In [142…

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

In [143…

```
file_path = "Churn_Modelling.csv"
df = pd.read_csv(file_path)
```

In [144…

```
df
```

Out[144…

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Ten
0	1	15634602	Hargrave	619	France	Female	42	
1	2	15647311	Hill	608	Spain	Female	41	
2	3	15619304	Onio	502	France	Female	42	
3	4	15701354	Boni	699	France	Female	39	
4	5	15737888	Mitchell	850	Spain	Female	43	
...	...	...	...	...	...	...	...	
9995	9996	15606229	Obijiaku	771	France	Male	39	
9996	9997	15569892	Johnstone	516	France	Male	35	
9997	9998	15584532	Liu	709	France	Female	36	
9998	9999	15682355	Sabbatini	772	Germany	Male	42	
9999	10000	15628319	Walker	792	France	Female	28	

10000 rows × 14 columns

In [145…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   RowNumber        10000 non-null  int64  
 1   CustomerId       10000 non-null  int64  
 2   Surname          10000 non-null  object 
 3   CreditScore      10000 non-null  int64  
 4   Geography        10000 non-null  object 
 5   Gender           10000 non-null  object 
 6   Age              10000 non-null  int64  
 7   Tenure           10000 non-null  int64  
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64  
 10  HasCrCard        10000 non-null  int64  
 11  IsActiveMember   10000 non-null  int64  
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64  
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

In [146…

```
shuffled_indices = np.random.permutation(df.index)
df_shuffled = df.loc[shuffled_indices].reset_index(drop=True)

print(df_shuffled.head())
```

```
   RowNumber  CustomerId     Surname  CreditScore Geography  Gender  Age  \
0       9722    15724876    McGregor          560    France  Female   38   
1       2895    15644119     Sochima          531    France    Male   31   
2       9866    15691950       Parry          591    France    Male   49   
3       9570    15643523       Power          710     Spain  Female   30   
4       9562    15810010  Dahlenburg          678   Germany    Male   36   

   Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  \
0       5   83714.41              1          1               1   
1       3       0.00              1          1               1   
2       3       0.00              2          1               0   
3      10       0.00              2          1               0   
4       6  118448.15              2          1               0   

   EstimatedSalary  Exited  
0         33245.97       0  
1         42589.33       0  
2         50123.44       0  
3         19500.10       0  
4         53172.02       0  
```

In [147…

```
df["Sal_bal"] = df["EstimatedSalary"] / df["Balance"]
```

In [148…

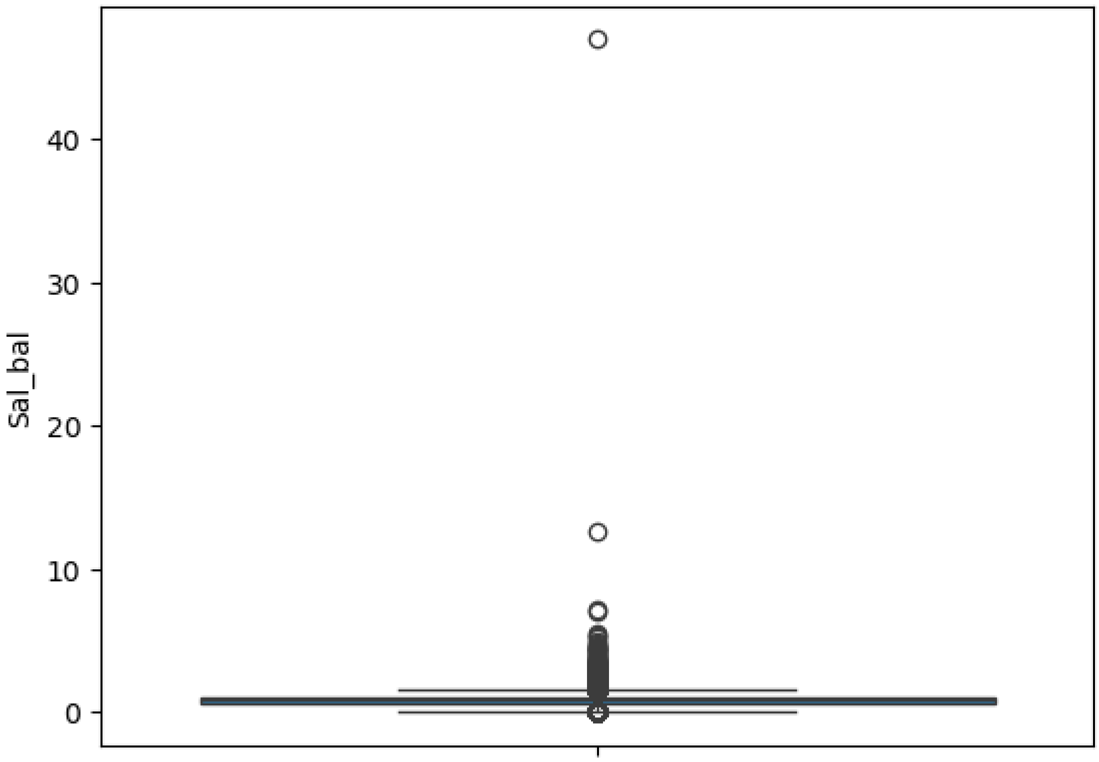
```
df.replace({"Sal_bal":[np.inf, -np.inf]}, np.nan, inplace=True)
df.fillna({"Sal_bal": df["Sal_bal"].mean()}, inplace=True)
```

In [149…

```
sns.boxplot(y=df['Sal_bal'])
```

Out[149…

```
<Axes: ylabel='Sal_bal'>
```



In [150…

```
f_df = df[df['Sal_bal'] <6]
f_df
```

Out[150…

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Ten
0	1	15634602	Hargrave	619	France	Female	42	
1	2	15647311	Hill	608	Spain	Female	41	
2	3	15619304	Onio	502	France	Female	42	
3	4	15701354	Boni	699	France	Female	39	
4	5	15737888	Mitchell	850	Spain	Female	43	
...	...	...	...	...	...	...	...	...
9995	9996	15606229	Obijiaku	771	France	Male	39	
9996	9997	15569892	Johnstone	516	France	Male	35	
9997	9998	15584532	Liu	709	France	Female	36	
9998	9999	15682355	Sabbatini	772	Germany	Male	42	
9999	10000	15628319	Walker	792	France	Female	28	

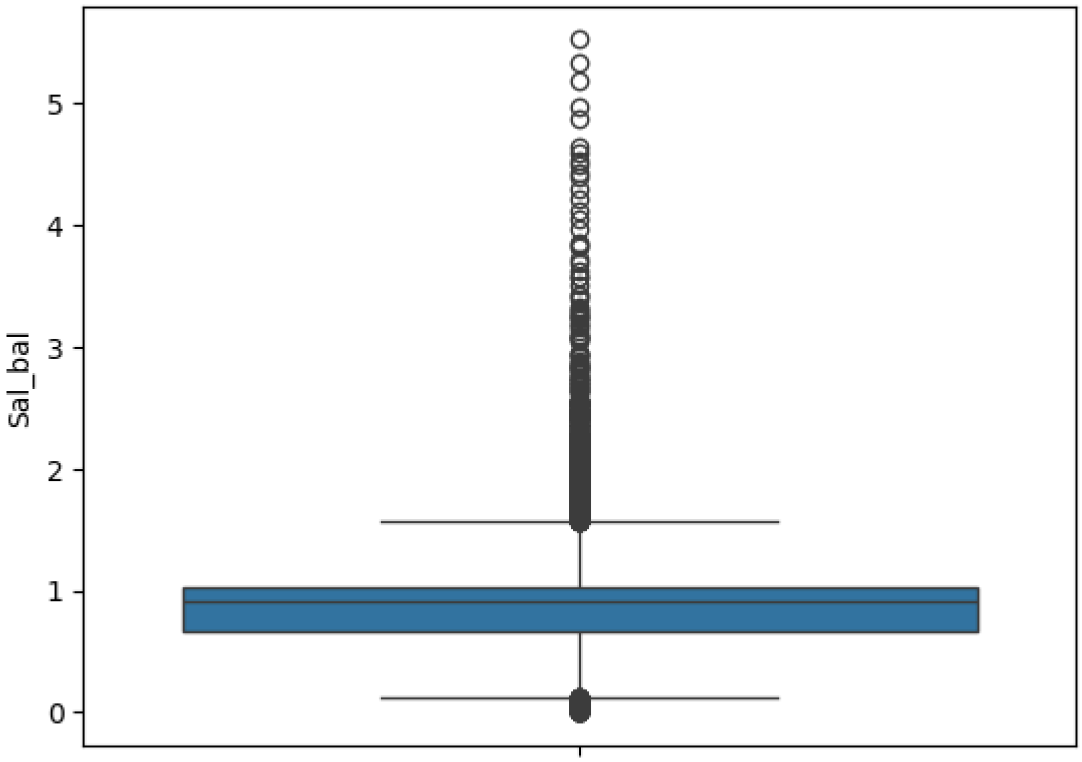
9996 rows × 15 columns

In [151…

```
sns.boxplot(y=f_df['Sal_bal'])
```

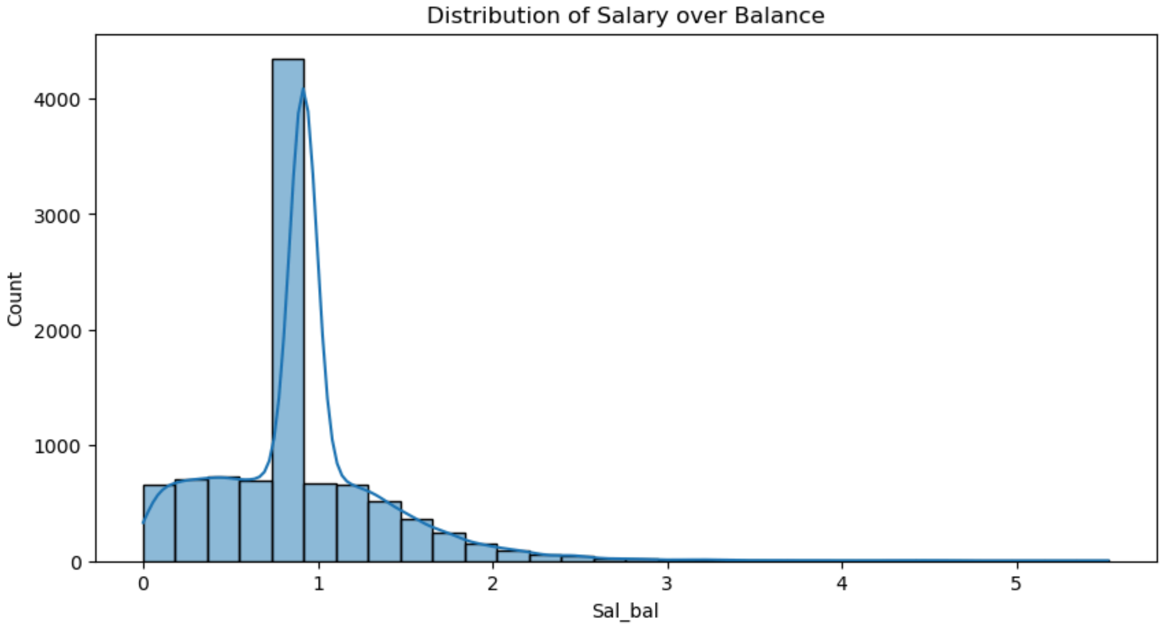
Out[151…

```
<Axes: ylabel='Sal_bal'>
```



In [152…

```
plt.figure(figsize=(10, 5))
sns.histplot(f_df["Sal_bal"], bins=30, kde=True) #Kernel Density Estimation
plt.title("Distribution of Salary over Balance")
plt.xlabel("Sal_bal")
plt.ylabel("Count")
plt.show()
```



In [153…

```
X =f_df[["Sal_bal","CreditScore"]]
y = f_df["HasCrCard"]
```

In [154…

```
dsize = int(0.8 * len(f_df))
X_train = X[:dsize]
y_train = y[:dsize]
```

In [155…

```
X_test = X[dsize:]
y_test = y[dsize:]
```

In [156…

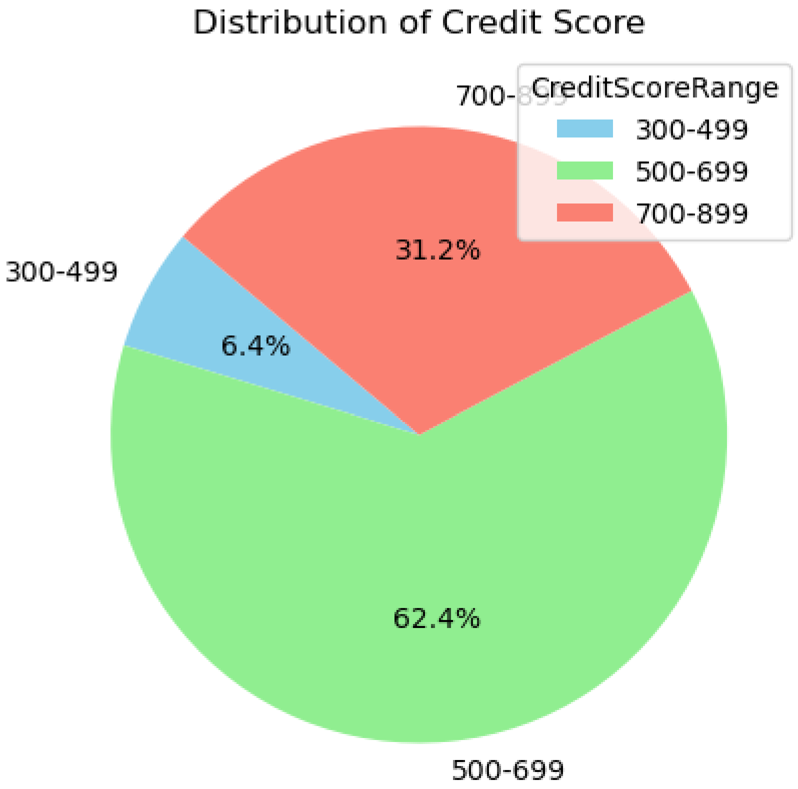
```
y_train
```

Out[156…

```
0       1
1       0
2       1
3       0
4       1
       ..
7994    1
7995    0
7996    1
7997    1
7998    1
Name: HasCrCard, Length: 7996, dtype: int64
```

In [157…

```
bins = [300, 500, 700, 900]
labels = ['300-499', '500-699', '700-899']

df['CreditScoreRange'] = pd.cut(f_df['CreditScore'], bins=bins, labels=labels)

range_counts = df['CreditScoreRange'].value_counts(sort=False)

plt.figure(figsize=(6, 5))
plt.pie(range_counts, labels=range_counts.index, autopct='%1.1f%%', startangle=1
plt.legend(range_counts.index, title="CreditScoreRange", loc="upper right")
plt.title("Distribution of Credit Score")
plt.show()
```



In [158…

```
from matplotlib.colors import ListedColormap
cmap_bold = ListedColormap(["#FF0000","#00FF00","#0000FF"])
```

```
cmap_light = ListedColormap(["#FFBBBB", "#BBFFBB","#BBBBFF"])
```

In [159…

```
# Activations

def linear(H):
  return H

def ReLU(H):
  return H*(H>0)

def sigmoid(H):
  return 1/(1+np.exp(-H))

def softmax(H):
  eH=np.exp(H)
  return eH/eH.sum(axis=1, keepdims=True)

#Loss Functions

def cross_entropy(Y, P_hat):
  return -(1/len(Y))*np.sum(Y*np.log(P_hat))

def OLS(Y, Y_hat):
  return (1/(2*len(Y)))*np.sum((Y-Y_hat)**2)

#Misc

def one_hot(y):
  N=len(y)
  K=len(set(y))
  Y = np.zeros((N,K))

  for i in range(N):
      if y[i] < K:
          Y[i,y[i]]=1

  return Y

def accuracy(y,y_hat):
  return np.mean(y==y_hat)

def R2(y,y_hat):
  return 1-np.sum((y-y_hat)**2)/np.sum((y - y.mean())**2)
```

In [160…

```
def derivative(Z, a):
  if a==linear:
    return 1
  elif a==sigmoid:
    return Z*(1-Z)
  elif a==np.tanh:
    return 1-Z*Z
  elif a==ReLU:
    return (Z>0).astype(int)
  else:
    ValueError("UnknownActivation")
```

In [161…

```
class ANN():
  def __init__(self, architecture, activations=None, mode=0):
    self.mode=mode
    self.architecture=architecture
    self.activations = activations
    self.L = len(architecture)+1

  def fit (self, X, y, eta=1e-3, epochs=1e3, show_curve=True):
    epochs = int(epochs)

    if self.mode:
      Y=y
      K=1
    else:
      Y=one_hot(y)
      K=Y.shape[1]

    N, D = X.shape
    # Initialize Weights and Biases
    self.W = {l: np.random.randn(M[0],M[1]) for l, M in enumerate(zip(([D]+self.
    self.B = {l: np.random.randn(M) for l, M in enumerate(self.architecture+[K],


    #Activations
    if self.activations is None:
      self.a= {l: ReLU for l in range(1, self.L)}
    else:
      self.a={l: act for l,act in enumerate(self.activations, 1)}

    #Output Activation Functions
    if self.mode:
      self.a[self.L]=linear
    else:
      self.a[self.L]=softmax

    J = np.zeros(epochs)

    #SGD Progression
    for epoch in range(epochs):
      self.__forward__(X)
      if self.mode:
        J[epoch]=OLS(Y, self.Z[self.L])
      else:
        J[epoch]=cross_entropy(Y, self.Z[self.L])

      dH = (1/N)*(self.Z[self.L]-Y)

      for l in sorted(self.W.keys(), reverse=True):

        dW = self.Z[l-1].T@dH
        dB = dH.sum(axis=0)

        self.W[l] -= eta*dW
        self.B[l] -= eta*dB

        if l>1:
          dZ =dH@self.W[l].T
          dH = dZ*derivative(self.Z[l-1], self.a[l-1])

    if show_curve:
      plt.figure()
      plt.plot(J)
      plt.xlabel("epochs")
      plt.ylabel("J")
      plt.title("Training Curve")

  def __forward__(self,X):
    self.Z={0:X}
    for l in sorted(self.W.keys()):
      self.Z[l] = self.a[l](self.Z[l-1]@self.W[l]+self.B[l])


  def predict(self, X):
    self.__forward__(X)
    if self.mode:
      return self.Z[self.L]
    else:
      return self.Z[self.L].argmax(axis=1)
```

In [162…

```
def main_class():
    D = 2
    K = 3
    N = int(K*1e3)

    X0 = np.random.randn((N//K),D) + np.array([2,2])
    X1 = np.random.randn((N//K),D) + np.array([0,-2])
    X2 = np.random.randn((N//K),D) + np.array([-2,2])
    X = np.vstack((X0,X1,X2))

    y = np.array([0]*(N//K) + [1]*(N//K) + [2]*(N//K))

    plt.figure()
    plt.scatter(X[:,0],X[:,1], c=y, s=6, alpha=0.6)

    my_ann_classifier = ANN(architecture=[10,7,6],activations=[np.tanh,ReLU,ReLU
    my_ann_classifier.fit(X,y,eta=5e-3,epochs=1e3)#eta=2e-3,epochs=1e4)
    y_hat = my_ann_classifier.predict(X)



    print(my_ann_classifier.W)
    print(my_ann_classifier.B)
    print(f"Training Accuracy: {accuracy(y,y_hat):0.4f}")

    x1 = np.linspace(X[:,0].min() - 1, X[:,0].max() + 1, 1000)
    x2 = np.linspace(X[:,1].min() - 1, X[:,1].max() + 1, 1000)

    xx1, xx2 = np.meshgrid(x1, x2)
    Z = my_ann_classifier.predict(np.c_[xx1.ravel(),xx2.ravel()]).reshape(*xx1.s

    plt.figure()
    plt.pcolormesh(xx1, xx2, Z, cmap = cmap_light)
    plt.scatter(X[:,0], X[:,1], c = y, cmap = cmap_bold,alpha=0.2)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())
    plt.show()

    plt.figure()
    plt.scatter(X[:,0],X[:,1],c=y_hat,s=6)
```

In [163…

```
X_train = pd.concat([X_train, X_train.iloc[:4]], ignore_index=True)
```

In [183…

```
y_train = pd.concat([y_train, y_train.iloc[:4]], ignore_index=True)
```

In [193…

```
y_train.head()
```

Out[193…

```
0    1
1    0
2    1
3    0
4    1
Name: HasCrCard, dtype: int64
```

In [184…

```
X_train.shape
```

Out[184…

```
(8000, 2)
```

In [185…

```
y_train.shape
```

Out[185…

```
(8000,)
```

In [186…

```
np.unique(y_train)
```

Out[186…

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

In [187…

```
X_min = np.min(X_train, axis=0)  # Minimum value for each feature
X_max = np.max(X_train, axis=0)  # Maximum value for each feature

X_scaled = (X_train - X_min) / (X_max - X_min)  # Apply scaling
print(X_scaled)
```

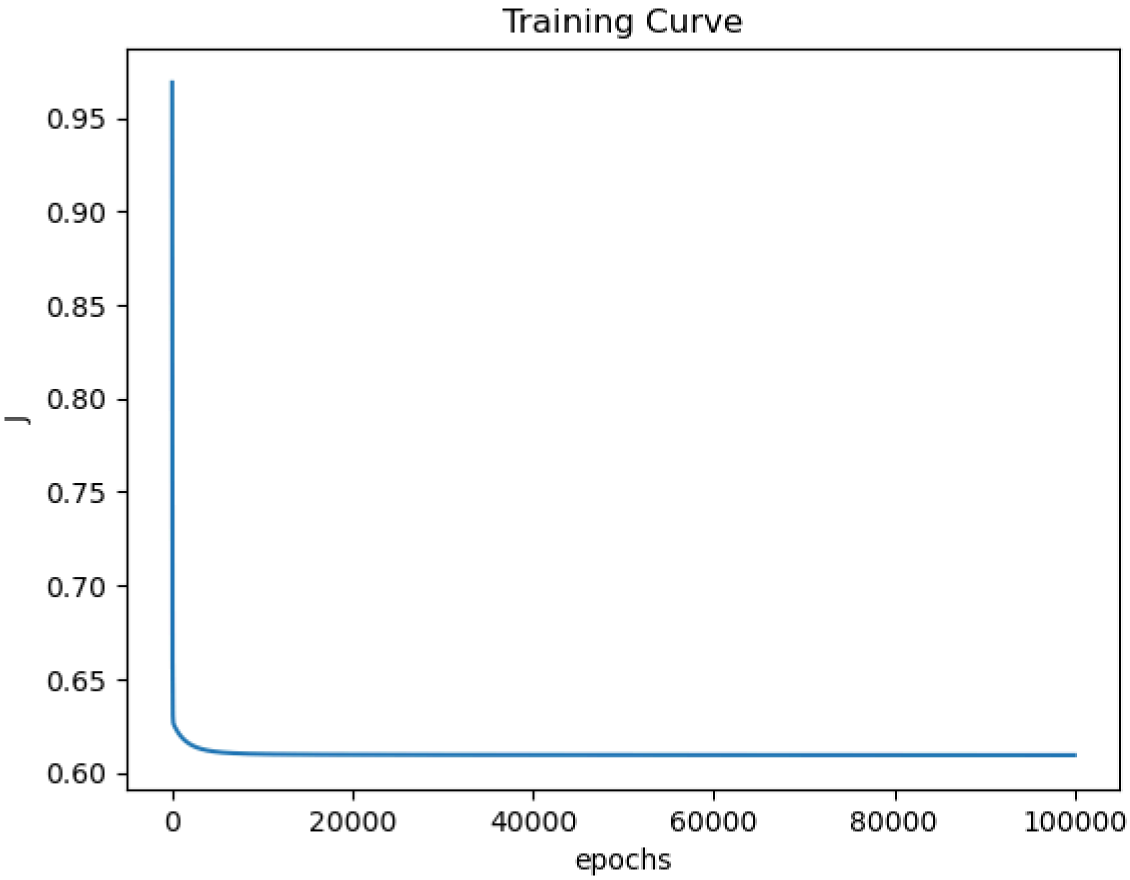
```
       Sal_bal  CreditScore
0     0.166032        0.538
1     0.243041        0.516
2     0.129142        0.304
3     0.166032        0.698
4     0.114030        1.000
...        ...          ...
7995  0.143367        0.958
7996  0.166032        0.538
7997  0.243041        0.516
7998  0.129142        0.304
7999  0.166032        0.698

[8000 rows x 2 columns]
```

In [196…

```
my_ann_classifier = ANN(architecture=[6, 4],activations=[np.tanh,ReLU])#architec
```

In [218…

```
try:
    my_ann_classifier.fit(X_scaled.values,y_train.values,eta=3e-3,epochs=1e5)#et
except Exception as e:
    print(f"Error during training: {e}")
```



In [219…

```
y_hat = my_ann_classifier.predict(X_test.values)
```

In [220…

```
print(f"Training Accuracy: {accuracy(y_test.values,y_hat):0.4f}")
```

```
Training Accuracy: 0.7205
```

In [ ]: