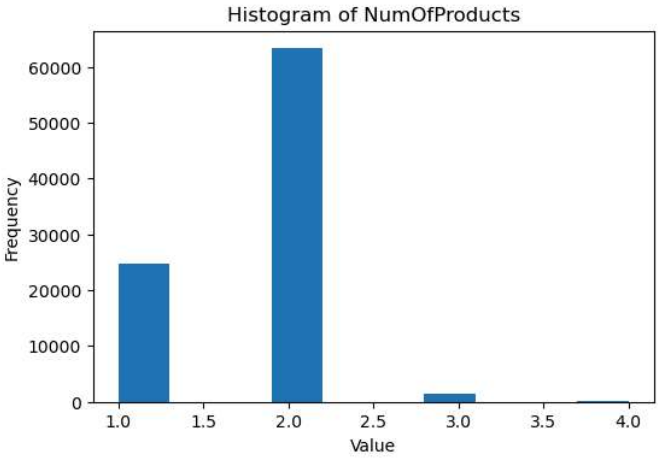


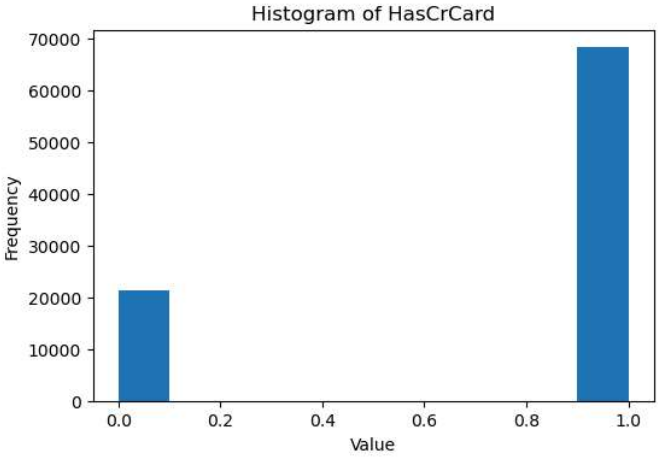


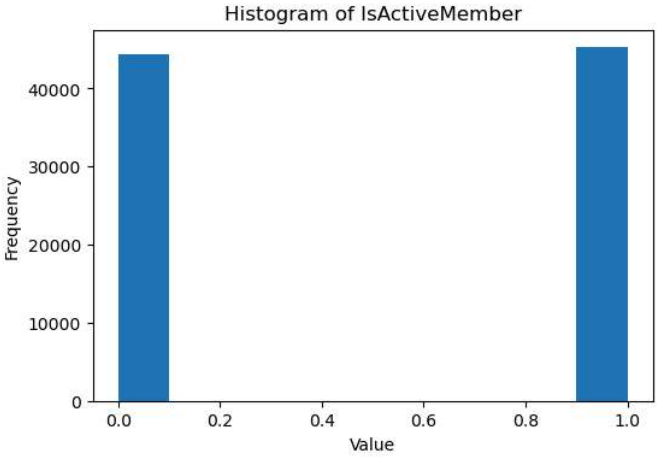


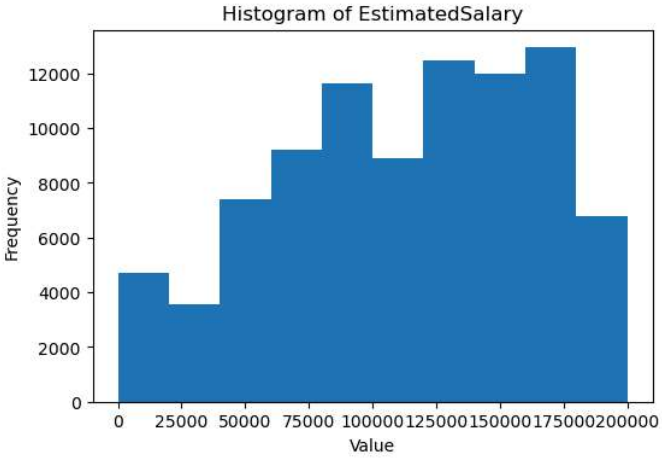

Histogram of Tenure
Frequency
Value
0
2000
4000
6000
8000
10000
12000
0
2
4
6
8
10

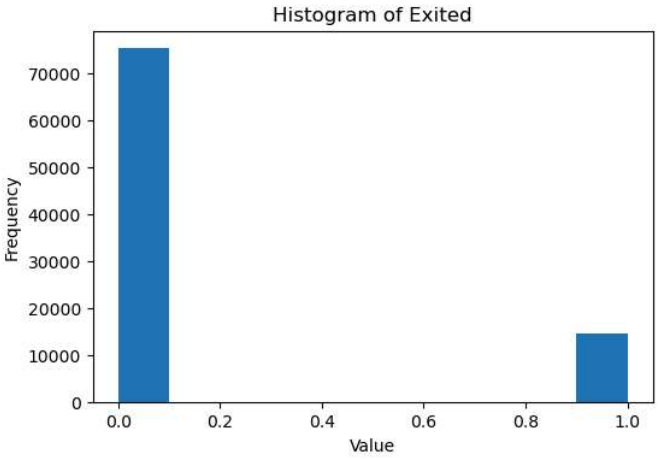

Histogram of Balance
Frequency
Value
0
20000
40000
60000
80000
−0.4
−0.2
0.0
0.2
0.4


Histogram of NumOfProducts
Frequency
Value
0
10000
20000
30000
40000
50000
60000
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Histogram of HasCrCard
Frequency
Value
0
10000
20000
30000
40000
50000
60000
70000
0.0
0.2
0.4
0.6
0.8
1.0

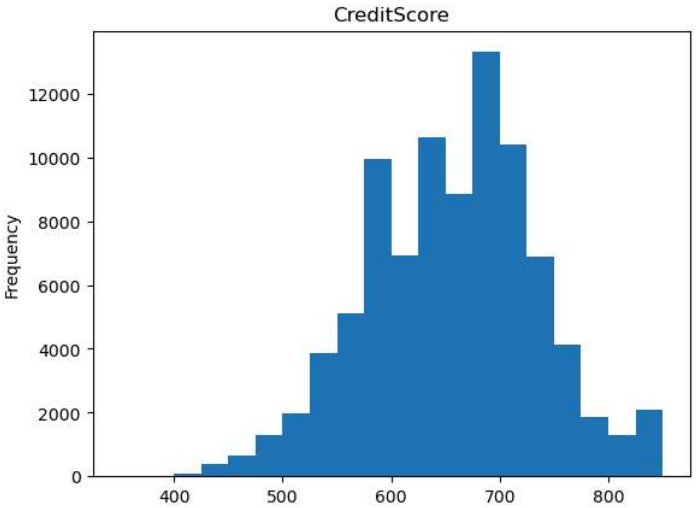






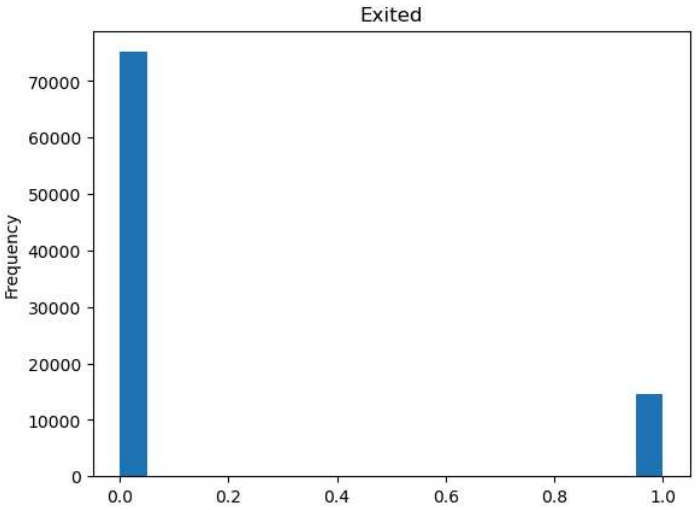
In [18]:
```
df_0_balance['CreditScore'].plot(kind='hist', bins=20, title='CreditScore')
```

Out[18]:
```
<Axes: title={'center': 'CreditScore'}, ylabel='Frequency'>
```



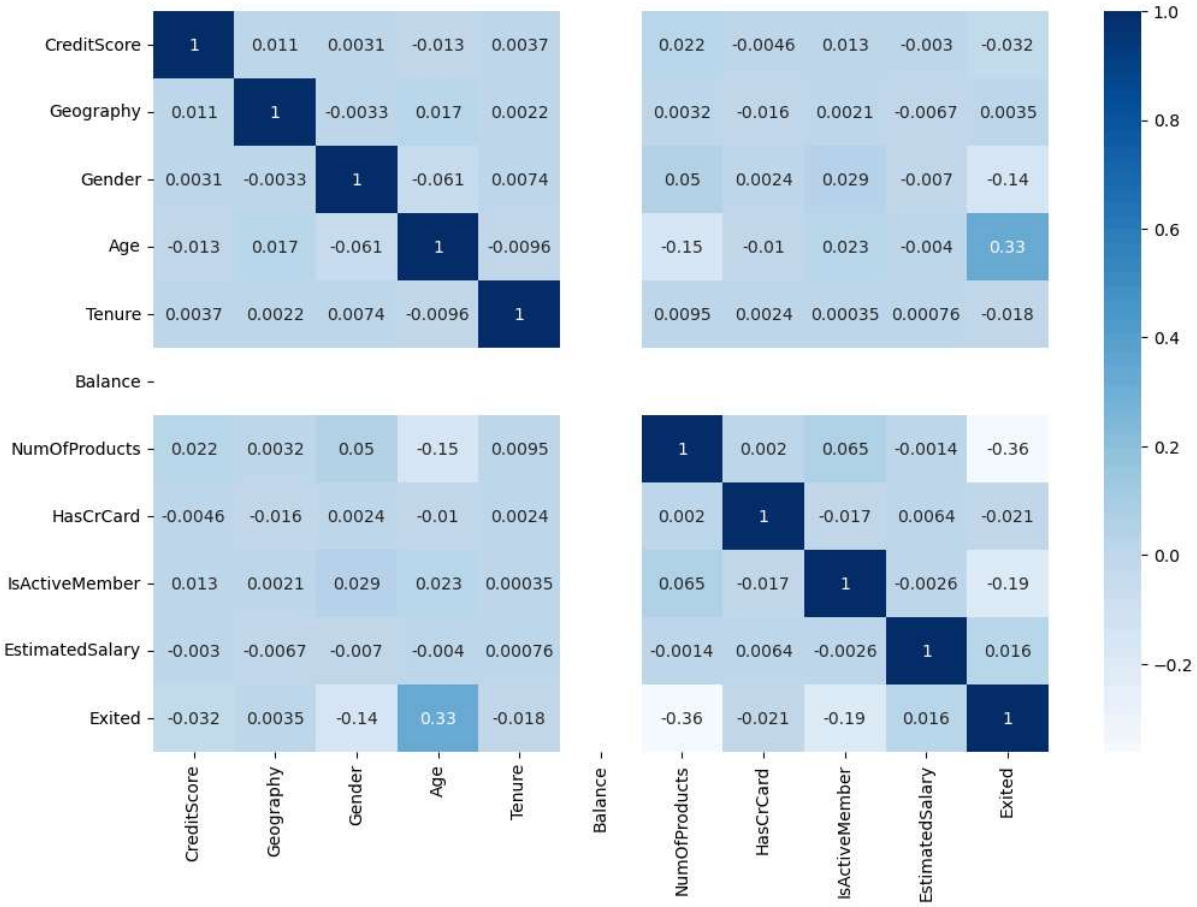
In [19]:
```
df_0_balance['Exited'].plot(kind='hist', bins=20, title='Exited')
```

Out[19]:
```
<Axes: title={'center': 'Exited'}, ylabel='Frequency'>
```



In [20]:
```
# Check the correlation coefficient of each feature.

correlation_matrix = df_0_balance.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, cmap='Blues', annot=True)
```

Out[20]: <Axes: >



The data with the value of 0 in 'Balance' will be remained since 'Balance' has considerably low collinearity with 'Exited'. Although, it looks problematic as the quantile range of 'Balance' shows 0 in 25% and 0 in 50%, the data with 0 in 'Balance' is still meaningful as it shows a similar distribution of the features to the distribution in the data with 0 and the other values in 'Balance'.

Split the data set

In [21]:
```
# Split the dataset into features and the target.
X = df.iloc[:, 0:10]
y = df.iloc[:, 10]
print(X)
print(y)
```

```
        CreditScore  Geography  Gender   Age  Tenure    Balance  W
0               668          0       1  33.0       3       0.00
1               627          0       1  33.0       1       0.00
2               678          0       1  40.0      10       0.00
3               581          0       1  34.0       2  148882.54
4               716          1       1  33.0       5       0.00
...             ...        ...     ...   ...     ...        ...
165029          667          1       0  33.0       2       0.00
165030          792          0       1  35.0       3       0.00
165031          565          0       1  31.0       5       0.00
165032          554          1       0  30.0       7  161533.00
165033          850          0       1  31.0       1       0.00

        NumOfProducts  HasCrCard  IsActiveMember  EstimatedSalary
0                   2          1               0        181449.97
1                   2          1               1         49503.50
2                   2          1               0        184866.69
3                   1          1               1         84560.88
4                   2          1               1         15068.83
...               ...        ...             ...              ...
165029              1          1               1        131834.75
165030              1          0               0        131834.45
165031              1          1               1        127429.56
165032              1          0               1         71173.03
165033              1          1               0         61581.79

[165034 rows x 10 columns]
0         0
1         0
2         0
3         0
4         0
         ..
165029    0
165030    0
165031    0
165032    0
165033    1
Name: Exited, Length: 165034, dtype: int64
```

In [22]:
```
print(type(X))
print(type(y))
print(X.shape)
print(y.shape)
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
(165034, 10)
(165034,)
```

In [23]:
```
# Convert Pandas to Numpy array.
X = X.values
y = y.values

print(type(X))
print(type(y))
print(X.shape)
print(y.shape)
```

```
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
(165034, 10)
(165034,)
```

In [24]:
```
# Convert Numpy array to Tensor.
X = torch.tensor(X, dtype=torch.float32)
y = torch.tensor(y, dtype=torch.float32)
print(type(X))
print(type(y))
print(X.shape)
print(y.shape)
```

```
<class 'torch.Tensor'>
<class 'torch.Tensor'>
torch.Size([165034, 10])
torch.Size([165034])
```

In [25]:
```
# Set the number of random state to shuffle the data.
random_state = 123

# Split the training set into 70% and testing set into 30%.
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.3,
                                                    random_state=random_state
)

# Check the data type.
print(X_train.dtype)

# Split the testing set into validation (50%) and test (50%).
X_test, X_vali, y_test, y_vali = train_test_split(X_test, y_test, test_size=0.5, random_state=random_state)
```

```
torch.float32
```

In [26]:
```
# Check the dimensionality and data type of each set.
print("Training dataset")
print(X_train.dtype, X_train.shape)
print(y_train.dtype, y_train.shape)

print("Validation dataset")
print(X_vali.dtype, X_vali.shape)
print(y_vali.dtype, y_vali.shape)

print("Testing dataset")
print(X_test.dtype, X_test.shape)
print(y_test.dtype, y_test.shape)
```

```
Training dataset
torch.float32 torch.Size([115523, 10])
torch.float32 torch.Size([115523])
Validation dataset
torch.float32 torch.Size([24756, 10])
torch.float32 torch.Size([24756])
Testing dataset
torch.float32 torch.Size([24755, 10])
torch.float32 torch.Size([24755])
```

In [27]:
```
# Import SMOTE to balance the target in training set.
from imblearn.over_sampling import SMOTE

# Create a variable for SMOTE
smote = SMOTE(random_state=random_state)

# Resampling the X and y in the training set using SMOTE.
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
```

In [28]:
```
# Check the dimensionality and data type of each set.
print("Training dataset")
print(X_train.dtype, X_train.shape)
print(y_train.dtype, y_train.shape)

print("Training dataset (Smote)")
print(X_train_smote.dtype, X_train_smote.shape)
print(y_train_smote.dtype, y_train_smote.shape)

print("Validation dataset")
print(X_vali.dtype, X_vali.shape)
print(y_vali.dtype, y_vali.shape)

print("Testing dataset")
print(X_test.dtype, X_test.shape)
print(y_test.dtype, y_test.shape)
```

```
Training dataset
torch.float32 torch.Size([115523, 10])
torch.float32 torch.Size([115523])
Training dataset (Smote)
float32 (182382, 10)
float32 (182382,)
Validation dataset
torch.float32 torch.Size([24756, 10])
torch.float32 torch.Size([24756])
Testing dataset
torch.float32 torch.Size([24755, 10])
torch.float32 torch.Size([24755])
```

Data scaling

In [29]:
```
# Normalise the features in each set.
from sklearn.preprocessing import MinMaxScaler
normal_scaler = MinMaxScaler()

X_train_smote_normalized = normal_scaler.fit_transform(X_train_smote)
X_vali_normalized = normal_scaler.transform(X_vali)
X_test_normalized = normal_scaler.transform(X_test)
```

In [30]:
```
# Convert to Tensor and rename the variables for the features in each set.
X_train = X_train_smote_normalized = torch.tensor(X_train_smote_normalized, dtype=torch.float32)
X_vali = X_vali_normalized = torch.tensor(X_vali_normalized, dtype=torch.float32)
X_test = X_test_normalized = torch.tensor(X_test_normalized, dtype=torch.float32)

# Rename y_train_smote to y_train.
y_train = torch.tensor(y_train_smote, dtype=torch.float32)
```

In [31]:
```
# Check the size and type of datasets after the splitting

print("Training dataset (Smote)")
print(X_train.dtype, X_train.shape)
print(y_train.dtype, y_train.shape)

print("Validation dataset")
print(X_vali.dtype, X_vali.shape)
print(y_vali.dtype, y_vali.shape)

print("Testing dataset")
print(X_test.dtype, X_test.shape)
print(y_test.dtype, y_test.shape)
```

```
Training dataset (Smote)
torch.float32 torch.Size([182382, 10])
torch.float32 torch.Size([182382])
Validation dataset
torch.float32 torch.Size([24756, 10])
torch.float32 torch.Size([24756])
Testing dataset
torch.float32 torch.Size([24755, 10])
torch.float32 torch.Size([24755])
```

In [32]:
```
# Increase the dimension of all target variables.
y_train = y_train.unsqueeze(1)
y_test = y_test.unsqueeze(1)
y_vali = y_vali.unsqueeze(1)

# Make sure that the data type and dimension of each set are all clear.
print("Training dataset")
print(X_train.dtype, X_train.shape, type(X_train))
print(y_train.dtype, y_train.shape, type(y_train))

print("Validation dataset")
print(X_vali.dtype, X_vali.shape, type(X_vali))
print(y_vali.dtype, y_vali.shape, type(y_vali))

print("Testing dataset")
print(X_test.dtype, X_test.shape, type(X_train))
print(y_test.dtype, y_test.shape, type(y_train))
```

```
Training dataset
torch.float32 torch.Size([182382, 10]) <class 'torch.Tensor'>
torch.float32 torch.Size([182382, 1]) <class 'torch.Tensor'>
Validation dataset
torch.float32 torch.Size([24756, 10]) <class 'torch.Tensor'>
torch.float32 torch.Size([24756, 1]) <class 'torch.Tensor'>
Testing dataset
torch.float32 torch.Size([24755, 10]) <class 'torch.Tensor'>
torch.float32 torch.Size([24755, 1]) <class 'torch.Tensor'>
```

In [33]:
```
# To ensure a fair comparison between Python and Matlab,
# we use the same split datasets to train, validate, and test a model in both environments.
# Tensors are converted into CSV files for application in Matlab.
# After conversion, the following codes are commented out.

# Convert the datasets from Tensor to Numpy
#X_train_ndarray = X_train.numpy()
#X_test_ndarray = X_test.numpy()
#X_vali_ndarray = X_vali.numpy()
#y_train_ndarray = y_train.numpy()
#y_test_ndarray = y_test.numpy()
#y_vali_ndarray = y_vali.numpy()

# Save all datasets to CSV file
#np.savetxt('X_train_ndarray.csv', X_train_ndarray, delimiter=',')
#np.savetxt('X_test_ndarray.csv', X_test_ndarray, delimiter=',')
#np.savetxt('X_vali_ndarray.csv', X_vali_ndarray, delimiter=',')
#np.savetxt('y_train_ndarray.csv', y_train_ndarray, delimiter=',')
#np.savetxt('y_test_ndarray.csv', y_test_ndarray, delimiter=',')
#np.savetxt('y_vali_ndarray.csv', y_vali_ndarray, delimiter=',')
```

Build a neural network model

In [34]:
```
# Define the structure of the neural network model.
input_features = 10
output_features = 1

# Set the list of hyperparameters for grid search.
learning_rates_grid = [0.01, 0.1, 0.2, 0.3]
hidden_neurons_grid = [10, 25, 50, 100, 200]

# Set the number of epochs.
number_of_epochs = 1000

# Define variables to store the item from the result of the model.
best_accuracy = 0.0
best_hyperparameters = {}

# Make empty lists to store the result of every combination of hyperparameters for visualisation.
learning_rates_list_heatmap = []
hidden_neurons_list_heatmap = []
accuracy_list_heatmap = []

# Make an empty list to store the elapsed time from 1000 epochs of the best model.
import time
python_training_time_bestmodel = []
```

In [35]:
```
# Make the class for the neural network model.
class Credit_Churn_NN_Model(nn.Module):
    def __init__(self, input_features, hidden_neurons, output_features):
        super(Credit_Churn_NN_Model, self).__init__()
        self.fc1 = nn.Linear(input_features, hidden_neurons)
        self.fc2 = nn.Linear(hidden_neurons, output_features)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        output_layer1 = self.fc1(x)
        l1_activated = self.relu(output_layer1)
        output_layer2 = self.fc2(l1_activated)
        output_activated = self.sigmoid(output_layer2)
        return output_activated
```

In [36]:
```
# Make the function to train the model.
def train_NN_model(learning_rate, hidden_neurons, X_train, y_train, X_vali, y_vali, number_of_epochs):
    model = Credit_Churn_NN_Model(input_features, hidden_neurons, output_features)
    criterion = torch.nn.BCELoss(reduction='mean')
    optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

    # Make lists to store the item from the loss and elapsed time in order to plot a graph.
    plot_train_loss = []
    plot_vali_loss = []
    plot_train_time = []

    # Iterate the model following each epoch.
    for e in range(number_of_epochs):
        # Train the model.
        model.train()
        # Start the timer.
        start_time = time.time()
        # Make the gradient descent start from 0
        optimizer.zero_grad()
        y_train_pred = model(X_train)
        # Call the loss function.
        train_loss = criterion(y_train_pred, y_train)
        # Implement backpropagation.
        train_loss.backward()
        optimizer.step()
        # Stop the timer.
        end_time = time.time()
        # Calculate the elapsed time.
        Total_time = end_time - start_time
        plot_train_time.append(Total_time)
        # Append the loss value from training to the following list.
        plot_train_loss.append(train_loss)

        # Validate the model.
        model.eval()
        with torch.no_grad():
            y_vali_pred = model(X_vali)
            vali_loss = criterion(y_vali_pred, y_vali)
```

```
            # Append the loss value from validation to the following list.
            plot_vali_loss.append(vali_loss)

    return model, plot_train_loss, plot_vali_loss, plot_train_time
```

Train the model with grid search

In [37]:
```
# Define an empty list to store the training time from the best model.
Total_time = []

# Apply a nested loop for grid search.
for learning_rate in learning_rates_grid:
    for hidden_neurons in hidden_neurons_grid:
        print(f"\nTraining the NNet model with learning rate={learning_rate} and hidden_neurons={hidden_neurons}")
        # Call the function to train and evaluate the model with the nested loop.
        trained_model, train_loss, vali_loss, train_time = train_NN_model(learning_rate, hidden_neurons,
                                                                          X_train, y_train,
                                                                          X_vali, y_vali,
                                                                          number_of_epochs)

        # Get the predictions from the X_val, and convert them to either 1 or 0.
        y_vali_pred = (trained_model(X_vali).detach().numpy() > 0.5).astype(int)
        # Calculate the validation accuracy.
        accuracy_validation_set = (y_vali_pred == y_vali.numpy()).mean()
        print("The validation accuracy = %.2f%%" % (accuracy_validation_set*100))

        # Append each value to the list for plotting.
        learning_rates_list_heatmap.append(learning_rate)
        hidden_neurons_list_heatmap.append(hidden_neurons)
        accuracy_list_heatmap.append(accuracy_validation_set)
        Total_time.append(train_time)

        # Pick the best model comparing the validation accuracy from each model.
        if accuracy_validation_set > best_accuracy: # If it is so,
            best_accuracy = accuracy_validation_set # Reassign the validation accuracy to the variable of 'best_accuracy'.
            # Define the best hyperparameters with the selected learning rate and the number of neurons.
            best_hyperparameters = {'learning_rate': learning_rate, 'hidden_neurons': hidden_neurons}
            # Pick the value of the loss and training time from the best model.
            train_loss_for_plotting = train_loss
            vali_loss_for_plotting = vali_loss
            python_training_time_bestmodel = train_time
            # Replace the model with the best model.
            best_model = trained_model

# Print the best hyperparameters from grid search.
print("\nThe Best Hyperparameters are ", best_hyperparameters)
```

```
Training the NNet model with learning rate=0.01 and hidden_neurons=10
The validation accuracy = 67.73%

Training the NNet model with learning rate=0.01 and hidden_neurons=25
The validation accuracy = 68.15%

Training the NNet model with learning rate=0.01 and hidden_neurons=50
The validation accuracy = 69.98%

Training the NNet model with learning rate=0.01 and hidden_neurons=100
The validation accuracy = 68.22%

Training the NNet model with learning rate=0.01 and hidden_neurons=200
The validation accuracy = 68.61%

Training the NNet model with learning rate=0.1 and hidden_neurons=10
The validation accuracy = 74.89%

Training the NNet model with learning rate=0.1 and hidden_neurons=25
The validation accuracy = 74.89%

Training the NNet model with learning rate=0.1 and hidden_neurons=50
The validation accuracy = 75.10%

Training the NNet model with learning rate=0.1 and hidden_neurons=100
The validation accuracy = 75.18%

Training the NNet model with learning rate=0.1 and hidden_neurons=200
The validation accuracy = 75.88%

Training the NNet model with learning rate=0.2 and hidden_neurons=10
The validation accuracy = 75.27%

Training the NNet model with learning rate=0.2 and hidden_neurons=25
The validation accuracy = 76.19%

Training the NNet model with learning rate=0.2 and hidden_neurons=50
The validation accuracy = 76.86%

Training the NNet model with learning rate=0.2 and hidden_neurons=100
The validation accuracy = 76.97%

Training the NNet model with learning rate=0.2 and hidden_neurons=200
The validation accuracy = 78.32%

Training the NNet model with learning rate=0.3 and hidden_neurons=10
The validation accuracy = 77.48%

Training the NNet model with learning rate=0.3 and hidden_neurons=25
The validation accuracy = 78.12%

Training the NNet model with learning rate=0.3 and hidden_neurons=50
The validation accuracy = 78.29%

Training the NNet model with learning rate=0.3 and hidden_neurons=100
The validation accuracy = 73.21%

Training the NNet model with learning rate=0.3 and hidden_neurons=200
The validation accuracy = 75.07%

The Best Hyperparameters are  {'learning_rate': 0.2, 'hidden_neurons': 200}
```

In [38]:
```
# Check the best combination of hyperparameters.
print(best_hyperparameters)
```

```
best_learning_rate = best_hyperparameters['learning_rate']
best_hidden_neurons = best_hyperparameters['hidden_neurons']

print(best_learning_rate)
print(best_hidden_neurons)
```

```
{'learning_rate': 0.2, 'hidden_neurons': 200}
0.2
200
```

In [39]:
```
# Check the loss values from the best model.
train_loss_for_plotting
```

Out[39]:
```
[tensor(0.7019, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6946, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6899, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6858, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6819, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6783, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6748, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6714, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6682, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6652, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6622, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6593, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6566, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6540, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6514, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6490, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6466, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6443, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6421, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6400, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6379, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6359, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6340, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6321, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6303, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6286, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6269, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6253, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6237, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6222, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6207, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6193, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6179, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6166, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6153, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6141, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6129, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6117, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6105, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6094, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6084, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6073, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6063, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6053, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6044, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6035, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6025, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6017, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6008, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.6000, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5992, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5984, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5976, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5968, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5961, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5953, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5946, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5939, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5932, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5925, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5919, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5912, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5906, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5899, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5893, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5887, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5881, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5875, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5869, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5863, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5857, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5852, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5846, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5841, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5835, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5830, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5824, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5819, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5814, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5808, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5803, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5798, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5793, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5788, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5783, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5777, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5772, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5767, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5763, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5758, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5753, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5748, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5743, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5738, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5733, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5729, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5724, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5719, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5714, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5710, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5705, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5700, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5696, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5691, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5687, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5682, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5677, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5673, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5668, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5664, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5659, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5655, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5650, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5646, grad_fn=<BinaryCrossEntropyBackward0>),
 tensor(0.5642, grad_fn=<BinaryCrossEntropyBackward0>),
```

tensor(0.5637, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5633, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5628, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5624, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5620, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5615, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5611, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5607, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5602, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5598, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5594, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5590, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5585, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5581, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5577, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5573, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5568, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5564, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5560, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5556, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5552, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5548, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5544, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5540, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5536, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5532, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5527, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5523, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5519, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5515, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5511, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5508, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5504, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5500, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5496, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5492, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5488, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5484, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5480, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5476, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5473, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5469, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5465, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5461, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5457, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5454, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5450, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5446, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5443, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5439, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5435, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5432, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5428, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5424, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5421, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5417, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5414, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5410, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5406, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5403, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5399, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5396, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5393, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5389, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5386, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5382, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5379, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5375, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5372, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5369, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5365, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5362, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5359, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5356, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5352, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5349, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5346, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5343, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5339, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5336, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5333, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5330, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5327, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5324, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5321, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5317, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5314, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5311, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5308, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5305, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5302, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5299, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5296, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5293, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5291, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5288, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5285, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5282, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5279, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5276, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5273, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5271, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5268, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5265, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5262, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5259, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5257, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5254, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5251, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5249, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5246, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5243, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5241, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5238, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5235, grad_fn=<BinaryCrossEntropyBackward0>),

tensor(0.5233, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5230, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5228, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5225, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5223, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5220, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5218, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5215, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5213, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5210, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5208, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5205, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5203, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5201, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5198, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5196, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5194, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5191, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5189, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5187, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5184, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5182, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5180, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5178, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5175, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5173, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5171, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5169, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5166, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5164, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5162, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5160, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5158, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5156, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5154, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5152, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5149, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5147, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5145, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5143, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5141, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5139, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5137, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5135, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5133, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5131, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5129, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5127, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5125, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5123, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5122, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5120, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5118, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5116, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5114, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5112, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5110, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5108, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5107, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5105, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5103, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5101, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5099, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5098, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5096, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5094, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5092, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5091, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5089, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5087, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5086, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5084, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5082, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5080, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5079, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5077, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5076, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5074, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5072, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5071, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5069, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5067, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5066, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5064, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5063, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5061, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5060, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5058, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5057, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5055, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5053, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5052, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5050, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5049, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5047, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5046, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5045, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5043, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5042, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5040, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5039, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5037, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5036, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5034, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5033, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5032, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5030, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5029, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5027, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5026, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5025, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5023, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5022, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5021, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5019, grad_fn=<BinaryCrossEntropyBackward0>),

tensor(0.5018, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5017, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5015, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5014, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5013, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5011, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5010, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5009, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5007, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5006, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5005, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5004, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5002, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5001, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.5000, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4999, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4997, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4996, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4995, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4994, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4992, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4991, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4990, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4989, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4988, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4986, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4985, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4984, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4983, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4982, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4981, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4979, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4978, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4977, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4976, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4975, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4974, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4972, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4971, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4970, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4969, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4968, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4967, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4966, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4965, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4964, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4962, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4961, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4960, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4959, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4958, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4957, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4956, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4955, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4954, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4953, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4952, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4951, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4950, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4949, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4947, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4946, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4945, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4944, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4943, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4942, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4941, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4940, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4939, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4938, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4937, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4936, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4935, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4934, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4933, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4932, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4931, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4930, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4929, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4928, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4927, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4926, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4925, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4924, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4923, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4922, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4921, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4921, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4920, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4919, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4918, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4917, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4916, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4915, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4914, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4913, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4912, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4911, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4910, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4909, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4908, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4907, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4906, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4905, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4904, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4904, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4903, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4902, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4901, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4900, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4899, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4898, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4897, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4896, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4895, grad_fn=<BinaryCrossEntropyBackward0>),

tensor(0.4894, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4894, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4893, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4892, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4891, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4890, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4889, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4888, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4887, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4886, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4886, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4885, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4884, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4883, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4882, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4881, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4880, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4879, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4878, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4878, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4877, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4876, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4875, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4874, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4873, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4872, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4871, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4871, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4870, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4869, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4868, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4867, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4866, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4865, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4865, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4864, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4863, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4862, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4861, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4860, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4859, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4859, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4858, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4857, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4856, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4855, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4854, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4854, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4853, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4852, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4851, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4850, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4849, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4848, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4848, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4847, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4846, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4845, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4844, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4843, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4843, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4842, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4841, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4840, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4839, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4838, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4838, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4837, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4836, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4835, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4834, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4833, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4833, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4832, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4831, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4830, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4829, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4828, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4828, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4827, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4826, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4825, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4824, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4823, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4823, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4822, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4821, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4820, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4819, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4819, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4818, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4817, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4816, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4815, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4814, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4814, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4813, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4812, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4811, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4810, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4810, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4809, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4808, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4807, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4806, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4805, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4805, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4804, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4803, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4802, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4801, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4801, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4800, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4799, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4798, grad_fn=<BinaryCrossEntropyBackward0>),

tensor(0.4797, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4796, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4796, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4795, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4794, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4793, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4792, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4791, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4791, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4790, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4789, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4788, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4787, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4787, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4786, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4785, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4784, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4783, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4782, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4782, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4781, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4780, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4779, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4778, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4778, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4777, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4776, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4775, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4774, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4773, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4773, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4772, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4771, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4770, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4769, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4769, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4768, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4767, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4766, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4765, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4764, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4764, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4763, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4762, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4761, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4760, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4760, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4759, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4758, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4757, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4756, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4755, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4755, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4754, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4753, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4752, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4751, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4750, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4750, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4749, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4748, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4747, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4746, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4745, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4745, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4744, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4743, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4742, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4741, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4741, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4740, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4739, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4738, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4737, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4736, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4736, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4735, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4734, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4733, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4732, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4731, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4731, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4730, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4729, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4728, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4727, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4726, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4726, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4725, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4724, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4723, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4722, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4721, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4720, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4720, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4719, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4718, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4717, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4716, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4715, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4715, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4714, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4713, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4712, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4711, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4710, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4710, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4709, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4708, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4707, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4706, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4705, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4704, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4704, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4703, grad_fn=<BinaryCrossEntropyBackward0>),

tensor(0.4702, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4701, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4700, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4699, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4698, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4698, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4697, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4696, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4695, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4694, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4693, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4692, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4692, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4691, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4690, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4689, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4688, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4687, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4686, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4686, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4685, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4684, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4683, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4682, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4681, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4680, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4679, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4679, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4678, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4677, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4676, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4675, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4674, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4673, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4672, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4672, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4671, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4670, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4669, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4668, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4667, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4666, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4665, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4665, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4664, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4663, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4662, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4661, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4660, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4659, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4658, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4658, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4657, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4656, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4655, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4654, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4653, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4652, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4652, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4651, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4650, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4649, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4648, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4647, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4646, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4645, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4645, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4644, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4643, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4642, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4641, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4640, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4639, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4638, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4638, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4637, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4636, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4635, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4634, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4633, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4632, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4631, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4631, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4630, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4629, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4628, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4627, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4626, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4625, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4624, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4624, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4623, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4622, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4621, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4620, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4619, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4618, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4617, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4617, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4616, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4615, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4614, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4613, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4612, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4611, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4610, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4610, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4609, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4608, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4607, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4606, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4605, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4604, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4603, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4603, grad_fn=<BinaryCrossEntropyBackward0>),

tensor(0.4602, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4601, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4600, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4599, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4598, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4597, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4596, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4595, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4595, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4594, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4593, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4592, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4591, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4590, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4589, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4588, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4588, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4587, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4586, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4585, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4584, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4583, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4582, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4581, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4580, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4580, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4579, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4578, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4577, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4576, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4575, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4574, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4573, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4572, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4572, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4571, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4570, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4569, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4568, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4567, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4566, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4565, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4564, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4563, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4563, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4562, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4561, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4560, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4559, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4558, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4557, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4556, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4555, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4555, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4554, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4553, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4552, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4551, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4550, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4549, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4548, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4547, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4546, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4546, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4545, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4544, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4543, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4542, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4541, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4540, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4539, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4538, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4538, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4537, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4536, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4535, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4534, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4533, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4532, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4531, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4530, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4529, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4529, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4528, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4527, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4526, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4525, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4524, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4523, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4522, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4521, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4520, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4519, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4519, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4518, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4517, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4516, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4515, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4514, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4513, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4512, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4511, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4510, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4510, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4509, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4508, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4507, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4506, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4505, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4504, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4503, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4502, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4501, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4501, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4500, grad_fn=<BinaryCrossEntropyBackward0>),

```
tensor(0.4499, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4498, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4497, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4496, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4495, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4494, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4493, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4492, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4491, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4491, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4490, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4489, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4488, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4487, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4486, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4485, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4484, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4483, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4482, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4481, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4481, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4480, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4479, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4478, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4477, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4476, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4475, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4474, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4473, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4472, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4471, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4471, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4470, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4469, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4468, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4467, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4466, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4465, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4464, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4463, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4462, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4461, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4461, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4460, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4459, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4458, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4457, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4456, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4455, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4454, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4453, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4452, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4451, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4450, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4450, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4449, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4448, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4447, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4446, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4445, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4444, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4443, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4442, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4441, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4440, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4440, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4439, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4438, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4437, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4436, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4435, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4434, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4433, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4432, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4431, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4430, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4430, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4429, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4428, grad_fn=<BinaryCrossEntropyBackward0>),
tensor(0.4427, grad_fn=<BinaryCrossEntropyBackward0>)]
```

In [40]:
```
# Convert from Tensor to list to plot using seaborn.
train_loss_for_plotting = [tensor_item.item() for tensor_item in train_loss_for_plotting]
vali_loss_for_plotting = [tensor_item.item() for tensor_item in vali_loss_for_plotting]
print(type(train_loss_for_plotting))
print(type(vali_loss_for_plotting))

# Create a numpy list for epochs which starts from 1 and ends with  1000.
Epochs = np.arange(1, number_of_epochs + 1)
type(Epochs)
```

```
<class 'list'>
<class 'list'>
```

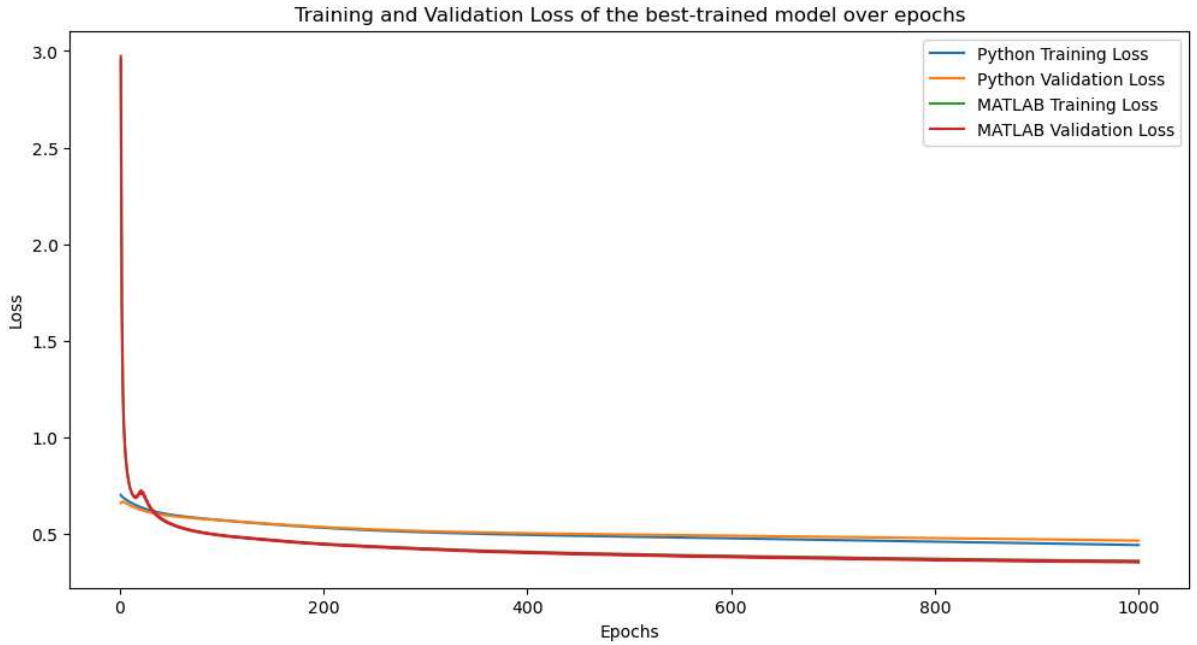
Out[40]:
```
numpy.ndarray
```

Model evaluation

In [41]:
```
# Import the recorded performance from MATLAB to plot in Python.
import csv
with open('matlab_training_time_bestmodel.csv', 'r') as file1:
    reader1 = csv.reader(file1)
    matlab_training_time_bestmodel = np.array(list(reader1), dtype=float)

with open('matlab_train_loss_for_plotting.csv', 'r') as file2:
    reader2 = csv.reader(file2)
    matlab_train_loss_for_plotting = np.array(list(reader2), dtype=float)

with open('matlab_vali_loss_for_plotting.csv', 'r') as file3:
    reader3 = csv.reader(file3)
    matlab_vali_loss_for_plotting = np.array(list(reader3), dtype=float)
```

In [42]:
```
# Change the nested list to flat lists
matlab_training_time_bestmodel = matlab_training_time_bestmodel.tolist()[0]
matlab_train_loss_for_plotting = matlab_train_loss_for_plotting.tolist()[0]
matlab_vali_loss_for_plotting = matlab_vali_loss_for_plotting.tolist()[0]
```

In [43]:
```
# Plot the training time of the model in Python and that of the model in MATLAB.
fig = plt.figure(figsize=(12, 6))
plt.plot(Epochs, train_loss_for_plotting, label='Python Training Loss')
plt.plot(Epochs, vali_loss_for_plotting, label='Python Validation Loss')
plt.plot(Epochs, matlab_train_loss_for_plotting, label='MATLAB Training Loss')
plt.plot(Epochs, matlab_vali_loss_for_plotting, label='MATLAB Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss of the best-trained model over epochs')
plt.show()
```



In [44]:
```
# Since the value in the training time is continuous, it needs to be changed cumulatively.
python_training_time_bestmodel = np.cumsum(python_training_time_bestmodel)

len(python_training_time_bestmodel)
```

Out[44]:
```
1000
```

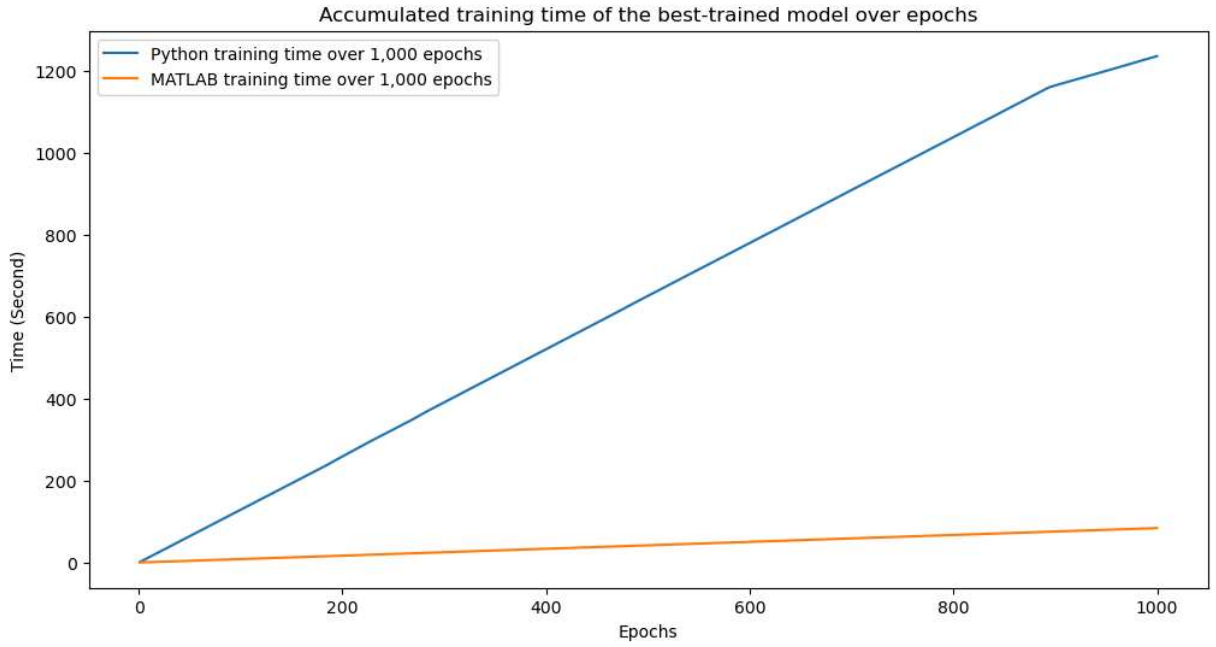
In [45]:
```
python_training_time_bestmodel
```

Out[45]:
```
array([  1.30332232,   2.58863735,   3.87094259,   5.15135574,
         6.43319321,   7.71303678,   9.00332785,  10.30966806,
        11.60496116,  12.87815285,  14.1724906 ,  15.41177082,
        16.70859504,  18.00489402,  19.30119109,  20.58648157,
        21.89680195,  23.18409252,  24.45637965,  25.72119737,
        26.98779511,  28.27513671,  29.58646774,  30.86280274,
        32.15514207,  33.44148135,  34.71978021,  36.00225878,
        37.29054976,  38.5690825 ,  39.85237122,  41.14368916,
        42.4139843 ,  43.70027518,  44.98661113,  46.24287534,
        47.52716565,  48.80245376,  50.15575957,  51.47508287,
        52.81146789,  54.11478496,  55.37011647,  56.65747929,
        57.94228029,  59.2300787 ,  60.50839996,  61.82270074,
        63.10899854,  64.41471553,  65.68045616,  66.93678594,
        68.24160409,  69.52590156,  70.81225491,  72.09055018,
        73.40192986,  74.67630863,  75.93059301,  77.22088456,
        78.5001862 ,  79.78547668,  81.07176614,  82.38207126,
        83.69441962,  85.00571513,  86.29701567,  87.54296231,
        88.83825326,  90.12305117,  91.40134001,  92.6866436 ,
        93.97793627,  95.26276374,  96.54826927,  97.85056257,
        99.10084939, 100.37458682, 101.67238784, 102.96769381,
       104.2779901 , 105.56830931, 106.83860373, 108.1238935 ,
       109.41016269, 110.69848418, 111.98577476, 113.28706884,
       114.58036542, 115.86866331, 117.15600085, 118.42128706,
       119.73564959, 121.01295686, 122.30071115, 123.58150578,
       124.8888011 , 126.19230533, 127.50460172, 128.79606557,
       130.09327245, 131.41257811, 132.69091439, 133.97425246,
       135.26254368, 136.56383824, 137.85512996, 139.15647244,
       140.45882082, 141.73312187, 143.04347181, 144.32633471,
       145.63263369, 146.90892196, 148.20422697, 149.50652981,
       150.80082703, 152.07063532, 153.32043028, 154.61472273,
       155.90901446, 157.18383217, 158.4481287 , 159.71542645,
       161.00971937, 162.29801083, 163.58629942, 164.86258841,
       166.15587926, 167.43657994, 168.71787024, 170.01116252,
       171.28846931, 172.54775667, 173.85105133, 175.14586687,
       176.41115665, 177.68544459, 178.99625921, 180.25055408,
       181.5277319 , 182.8170228 , 184.09331155, 185.37962103,
       186.66791201, 187.94725323, 189.23454332, 190.52386999,
       191.80215883, 193.09045792, 194.36785293, 195.67014813,
       196.9234302 , 198.23372602, 199.5160253 , 200.80782366,
       202.11714101, 203.39843941, 204.68028283, 205.97557545,
       207.26586795, 208.57116294, 209.88846564, 211.17775655,
       212.4800508 , 213.76034546, 215.0606401 , 216.36493444,
       217.64823723, 218.91252947, 220.19381905, 221.48711109,
       222.78540421, 224.07179308, 225.35308218, 226.64042354,
       227.9237349 , 229.24603319, 230.55632949, 231.77969694,
       233.04007506, 234.32754469, 235.63503742, 236.9457984 ,
       238.27713537, 239.64950228, 241.03983212, 242.44871664,
       243.82602572, 245.19836688, 246.59570312, 247.9100008 ,
       249.34032989, 250.72064185, 252.0949614 , 253.47027636,
       254.84558749, 256.19789386, 257.55222106, 258.90953636,
       260.20416379, 261.63868427, 262.99999905, 264.41435242,
       265.8126688 , 267.16149044, 268.56880808, 269.92815995,
       271.31147313, 272.75379896, 274.15112901, 275.51404762,
       276.82635212, 278.20568919, 279.49407363, 280.82244039,
       282.16080523, 283.51021194, 284.86052656, 286.15689278,
       287.41718268, 288.70305395, 289.99334502, 291.26466227,
       292.54910016, 293.83735847, 295.11664724, 296.39593768,
       297.69925261, 298.97204733, 300.26433921, 301.53262544,
       302.79791164, 304.090204  , 305.38649988, 306.68179321,
       307.96913958, 309.24442625, 310.54071879, 311.82156706,
       313.12586164, 314.40034175, 315.67865491, 316.96047378,
       318.27777076, 319.56207991, 320.84136868, 322.09372163,
       323.35302997, 324.6343646 , 325.90970135, 327.20099211,
       328.53639054, 329.84838033, 331.14069605, 332.42698789,
       333.70827746, 334.98461318, 336.3089602 , 337.58724904,
       338.8790946 , 340.1630497 , 341.47937036, 342.77273297,
       344.0330174 , 345.3208568 , 346.62415171, 347.93546224,
       349.29077601, 350.71109629, 352.10241079, 353.49972725,
       354.84305573, 356.26537681, 357.66769719, 359.00700593,
       360.39131927, 361.77563167, 363.24500513, 364.6253171 ,
       365.957618  , 367.37744665, 368.7697823 , 370.1150856 ,
       371.40837789, 372.70567799, 373.98796606, 375.30526471,
       376.6235621 , 377.92588878, 379.2322135 , 380.49552131,
       381.78185916, 383.07815957, 384.36745095, 385.62935352,
       386.91164327, 388.19893408, 389.50827789, 390.7975688 ,
       392.08586025, 393.35914731, 394.60643125, 395.89576983,
       397.19006968, 398.49336267, 399.78966594, 401.09999418,
       402.38728499, 403.69562793, 404.99169755, 406.29599142,
       407.58328271, 408.89258599, 410.2069478 , 411.49323821,
       412.77157497, 414.05386519, 415.35028362, 416.62089944,
       417.91719246, 419.19552684, 420.48791146, 421.7692492 ,
       423.03258157, 424.31087065, 425.59816146, 426.88795948,
       428.14835882, 429.45367718, 430.74201179, 432.02433419,
       433.30464458, 434.58798313, 435.86727238, 437.15556335,
       438.43336129, 439.7266531 , 441.01495266, 442.27228212,
       443.54057455, 444.83086562, 446.11815667, 447.40844703,
       448.68829226, 450.11061358, 451.4299736 , 452.7143085 ,
       453.99561906, 455.28391004, 456.56224108, 457.8275497 ,
       459.10284591, 460.38320088, 461.6692605 , 462.98055768,
       464.26494646, 465.54626036, 466.85252595, 468.16682363,
       469.45962286, 470.75200915, 472.05835271, 473.34168458,
       474.6430738 , 475.9398849 , 477.24419165, 478.53348303,
       479.82082057, 481.10525155, 482.40854621, 483.70183945,
       485.00913453, 486.30042696, 487.58073759, 488.87403035,
       490.18332505, 491.48068929, 492.78698325, 494.0829641,
       495.39159036, 496.68288183, 497.97317505, 499.26549077,
       500.55429745, 501.84663796, 503.18988013, 504.50621915,
       505.80351281, 507.08108711, 508.33341742, 509.60721278,
       510.86350298, 512.14979434, 513.42515588, 514.75392795,
       516.10024881, 517.40255642, 518.71137452, 520.00372577,
       521.28601789, 522.56235123, 523.84064102, 525.135957  ,
       526.39324045, 527.623564  , 528.92390561, 530.23224425,
       531.53954625, 532.93193603, 534.22574854, 535.59046721,
       536.93323922, 538.25043726, 539.57482481, 540.87316537,
       542.185462  , 543.48126268, 544.76608849, 546.13140798,
       547.40079904, 548.70739174, 550.0297153 , 551.3360579 ,
       552.62335777, 553.93770027, 555.24504352, 556.55536842,
       557.86368823, 559.19798994, 560.55882692, 561.84911752,
       563.15155053, 564.45489359, 565.75123191, 567.044554  ,
       568.33585715, 569.63409305, 570.91638303, 572.20979142,
       573.51813197, 574.80746937, 576.03877139, 577.32908463,
       578.64039731, 579.91820359, 581.17498112, 582.45028043,
       583.73162389, 585.01191306, 586.29420924, 587.58380198,
       588.86814332, 590.14543247, 591.4407692 , 592.70205402,
       593.97939038, 595.26269245, 596.54502535, 597.8243196 ,
```

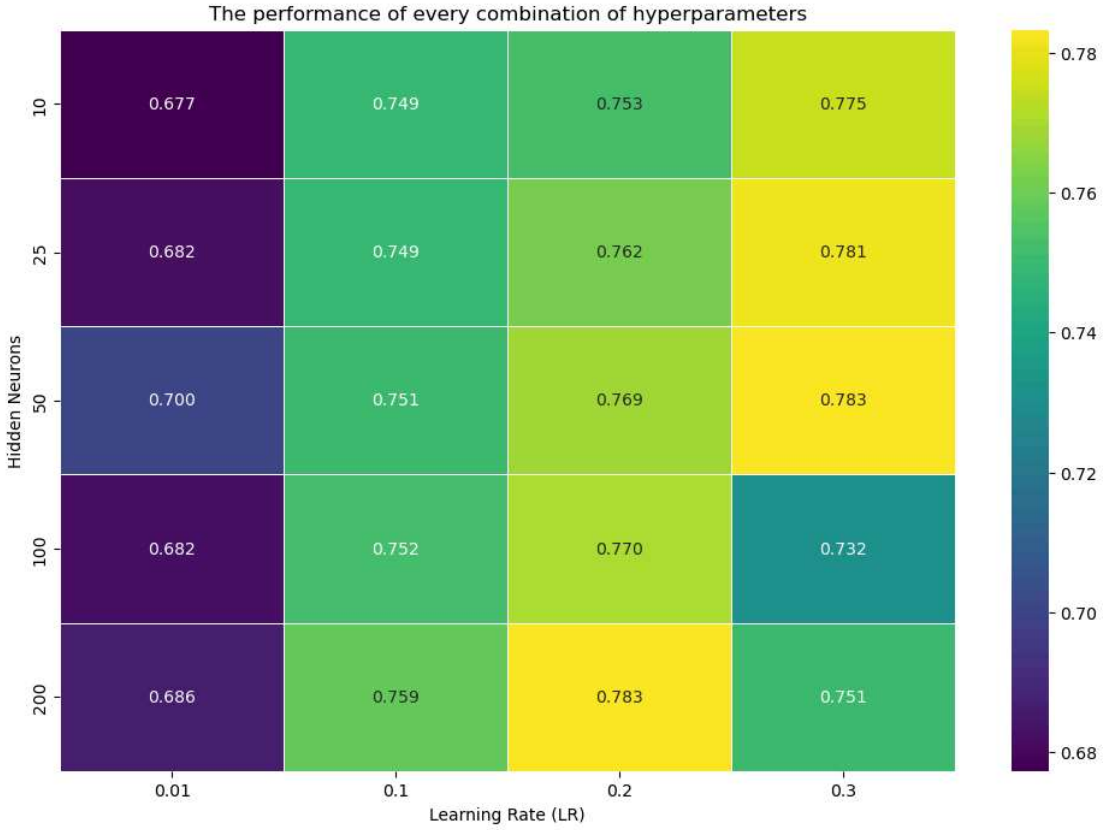
599.11479378, 600.40008354, 601.70837879, 603.00468135,
604.32697988, 605.61529136, 606.90358377, 608.22939229,
609.62370729, 610.97502422, 612.3293426 , 613.71365499,
615.05597377, 616.46129179, 617.86660957, 619.26792479,
620.55272603, 621.91303372, 623.21083403, 624.4971292 ,
625.78955841, 627.11885834, 628.41179228, 629.69708157,
631.0089047 , 632.29219556, 633.5755558 , 634.86857414,
636.15086436, 637.45166588, 638.73898387, 640.02528596,
641.31557846, 642.62689567, 643.93569708, 645.23399949,
646.51731253, 647.78759933, 649.08189964, 650.36818981,
651.62547374, 652.90127492, 654.18857408, 655.47186422,
656.75615311, 658.06144881, 659.35024548, 660.65253997,
661.94683313, 663.24313474, 664.55193806, 665.87023497,
667.16152716, 668.44682312, 669.72611237, 671.0334084 ,
672.32021523, 673.61050797, 674.90479994, 676.17708731,
677.49138355, 678.78018022, 680.06446958, 681.36926818,
682.65363622, 683.94893456, 685.22622323, 686.49852347,
687.79691648, 689.10464573, 690.43395472, 691.72280216,
693.01814938, 694.30253243, 695.6204071 , 696.89874172,
698.20714736, 699.5095017 , 700.7878027 , 702.07214069,
703.362432 , 704.64476705, 705.93506622, 707.21335506,
708.49464417, 709.78399229, 711.06728196, 712.35461998,
713.60894775, 714.92124772, 716.20893908, 717.52174354,
718.8020792 , 720.09337854, 721.37767005, 722.67000818,
723.95133185, 725.23064446, 726.53544927, 727.82478499,
729.08785009, 730.37618804, 731.64547157, 732.9369719 ,
734.24207759, 735.52940917, 736.8227191 , 738.13350081,
739.43281221, 740.72011805, 742.00672579, 743.28908134,
744.5643909 , 745.85312724, 747.15346289, 748.44777465,
749.71308446, 751.02485037, 752.30418277, 753.58849192,
754.88578439, 756.1551137 , 757.43759537, 758.71993852,
760.01933289, 761.33864212, 762.63393164, 763.94523573,
765.23052645, 766.51381636, 767.80110669, 769.0814023 ,
770.36269212, 771.64698195, 772.9353025 , 774.22368693,
775.50129652, 776.76458192, 778.04114604, 779.35344863,
780.64876318, 781.97906375, 783.26221132, 784.53152204,
785.82081795, 787.1081152 , 788.39341497, 789.68418694,
790.99153471, 792.24982738, 793.5301609 , 794.78517914,
796.06946874, 797.35075736, 798.61708951, 799.89938211,
801.2036767 , 802.49397373, 803.78029203, 805.07259393,
806.35388327, 807.63422036, 808.94651604, 810.22480583,
811.52313995, 812.82645464, 814.12375546, 815.41904807,
816.72587585, 818.0295732 , 819.33087444, 820.62144113,
821.91778469, 823.21707749, 824.5073688 , 825.79266357,
827.10296273, 828.4113028 , 829.72359896, 831.00995803,
832.30625105, 833.60554481, 834.86783743, 836.1291225 ,
837.42641449, 838.70027137, 839.99160671, 841.29591393,
842.58254504, 843.86283493, 845.15817618, 846.43446493,
847.70875311, 849.00004458, 850.28735232, 851.58369136,
852.87502027, 854.17236328, 855.45393443, 856.74022388,
858.0225122 , 859.29584932, 860.58714128, 861.87459755,
863.1589067 , 864.46620202, 865.74549031, 867.06483412,
868.357126 , 869.6394155 , 870.93270612, 872.24902964,
873.67635989, 874.92670679, 876.20599651, 877.50039721,
878.79909849, 880.08039641, 881.36169219, 882.67716694,
883.99048495, 885.28380871, 886.58610582, 887.87145042,
889.15578103, 890.43507004, 891.71124268, 893.0016067 ,
894.28630948, 895.59561038, 896.88594508, 898.17425561,
899.47459292, 900.76044941, 902.04978514, 903.32511663,
904.60742736, 905.90372562, 907.18303752, 908.46732712,
909.74763393, 911.02859116, 912.32388997, 913.60818028,
914.902493 , 916.18379235, 917.47009325, 918.75838375,
920.06167746, 921.34434223, 922.64114285, 923.96247196,
925.28881979, 926.5721221 , 927.86614919, 929.15047741,
930.45516205, 931.73950768, 933.02279758, 934.27558494,
935.52586699, 936.81515813, 938.09849691, 939.38978815,
940.69314241, 941.98443294, 943.26474833, 944.54908705,
945.83701563, 947.12631345, 948.4090426 , 949.71384621,
951.00218821, 952.26249409, 953.55481768, 954.82810497,
956.11740065, 957.42071223, 958.68801951, 959.96556497,
961.24487185, 962.5291214 , 963.82643247, 965.12476873,
966.41114736, 967.70750546, 969.00881767, 970.29267263,
971.57049608, 972.86181116, 974.14818954, 975.42252445,
976.71547818, 978.0039916 , 979.31231284, 980.59760737,
981.91896152, 983.17424607, 984.45304537, 985.74138951,
987.00472212, 988.2810092 , 989.5672996 , 990.85161996,
992.12590694, 993.41570449, 994.70399523, 995.99232697,
997.26761436, 998.55490613, 999.83924031, 1001.12152982,
1002.42983341, 1003.73712897, 1005.02746511, 1006.30682349,
1007.59616017, 1008.85290408, 1010.14322591, 1011.44252014,
1012.73685908, 1014.02714968, 1015.31444001, 1016.59872985,
1017.90707278, 1019.19344306, 1020.49573731, 1021.7870667 ,
1023.08043051, 1024.35471821, 1025.64000821, 1026.9303 ,
1028.2180984 , 1029.50543261, 1030.78872252, 1032.04400587,
1033.32329559, 1034.60359049, 1035.88492465, 1037.17330146,
1038.46859384, 1039.74510121, 1041.02895761, 1042.31331468,
1043.60069919, 1044.91509032, 1046.30151653, 1047.66994882,
1049.02315164, 1050.34150982, 1051.63786936, 1052.95318437,
1054.25054836, 1055.54391599, 1056.82724571, 1058.13055992,
1059.38986373, 1060.70718908, 1061.99549866, 1063.2838552 ,
1064.57423449, 1065.83861232, 1067.11692452, 1068.39577365,
1069.68159294, 1070.96790433, 1072.25019217, 1073.53255057,
1074.83789372, 1076.14421296, 1077.44850755, 1078.7348454 ,
1080.04639339, 1081.31668735, 1082.61598015, 1083.9144876 ,
1085.21178055, 1086.53010821, 1087.83842516, 1089.12773967,
1090.40105104, 1091.69634867, 1093.00707746, 1094.2654047 ,
1095.54569411, 1096.88702011, 1098.19732642, 1099.49366307,
1100.76997519, 1102.05328941, 1103.34362769, 1104.59995747,
1105.9033289 , 1107.18762589, 1108.47596312, 1109.75927377,
1111.02158999, 1112.28614211, 1113.58150411, 1114.87179518,
1116.18068504, 1117.46898985, 1118.7522881 , 1120.05559444,
1121.38191462, 1122.70923686, 1124.04398608, 1125.4245677 ,
1126.77488089, 1128.1242373 , 1129.46459723, 1130.72239423,
1132.02526116, 1133.33657789, 1134.62187982, 1135.904181 ,
1137.1844821 , 1138.46379852, 1139.74409962, 1141.03038907,
1142.34669924, 1143.66199589, 1144.96929026, 1146.21996737,
1147.49627924, 1148.77656817, 1150.05827594, 1151.36157012,
1152.65487146, 1153.97068572, 1155.26697707, 1156.54537249,
1157.83966589, 1159.04493737, 1159.81210995, 1160.59229255,
1161.34446216, 1162.0856297 , 1162.83480239, 1163.54496789,
1164.2461257 , 1164.9532876 , 1165.66144729, 1166.38861227,
1167.14478254, 1167.90495348, 1168.64820552, 1169.34236169,
1170.03752279, 1170.72667861, 1171.40783238, 1172.0999887 ,
1172.78414369, 1173.47630453, 1174.18446398, 1174.90263009,
1175.60678959, 1176.3349545 , 1177.04612184, 1177.75928283,

```
       1178.48144579, 1179.17760086, 1179.87875843, 1180.58291745,
       1181.26407552, 1181.97623587, 1182.66539145, 1183.40756226,
       1184.11472368, 1184.83588886, 1185.53604627, 1186.25821233,
       1186.95287204, 1187.68403745, 1188.3931973 , 1189.08636189,
       1189.81152606, 1190.52268672, 1191.22984648, 1191.93501067,
       1192.62216544, 1193.32832527, 1194.03848529, 1194.75464749,
       1195.46481156, 1196.17397523, 1196.88213491, 1197.60229826,
       1198.30546856, 1199.01062751, 1199.71678686, 1200.42294979,
       1201.12811303, 1201.82627726, 1202.53644156, 1203.23459864,
       1203.92775488, 1204.64191604, 1205.33909059, 1206.02524972,
       1206.74391437, 1207.45207453, 1208.15823436, 1208.94241214,
       1209.70058751, 1210.44075465, 1211.19392467, 1211.92408967,
       1212.67591119, 1213.4255867 , 1214.16875482, 1214.9209311 ,
       1215.68510842, 1216.47628903, 1217.22546411, 1217.97563863,
       1218.74781322, 1219.50898504, 1220.22214627, 1220.93030882,
       1221.6344676 , 1222.33762622, 1223.03678441, 1223.72994089,
       1224.43110299, 1225.12826562, 1225.84442687, 1226.54658604,
       1227.24974465, 1227.96390533, 1228.68257594, 1229.37073135,
       1230.06888795, 1230.79805231, 1231.48820829, 1232.19836926,
       1232.88952756, 1233.60468888, 1234.30984807, 1235.00400949])
```

In [46]:
```python
# Plot the loss of the model in Python and that of the model in MATLAB.
fig = plt.figure(figsize=(12, 6))
plt.plot(Epochs, python_training_time_bestmodel, label='Python training time over 1,000 epochs')
plt.plot(Epochs, matlab_training_time_bestmodel, label='MATLAB training time over 1,000 epochs')
plt.xlabel('Epochs')
plt.ylabel('Time (Second)')
plt.legend()
plt.title('Accumulated training time of the best-trained model over epochs')
plt.show()
```



In [47]:
```python
# Combine the lists into a DataFrame
df_for_heatmap = pd.DataFrame({'Learning Rate': learning_rates_list_heatmap, 'Hidden Neurons': hidden_neurons_list_heatmap, 'Accuracy': accuracy_list_heatmap})

# Make a pivot table from the DataFrame to create a matrix which is suitable for a heatmap.
pivoted_heatmap_data = df_for_heatmap.pivot('Hidden Neurons', 'Learning Rate', 'Accuracy')

# Create a heatmap using seaborn.
plt.figure(figsize=(12, 8))
sns.heatmap(pivoted_heatmap_data, annot=True, cmap='viridis', fmt=".3f", linewidths=.5)
plt.title('The performance of every combination of hyperparameters')
plt.xlabel('Learning Rate (LR)')
plt.ylabel('Hidden Neurons')
plt.show()
```



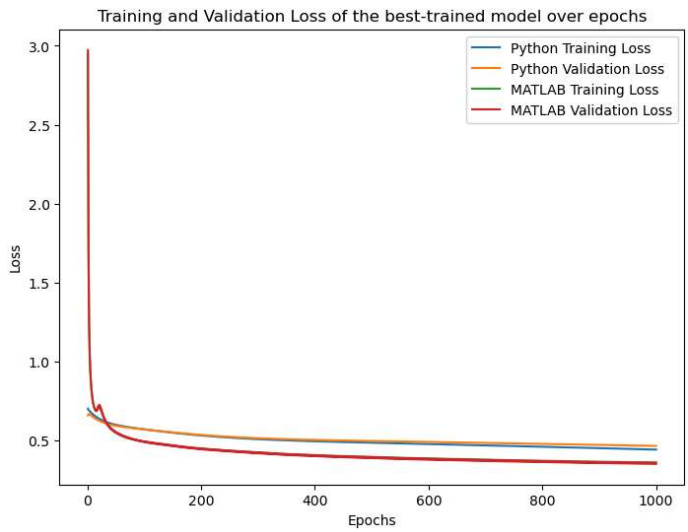
In [53]:
```
# Evaluate the model in Python with the testing set.
best_model.eval()

# Change the predictions into 0 or 1.
y_test_pred =  best_model(X_test)
y_test_pred = y_test_pred.round()

# Calculate the accuracy rate.
accuracy_rate_test_set = (y_test_pred == y_test).float().mean()
accuracy_rate_test_set = float(accuracy_rate_test_set)
accuracy_rate_test_set_python = accuracy_rate_test_set * 100
print("The test accuracy for the best-trained model in Python  =  %.2f%%" % (accuracy_rate_test_set_python))
```

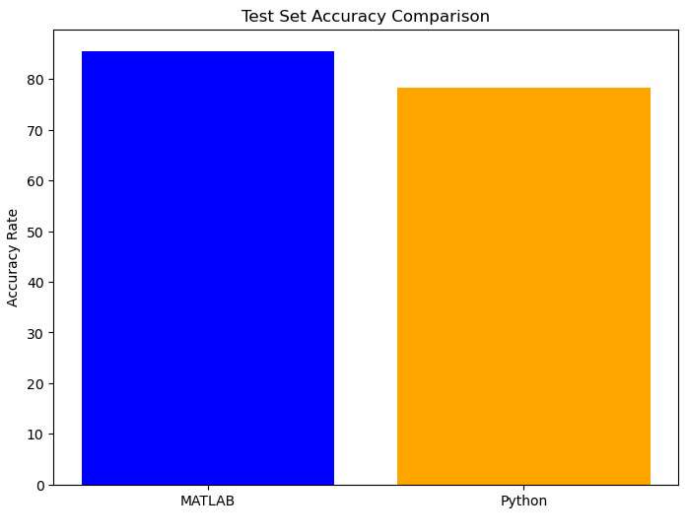
```
The test accuracy for the best-trained model in Python  =  78.22%
```

In [54]:
```
# The accuracy rate of the model in MATLAB.
# This number is derived from MATLAB.
accuracy_rate_test_set_MATLAB = 85.34
print("The test accuracy for the best-trained model in MATLAB  =  %.2f%%" % (accuracy_rate_test_set_MATLAB))
```

```
The test accuracy for the best-trained model in MATLAB  =  85.34%
```

In [50]:
```
# Make one figure for two graphs.
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))

# Line chart about the training loss from the best model in Python and MATLAB.
ax1.plot(Epochs, train_loss_for_plotting, label='Python Training Loss')
ax1.plot(Epochs, vali_loss_for_plotting, label='Python Validation Loss')
ax1.plot(Epochs, matlab_train_loss_for_plotting, label='MATLAB Training Loss')
ax1.plot(Epochs, matlab_vali_loss_for_plotting, label='MATLAB Validation Loss')
ax1.set_xlabel('Epochs')
ax1.set_ylabel('Loss')
ax1.legend()
ax1.set_title('Training and Validation Loss of the best-trained model over epochs')

# Bar graph about the accuracy from the best model in Python and MATLAB.
accuracy_labels = ['MATLAB', 'Python']
accuracy_values = [accuracy_rate_test_set_MATLAB, accuracy_rate_test_set_python]
ax2.bar(accuracy_labels, accuracy_values, color=['blue', 'orange'])
ax2.set_ylabel('Accuracy Rate')
ax2.set_title('Test Set Accuracy Comparison')

plt.show()
```





In [51]:
```
# Calculate = True Positive (TP), True Nagative (TN), False Positive (FP), False Nagative (FN)
TP = ((y_test_pred.round() == 1) & (y_test == 1)).sum().item()
TN = ((y_test_pred.round() == 0) & (y_test == 0)).sum().item()
FP = ((y_test_pred.round() == 1) & (y_test == 0)).sum().item()
FN = ((y_test_pred.round() == 0) & (y_test == 1)).sum().item()

# Calculate the confusion matrix.
# Apart from the accuracy rate, measure Precision, Recall, and F1 score.
precision = TP / (TP + FP) if (TP + FP) > 0 else 0
recall = TP / (TP + FN) if (TP + FN) > 0 else 0
f1 = 2 * (precision * recall) / (precision + recall) if (precision + recall) > 0 else 0

print("Precision of the best-trained model = %.2f%%" % (precision*100))
print("Recall of best-trained model = %.2f%%" % (recall*100))
print("F1 score of best-trained model = %.2f%%" % (f1*100))
```

```
Precision of the best-trained model = 48.78%
Recall of best-trained model = 69.30%
F1 score of best-trained model = 57.26%
```

In [ ]:

In [ ]: