1. Load and Prepare Data

In [2]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import brier_score_loss

# Load the training data
train_data = pd.read_csv('train.csv')

# Explore the data
print(train_data.head())

# Encode categorical variables using one-hot encoding
categorical_cols = ['Education', 'EmploymentType', 'MaritalStatus', 'HasMort
train_data_encoded = pd.get_dummies(train_data, columns=categorical_cols)

# Separate features and target
X = train_data_encoded.drop(['ID', 'Default'], axis=1)
y = train_data_encoded['Default']

# Preprocessing: Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_scaled, y, test_size=0.2

# Initialize and train the logistic regression model
model = LogisticRegression(random_state=42)
model.fit(X_train, y_train)

# Predict probabilities on the validation set
probabilities = model.predict_proba(X_val)[:, 1]  # get the probability of t

# Evaluate the model using Brier score loss
brier_score = brier_score_loss(y_val, probabilities)
print(f"Brier score loss: {brier_score}")
```

```
   ID  Age  Income  LoanAmount  CreditScore  MonthsEmployed  NumCreditLines
0   0   21   78304      168713          653              60               1
\
1   1   28   63751       84674          681              58               1
2   2   57   96676      167540          467              98               4
3   3   24   79289       61546          358              63               4
4   4   31   98586      232342          692              10               2

   InterestRate  LoanTerm  DTIRatio    Education EmploymentType MaritalStat
us
0          8.80        60      0.59  High School      Part-time        Sing
le  \
1          4.91        48      0.21          PhD      Part-time       Marri
ed
2         16.78        36      0.63  High School     Unemployed        Sing
le
3          6.40        60      0.83     Master's      Full-time        Sing
le
4         19.97        60      0.51          PhD     Unemployed       Marri
ed

  HasMortgage HasDependents LoanPurpose HasCoSigner  Default
0          No           Yes        Home         Yes        0
1         Yes           Yes        Auto          No        0
2          No           Yes    Business         Yes        0
3         Yes           Yes    Business         Yes        0
4         Yes           Yes   Education         Yes        0
Brier score loss: 0.11529765418234914
```

2. data processing

In [10]:

```
# Assuming your preprocessing and model training steps are satisfactory and

# Load the test data
test_data = pd.read_csv('test.csv')

# Apply the same preprocessing to the test data
test_data_encoded = pd.get_dummies(test_data, columns=categorical_cols)
# Ensure the test data has the same features as the training data, filling m
test_data_encoded = test_data_encoded.reindex(columns=X.columns, fill_value=

# Scale the test data using the same scaler used for the train data
X_test_scaled = scaler.transform(test_data_encoded)

# Predict probabilities with the trained model
test_probabilities = model.predict_proba(X_test_scaled)[:, 1]

# Create submission DataFrame
submission = pd.DataFrame({
    'ID': test_data['ID'],
    'TARGET': test_probabilities
})

# Save the submission file
```

```
submission.to_csv('submission.csv', index=False)
print("Submission file created.")
```

```
Submission file created.
```

Version 2

In [3]:
```
import tensorflow as tf
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.metrics import BinaryAccuracy
from tensorflow.keras.optimizers import Adam
import joblib

X_train, X_val, y_train, y_val = train_test_split(X_scaled, y, test_size=0.2

# Model configuration
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')  # Output layer with sigmoid activation f
])

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='binary_crossentropy',
              metrics=[BinaryAccuracy(name='accuracy'), tf.keras.metrics.AUC

# Model summary
model.summary()

# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_d

# Evaluate the model on the validation set
val_loss, val_accuracy, val_auc = model.evaluate(X_val, y_val, verbose=0)
print(f'Validation Loss: {val_loss}')
print(f'Validation Accuracy: {val_accuracy}')
print(f'Validation AUC: {val_auc}')

model.save('model_1.h5')
joblib.dump(scaler, 'my_scaler.gz')
```

```
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` 
runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, l
ocated at `tf.keras.optimizers.legacy.Adam`.
```

```
Metal device set to: Apple M1

systemMemory: 8.00 GB
maxCacheSize: 2.67 GB

WARNING:absl:There is a known slowdown when using v2.11+ Keras optimizers o
n M1/M2 Macs. Falling back to the legacy Keras optimizer, i.e., `tf.keras.o
ptimizers.legacy.Adam`.
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               4096

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 64)                8256

 dropout_1 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 32)                2080

 dropout_2 (Dropout)         (None, 32)                0

 dense_3 (Dense)             (None, 1)                 33

=================================================================
Total params: 14,465
Trainable params: 14,465
Non-trainable params: 0
_________________________________________________________________
Epoch 1/50
2024-04-29 15:07:48.324738: W tensorflow/tsl/platform/profile_utils/cpu_uti
ls.cc:128] Failed to get CPU frequency: 0 Hz
```

```
3750/3750 [==============================] - 44s 11ms/step - loss: 0.3899 -
accuracy: 0.8466 - auc: 0.7231 - val_loss: 0.3767 - val_accuracy: 0.8500 -
val_auc: 0.7446
Epoch 2/50
3750/3750 [==============================] - 45s 12ms/step - loss: 0.3817 -
accuracy: 0.8487 - auc: 0.7389 - val_loss: 0.3753 - val_accuracy: 0.8502 -
val_auc: 0.7484
Epoch 3/50
3750/3750 [==============================] - 43s 11ms/step - loss: 0.3793 -
accuracy: 0.8492 - auc: 0.7439 - val_loss: 0.3738 - val_accuracy: 0.8507 -
val_auc: 0.7526
Epoch 4/50
3750/3750 [==============================] - 43s 11ms/step - loss: 0.3770 -
accuracy: 0.8505 - auc: 0.7479 - val_loss: 0.3741 - val_accuracy: 0.8509 -
val_auc: 0.7509
Epoch 5/50
3750/3750 [==============================] - 44s 12ms/step - loss: 0.3764 -
accuracy: 0.8502 - auc: 0.7492 - val_loss: 0.3728 - val_accuracy: 0.8508 -
val_auc: 0.7534
Epoch 6/50
3750/3750 [==============================] - 43s 12ms/step - loss: 0.3750 -
accuracy: 0.8512 - auc: 0.7514 - val_loss: 0.3719 - val_accuracy: 0.8508 -
val_auc: 0.7550
Epoch 7/50
3750/3750 [==============================] - 43s 12ms/step - loss: 0.3742 -
accuracy: 0.8513 - auc: 0.7535 - val_loss: 0.3725 - val_accuracy: 0.8513 -
val_auc: 0.7542
Epoch 8/50
3750/3750 [==============================] - 43s 11ms/step - loss: 0.3734 -
accuracy: 0.8512 - auc: 0.7543 - val_loss: 0.3778 - val_accuracy: 0.8498 -
val_auc: 0.7542
Epoch 9/50
3750/3750 [==============================] - 43s 11ms/step - loss: 0.3729 -
accuracy: 0.8519 - auc: 0.7547 - val_loss: 0.3751 - val_accuracy: 0.8512 -
val_auc: 0.7548
Epoch 10/50
3750/3750 [==============================] - 43s 12ms/step - loss: 0.3723 -
accuracy: 0.8518 - auc: 0.7567 - val_loss: 0.3733 - val_accuracy: 0.8508 -
val_auc: 0.7539
Epoch 11/50
3750/3750 [==============================] - 43s 11ms/step - loss: 0.3716 -
accuracy: 0.8526 - auc: 0.7577 - val_loss: 0.3715 - val_accuracy: 0.8500 -
val_auc: 0.7556
Epoch 12/50
3750/3750 [==============================] - 43s 11ms/step - loss: 0.3712 -
accuracy: 0.8521 - auc: 0.7584 - val_loss: 0.3723 - val_accuracy: 0.8512 -
val_auc: 0.7533
Epoch 13/50
3750/3750 [==============================] - 43s 12ms/step - loss: 0.3708 -
accuracy: 0.8524 - auc: 0.7588 - val_loss: 0.3721 - val_accuracy: 0.8508 -
val_auc: 0.7561
Epoch 14/50
3750/3750 [==============================] - 44s 12ms/step - loss: 0.3702 -
accuracy: 0.8526 - auc: 0.7603 - val_loss: 0.3715 - val_accuracy: 0.8518 -
val_auc: 0.7565
Epoch 15/50
```

```
3750/3750 [==============================] - 43s 11ms/step - loss: 0.3697 -
accuracy: 0.8530 - auc: 0.7607 - val_loss: 0.3718 - val_accuracy: 0.8515 -
val_auc: 0.7557
Epoch 16/50
3750/3750 [==============================] - 43s 11ms/step - loss: 0.3692 -
accuracy: 0.8534 - auc: 0.7613 - val_loss: 0.3732 - val_accuracy: 0.8507 -
val_auc: 0.7549
Epoch 17/50
3750/3750 [==============================] - 45s 12ms/step - loss: 0.3687 -
accuracy: 0.8529 - auc: 0.7628 - val_loss: 0.3714 - val_accuracy: 0.8509 -
val_auc: 0.7549
Epoch 18/50
3750/3750 [==============================] - 44s 12ms/step - loss: 0.3681 -
accuracy: 0.8537 - auc: 0.7636 - val_loss: 0.3706 - val_accuracy: 0.8516 -
val_auc: 0.7563
Epoch 19/50
3750/3750 [==============================] - 45s 12ms/step - loss: 0.3672 -
accuracy: 0.8536 - auc: 0.7650 - val_loss: 0.3716 - val_accuracy: 0.8512 -
val_auc: 0.7551
Epoch 20/50
3750/3750 [==============================] - 43s 11ms/step - loss: 0.3669 -
accuracy: 0.8541 - auc: 0.7653 - val_loss: 0.3717 - val_accuracy: 0.8503 -
val_auc: 0.7563
Epoch 21/50
3750/3750 [==============================] - 43s 12ms/step - loss: 0.3663 -
accuracy: 0.8547 - auc: 0.7654 - val_loss: 0.3710 - val_accuracy: 0.8507 -
val_auc: 0.7559
Epoch 22/50
3750/3750 [==============================] - 45s 12ms/step - loss: 0.3665 -
accuracy: 0.8544 - auc: 0.7656 - val_loss: 0.3722 - val_accuracy: 0.8520 -
val_auc: 0.7546
Epoch 23/50
3750/3750 [==============================] - 44s 12ms/step - loss: 0.3659 -
accuracy: 0.8547 - auc: 0.7667 - val_loss: 0.3727 - val_accuracy: 0.8502 -
val_auc: 0.7550
Epoch 24/50
3750/3750 [==============================] - 43s 12ms/step - loss: 0.3661 -
accuracy: 0.8542 - auc: 0.7669 - val_loss: 0.3716 - val_accuracy: 0.8508 -
val_auc: 0.7554
Epoch 25/50
3750/3750 [==============================] - 43s 12ms/step - loss: 0.3650 -
accuracy: 0.8547 - auc: 0.7686 - val_loss: 0.3760 - val_accuracy: 0.8507 -
val_auc: 0.7530
Epoch 26/50
3750/3750 [==============================] - 43s 12ms/step - loss: 0.3649 -
accuracy: 0.8546 - auc: 0.7689 - val_loss: 0.3744 - val_accuracy: 0.8508 -
val_auc: 0.7542
Epoch 27/50
3750/3750 [==============================] - 43s 12ms/step - loss: 0.3641 -
accuracy: 0.8546 - auc: 0.7703 - val_loss: 0.3729 - val_accuracy: 0.8509 -
val_auc: 0.7537
Epoch 28/50
3750/3750 [==============================] - 43s 12ms/step - loss: 0.3643 -
accuracy: 0.8549 - auc: 0.7699 - val_loss: 0.3729 - val_accuracy: 0.8502 -
val_auc: 0.7538
Epoch 29/50
```

```
3750/3750 [==============================] - 45s 12ms/step - loss: 0.3633 - accuracy: 0.8556 - auc: 0.7712 - val_loss: 0.3770 - val_accuracy: 0.8509 - val_auc: 0.7516
Epoch 30/50
3750/3750 [==============================] - 45s 12ms/step - loss: 0.3637 - accuracy: 0.8555 - auc: 0.7703 - val_loss: 0.3735 - val_accuracy: 0.8503 - val_auc: 0.7546
Epoch 31/50
3750/3750 [==============================] - 45s 12ms/step - loss: 0.3627 - accuracy: 0.8559 - auc: 0.7726 - val_loss: 0.3739 - val_accuracy: 0.8509 - val_auc: 0.7553
Epoch 32/50
3750/3750 [==============================] - 44s 12ms/step - loss: 0.3629 - accuracy: 0.8555 - auc: 0.7721 - val_loss: 0.3736 - val_accuracy: 0.8497 - val_auc: 0.7532
Epoch 33/50
3750/3750 [==============================] - 45s 12ms/step - loss: 0.3626 - accuracy: 0.8560 - auc: 0.7723 - val_loss: 0.3743 - val_accuracy: 0.8504 - val_auc: 0.7517
Epoch 34/50
3750/3750 [==============================] - 45s 12ms/step - loss: 0.3617 - accuracy: 0.8565 - auc: 0.7740 - val_loss: 0.3750 - val_accuracy: 0.8504 - val_auc: 0.7508
Epoch 35/50
3750/3750 [==============================] - 45s 12ms/step - loss: 0.3613 - accuracy: 0.8560 - auc: 0.7746 - val_loss: 0.3738 - val_accuracy: 0.8484 - val_auc: 0.7520
Epoch 36/50
3750/3750 [==============================] - 47s 13ms/step - loss: 0.3607 - accuracy: 0.8569 - auc: 0.7753 - val_loss: 0.3734 - val_accuracy: 0.8505 - val_auc: 0.7522
Epoch 37/50
3750/3750 [==============================] - 46s 12ms/step - loss: 0.3610 - accuracy: 0.8559 - auc: 0.7759 - val_loss: 0.3747 - val_accuracy: 0.8489 - val_auc: 0.7515
Epoch 38/50
3750/3750 [==============================] - 45s 12ms/step - loss: 0.3609 - accuracy: 0.8565 - auc: 0.7753 - val_loss: 0.3761 - val_accuracy: 0.8505 - val_auc: 0.7525
Epoch 39/50
3750/3750 [==============================] - 45s 12ms/step - loss: 0.3608 - accuracy: 0.8568 - auc: 0.7751 - val_loss: 0.3739 - val_accuracy: 0.8509 - val_auc: 0.7516
Epoch 40/50
3750/3750 [==============================] - 46s 12ms/step - loss: 0.3605 - accuracy: 0.8570 - auc: 0.7756 - val_loss: 0.3762 - val_accuracy: 0.8493 - val_auc: 0.7535
Epoch 41/50
3750/3750 [==============================] - 45s 12ms/step - loss: 0.3598 - accuracy: 0.8569 - auc: 0.7769 - val_loss: 0.3753 - val_accuracy: 0.8491 - val_auc: 0.7496
Epoch 42/50
3750/3750 [==============================] - 46s 12ms/step - loss: 0.3596 - accuracy: 0.8575 - auc: 0.7770 - val_loss: 0.3761 - val_accuracy: 0.8490 - val_auc: 0.7514
Epoch 43/50
```

```
3750/3750 [==============================] - 43s 12ms/step - loss: 0.3596 -
accuracy: 0.8575 - auc: 0.7772 - val_loss: 0.3737 - val_accuracy: 0.8498 -
val_auc: 0.7529
Epoch 44/50
3750/3750 [==============================] - 44s 12ms/step - loss: 0.3586 -
accuracy: 0.8572 - auc: 0.7784 - val_loss: 0.3751 - val_accuracy: 0.8497 -
val_auc: 0.7505
Epoch 45/50
3750/3750 [==============================] - 44s 12ms/step - loss: 0.3585 -
accuracy: 0.8584 - auc: 0.7786 - val_loss: 0.3746 - val_accuracy: 0.8503 -
val_auc: 0.7509
Epoch 46/50
3750/3750 [==============================] - 45s 12ms/step - loss: 0.3579 -
accuracy: 0.8579 - auc: 0.7797 - val_loss: 0.3747 - val_accuracy: 0.8497 -
val_auc: 0.7490
Epoch 47/50
3750/3750 [==============================] - 45s 12ms/step - loss: 0.3585 -
accuracy: 0.8575 - auc: 0.7796 - val_loss: 0.3767 - val_accuracy: 0.8492 -
val_auc: 0.7487
Epoch 48/50
3750/3750 [==============================] - 45s 12ms/step - loss: 0.3583 -
accuracy: 0.8574 - auc: 0.7796 - val_loss: 0.3757 - val_accuracy: 0.8496 -
val_auc: 0.7509
Epoch 49/50
3750/3750 [==============================] - 45s 12ms/step - loss: 0.3575 -
accuracy: 0.8578 - auc: 0.7808 - val_loss: 0.3746 - val_accuracy: 0.8499 -
val_auc: 0.7500
Epoch 50/50
3750/3750 [==============================] - 44s 12ms/step - loss: 0.3577 -
accuracy: 0.8576 - auc: 0.7807 - val_loss: 0.3760 - val_accuracy: 0.8485 -
val_auc: 0.7496
Validation Loss: 0.37599506974220276
Validation Accuracy: 0.8485333323478699
Validation AUC: 0.7496305108070374
```

Out[3]: ['my_scaler.gz']

In [6]:

```
# Load and preprocess the training data
train_data = pd.read_csv('train.csv')
categorical_cols = ['Education', 'EmploymentType', 'MaritalStatus', 'HasMort
train_data_encoded = pd.get_dummies(train_data, columns=categorical_cols)

# Separate features and target
X = train_data_encoded.drop(['ID', 'Default'], axis=1)
y = train_data_encoded['Default']

# Keep the feature names after encoding for later use
feature_names = X.columns

# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_scaled, y, test_size=0.2
```

```
# Load model and scaler
model = tf.keras.models.load_model('model_1.h5')
scaler = joblib.load('my_scaler.gz')

# Load and prepare the test data
test_data = pd.read_csv('test.csv')
test_data_encoded = pd.get_dummies(test_data, columns=categorical_cols)

# Ensure the test data has the same features as the training data, filling m
test_data_encoded = test_data_encoded.reindex(columns=feature_names, fill_va

# Scale the test data using the same scaler used for the train data
X_test_scaled = scaler.transform(test_data_encoded)

# Predicting probabilities on the test set
test_probabilities = model.predict(X_test_scaled).flatten()

# Creating a submission DataFrame
submission = pd.DataFrame({
    'ID': test_data['ID'],
    'TARGET': test_probabilities
})

# Save the submission file
submission.to_csv('neural_network_submission.csv', index=False)
print("Neural network submission file created.")
```

```
1563/1563 [==============================] - 3s 2ms/step
Neural network submission file created.
```

In [2]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import brier_score_loss
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder

# Load the training data
train_data = pd.read_csv('train.csv')

# Identify categorical and numerical columns
categorical_cols = ['Education', 'EmploymentType', 'MaritalStatus', 'HasMort
numerical_cols = ['Age', 'Income', 'LoanAmount', 'CreditScore', 'MonthsEmplo

# Define the preprocessing steps
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols)
    ])

# Append classifier to preprocessing pipeline
clf = Pipeline(steps=[('preprocessor', preprocessor),
                      ('classifier', RandomForestClassifier(n_estimators=100
```

```
# Separate features and target
X = train_data.drop(['ID', 'Default'], axis=1)
y = train_data['Default']

# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, rando

# Train the Random Forest model
clf.fit(X_train, y_train)

# Predict probabilities on the validation set
rf_probabilities = clf.predict_proba(X_val)[:, 1]

# Evaluate the model using Brier score loss
rf_brier_score = brier_score_loss(y_val, rf_probabilities)
print(f"Random Forest Brier score loss: {rf_brier_score}")
```

```
Random Forest Brier score loss: 0.11295756
```

In [3]:
```
import pandas as pd

# Load the test data
test_data = pd.read_csv('test.csv')

# Predict probabilities using the pipeline
# The pipeline will automatically handle the preprocessing
test_probabilities = clf.predict_proba(test_data.drop(['ID'], axis=1))[:, 1]

# Create the submission DataFrame
submission = pd.DataFrame({
    'ID': test_data['ID'],
    'TARGET': test_probabilities
})

# Save the submission file
submission.to_csv('random_forest.csv', index=False)
print("Submission file created successfully.")
```

```
Submission file created successfully.
```

In [5]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.metrics import brier_score_loss
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.pipeline import Pipeline

# Load the training data
train_data = pd.read_csv('train.csv')

# Identify categorical and numerical columns
categorical_cols = ['Education', 'EmploymentType', 'MaritalStatus', 'HasMort
numerical_cols = ['Age', 'Income', 'LoanAmount', 'CreditScore', 'MonthsEmplo

# Define the preprocessing steps
```

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols)
    ])

# Append classifier to preprocessing pipeline
# RandomForest with AdaBoost
rf = RandomForestClassifier(n_estimators=10, random_state=42)  # Using fewer
ada_boost = AdaBoostClassifier(base_estimator=rf, n_estimators=50, random_st

clf = Pipeline(steps=[('preprocessor', preprocessor),
                      ('classifier', ada_boost)])

# Separate features and target
X = train_data.drop(['ID', 'Default'], axis=1)
y = train_data['Default']

# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, rando

# Train the AdaBoosted RandomForest model
clf.fit(X_train, y_train)

# Predict probabilities on the validation set
probabilities = clf.predict_proba(X_val)[:, 1]

# Evaluate the model using Brier score loss
brier_score = brier_score_loss(y_val, probabilities)
print(f"Brier score loss with AdaBoosted RandomForest: {brier_score}")
```

```
/Users/yunzheyu/miniconda3/envs/tensorflow/lib/python3.10/site-packages/skl
earn/ensemble/_base.py:166: FutureWarning: `base_estimator` was renamed to
`estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
Brier score loss with AdaBoosted RandomForest: 0.11919698546731154
```