Risk Prediction Face-Off: Evaluating ANNs Against Logistic Regression for Credit Card risk of Customers

Author : Zeedan Shaikh (Bsc & Msc in Statistics Presidency University,kolkata)

Introduction

When assessing the risk of credit card default, financial institutions rely on predictive models to make informed decisions about lending. Two prominent methodologies for tackling this problem are Logistic Regression and Artificial Neural Networks (ANNs). Both techniques offer unique advantages and limitations, and choosing between them can significantly impact the accuracy and efficiency of risk prediction.

Logistic Regression is a traditional statistical method commonly used for binary classification problems, such as predicting whether a customer will default on a credit card payment. Its simplicity, interpretability, and relatively low computational cost make it a popular choice in the financial sector. Logistic Regression models the probability of default as a function of various customer characteristics, providing insights that are easy to understand and communicate.

In contrast, **Artificial Neural Networks** represent a more complex approach, inspired by the neural processes of the human brain. ANNs are capable of capturing intricate patterns and relationships within the data through their multiple layers and nodes. This capacity for modeling non-linear interactions and handling large datasets can potentially lead to more accurate predictions. However, the trade-offs include higher computational demands and a more opaque decision-making process, which can be challenging for interpretability.

Motiavtion

Understanding how these models compare in the context of credit card risk prediction is crucial for optimizing decision-making processes in financial services. This comparison not only highlights the strengths and weaknesses of each method but also helps in selecting the most appropriate model based on the specific needs and constraints of the lending institution.

Objective

- It is very important to observe we cannot really control **What Type of Customer Will Approach the Bank** so here regression aspect of logistic model is meaningless as we cannot change the predictors to get the suitable value of response.
- So we will use **Logistic Regression** for prediction Purpose and also compare it with the results of **ANN**
  - Accuracy comparison
  - F Score comparison
  - ROC Curve visulization

Methodology

Data Collection

Dataset Link : Click Here

▶ Variable Details

Data preprocessing

Getting the required libraries:

```
In [1]:
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from imblearn.over_sampling import SMOTENC
from sklearn.preprocessing import StandardScaler
from IPython.display import display, HTML
#--model
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau, ModelCheckpoint, EarlyStopping
# Metrices
from sklearn.metrics import accuracy_score , classification_report , precision_score , recall_score , confusion_matrix , f1_score , roc_auc_score , roc_curve
```

Loading the data

```
In [2]:
credit_data=pd.read_csv("/content/drive/MyDrive/credit_approval/dataset.csv")
credit_data.head()
```

Out[2]:

	ID	Gender	Own_car	Own_property	Work_phone	Phone	Email	Unemployed	Num_children	Num_family	Account_length	Total_income	Age	Years_employed	Income_type	Education_type	Family_status	Housing_type	Occupation_type	Target
0	5008804	1	1	1	1	0	0	0	0	2	15	427500.0	32.868574	12.435574	Working	Higher education	Civil marriage	Rented apartment	Other	1
1	5008806	1	1	1	0	0	0	0	0	2	29	112500.0	58.793815	3.104787	Working	Secondary_secondary special	Married	House_apartment	Security staff	0
2	5008808	0	0	1	0	1	1	0	0	1	4	270000.0	52.321403	8.353354	Commercial associate	Secondary_secondary special	Single_unmarried	House_apartment	Sales staff	0
3	5008812	0	0	1	0	0	0	1	0	1	20	283500.0	61.504343	0.000000	Pensioner	Higher education	Separated	House_apartment	Other	0
4	5008815	1	1	1	1	1	1	0	0	2	5	270000.0	46.193967	2.105450	Working	Higher education	Married	House_apartment	Accountants	0

```
In [3]:
#--checking NA values
credit_data.isna().sum().sum()
```

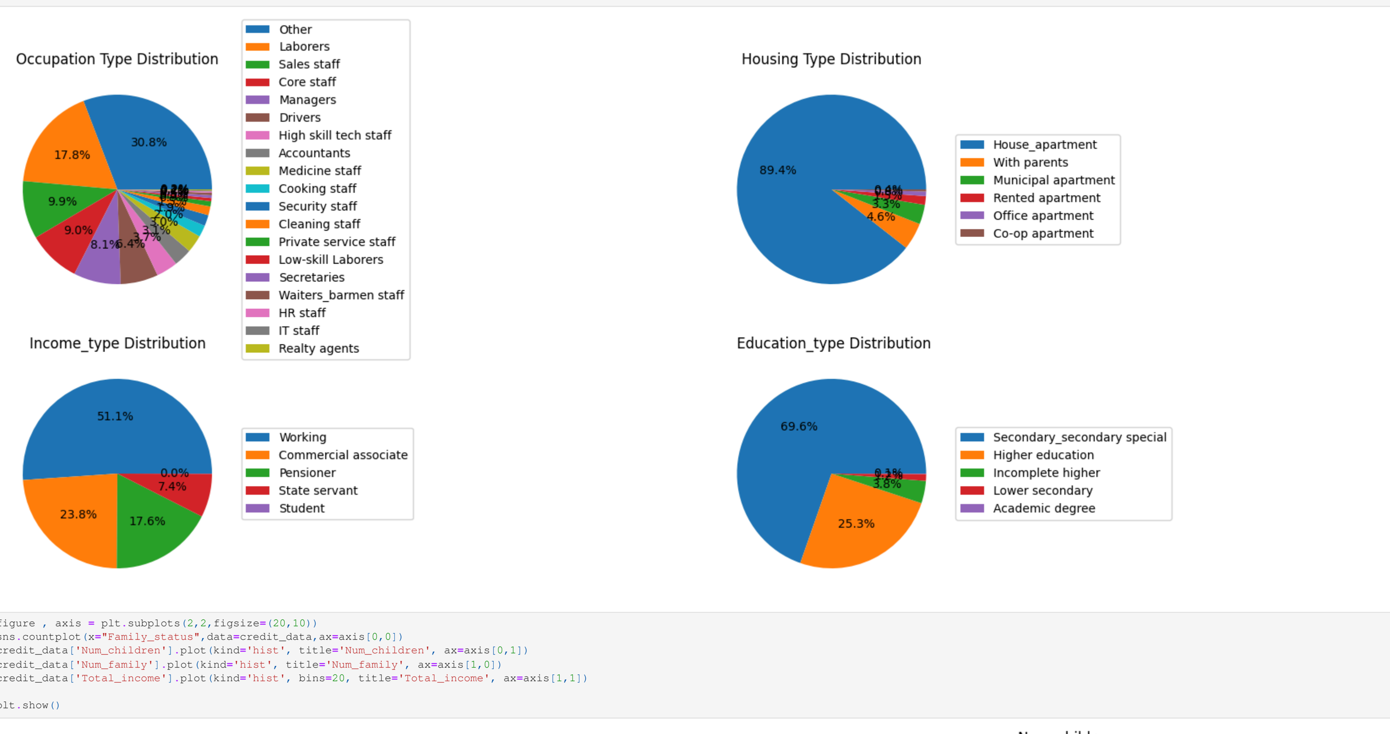
Out[3]: 0

```
In [4]:
#--removing the unnecessary columns and creating factor variables
credit_data.drop(credit_data.columns[[0]],axis=1,inplace=True)
for i in np.array([0,1,2,3,4,5,6,13,14,15,16,17,18]):
  credit_data.iloc[:,i]=pd.Categorical(credit_data.iloc[:,i])

credit_data['Occupation_type'] = pd.Categorical(credit_data['Occupation_type'])
credit_data['Housing_type']= pd.Categorical(credit_data['Housing_type'])
credit_data['Education_type']= pd.Categorical(credit_data['Education_type'])
credit_data['Family_status']= pd.Categorical(credit_data['Family_status'])
credit_data['Income_type']= pd.Categorical(credit_data['Income_type'])
credit_data.info()
```

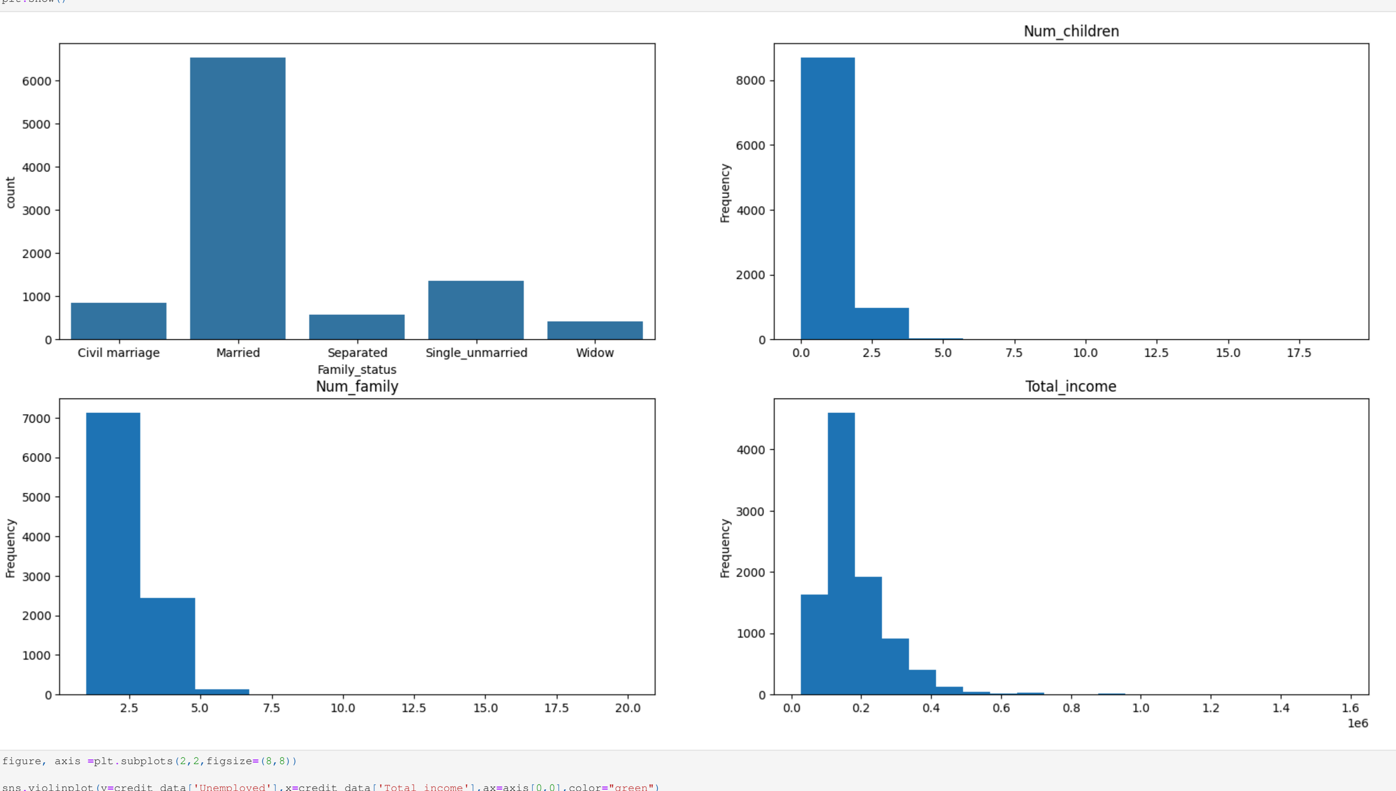
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9709 entries, 0 to 9708
Data columns (total 19 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Gender           9709 non-null   category
 1   Own_car          9709 non-null   category
 2   Own_property     9709 non-null   category
 3   Work_phone       9709 non-null   category
 4   Phone            9709 non-null   category
 5   Email            9709 non-null   category
 6   Unemployed       9709 non-null   category
 7   Num_children     9709 non-null   int64
 8   Num_family       9709 non-null   int64
 9   Account_length   9709 non-null   int64
 10  Total_income     9709 non-null   float64
 11  Age              9709 non-null   float64
 12  Years_employed   9709 non-null   float64
 13  Income_type      9709 non-null   category
 14  Education_type   9709 non-null   category
 15  Family_status    9709 non-null   category
 16  Housing_type     9709 non-null   category
 17  Occupation_type  9709 non-null   category
 18  Target           9709 non-null   category
dtypes: category(13), float64(3), int64(3)
memory usage: 581.0 KB
```

Data Visualisation

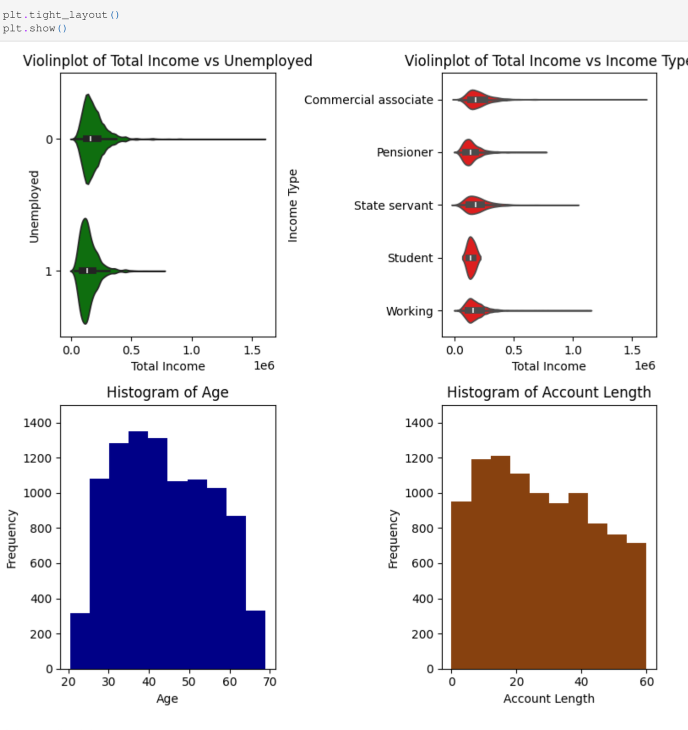
```
In [5]:
figure , axis = plt.subplots(2,2,figsize=(20,8))
occupation_counts = credit_data['Occupation_type'].value_counts()
axis[0,0].pie(occupation_counts, autopct='%1.1f%%')
axis[0,0].set_title('Occupation Type Distribution')
axis[0,0].legend(occupation_counts.index, loc='center left', bbox_to_anchor=(1, 0.5))


housing_counts = credit_data['Housing_type'].value_counts()
axis[0,1].pie(housing_counts, autopct='%1.1f%%')
axis[0,1].set_title('Housing Type Distribution')
axis[0,1].legend(housing_counts.index, loc='center left', bbox_to_anchor=(1, 0.5))

Income_type_counts = credit_data['Income_type'].value_counts()
axis[1,0].pie(Income_type_counts, autopct='%1.1f%%')
axis[1,0].set_title('Income_type Distribution')
axis[1,0].legend(Income_type_counts.index, loc='center left', bbox_to_anchor=(1, 0.5))

Education_type_counts = credit_data['Education_type'].value_counts()
axis[1,1].pie(Education_type_counts, autopct='%1.1f%%')
axis[1,1].set_title(' Education_type Distribution')
axis[1,1].legend(Education_type_counts.index, loc='center left', bbox_to_anchor=(1, 0.5))

plt.show()
```




```
In [6]:
figure , axis = plt.subplots(2,2,figsize=(20,10))
sns.countplot(x="Family_status",data=credit_data,ax=axis[0,0])
credit_data['Num_children'].plot(kind='hist', title='Num_children', ax=axis[0,1])
credit_data['Num_family'].plot(kind='hist', title='Num_family', ax=axis[1,0])
credit_data['Total_income'].plot(kind='hist', bins=20, title='Total_income', ax=axis[1,1])

plt.show()
```




```
In [7]:
figure, axis =plt.subplots(2,2,figsize=(8,8))

sns.violinplot(y=credit_data['Unemployed'],x=credit_data['Total_income'],ax=axis[0,0],color="green")
axis[0,0].set_xlabel("Total Income")
axis[0,0].set_ylabel("Unemployed")
axis[0,0].set_title("Violinplot of Total Income vs Unemployed")


sns.violinplot(y=credit_data['Income_type'],x=credit_data['Total_income'],ax=axis[0,1],color="red")
axis[0,1].set_xlabel("Total Income")
axis[0,1].set_ylabel("Income Type")
axis[0,1].set_title("Violinplot of Total Income vs Income Type")

axis[1,0].hist(credit_data["Age"],color="darkblue")
axis[1,0].set_xlabel("Age")
axis[1,0].set_ylabel("Frequency")
axis[1,0].set_title("Histogram of Age")
axis[1,0].set_ylim([0,1500])

axis[1,1].hist(credit_data["Account_length"],color="saddlebrown")
axis[1,1].set_xlabel("Account Length")
axis[1,1].set_ylabel("Frequency")
axis[1,1].set_title("Histogram of Account Length")
axis[1,1].set_ylim([0,1500])

plt.tight_layout()
plt.show()
```



Balancing the data

```
In [8]:
#--balancing the data
credit_data.shape
credit_data['Target'].value_counts()
```

Out[8]:

Target	count
0	8426
1	1283

dtype: int64

```
In [9]:
X=np.array(credit_data.drop(columns=['Target']))
y=credit_data['Target']
smote_nc=SMOTENC([0,1,2,3,4,5,6,13,14,15,16,17],random_state=45)
df=smote_nc.fit_resample(X,y)
credit_data_bal=pd.DataFrame(df[0])
credit_data_bal['Target']=df[1]
credit_data_bal.columns=credit_data.columns
credit_data_bal['Target'].value_counts()
```

Out[9]:

Target	count
0	8426
1	8426

dtype: int64

```
In [10]:
#--converting in suitable format
for i in credit_data_bal.columns[[0,1,2,3,4,5,6,13,14,15,16,17,18]]:
  credit_data_bal[i]=pd.Categorical(credit_data_bal[i])
for i in credit_data_bal.columns[[7,8,9,10,11,12]]:
  credit_data_bal[i]=pd.to_numeric(credit_data_bal[i])
for i in credit_data_bal.columns[[7,8,9]]:
  credit_data_bal[i]=np.floor(credit_data_bal[i]).astype(int)
credit_data_bal.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16852 entries, 0 to 16851
Data columns (total 19 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Gender           16852 non-null  category
 1   Own_car          16852 non-null  category
 2   Own_property     16852 non-null  category
 3   Work_phone       16852 non-null  category
 4   Phone            16852 non-null  category
 5   Email            16852 non-null  category
 6   Unemployed       16852 non-null  category
 7   Num_children     16852 non-null  int64
 8   Num_family       16852 non-null  int64
 9   Account_length   16852 non-null  int64
 10  Total_income     16852 non-null  float64
 11  Age              16852 non-null  float64
 12  Years_employed   16852 non-null  float64
 13  Income_type      16852 non-null  category
 14  Education_type   16852 non-null  category
 15  Family_status    16852 non-null  category
 16  Housing_type     16852 non-null  category
 17  Occupation_type  16852 non-null  category
 18  Target           16852 non-null  category
dtypes: category(13), float64(3), int64(3)
memory usage: 1006.5 KB
```

```
In [11]:
#--Scaling the neumerical variables and using OneHotEncoding for categorical variables
std=StandardScaler()
credit_data_bal_encoded=pd.get_dummies(credit_data_bal,columns=credit_data_bal.columns[[0,1,2,3,4,5,6,13,14,15,16,17]],drop_first=True)
credit_data_bal_encoded[credit_data_bal_encoded.columns[[0,1,2,3,4,5]]]=std.fit_transform(credit_data_bal_encoded[credit_data_bal_encoded.columns[[0,1,2,3,4,5]]])

for i in credit_data_bal_encoded.columns[np.arange(7,49)]:
  credit_data_bal_encoded[i]=credit_data_bal_encoded[i].astype(int)

credit_data_bal_encoded['Target']=credit_data_bal_encoded['Target'].astype(float)
```

Model Building

```
In [12]:
#--creating train-test data
X=credit_data_bal_encoded.drop(columns=['Target'])
y=credit_data_bal_encoded['Target']
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2)
```

Logistic regression

```
In [13]:
model1=LogisticRegression(max_iter=1000)
model1.fit(X_train,y_train)
```

Out[13]:

```
▾ LogisticRegression
LogisticRegression(max_iter=1000)
```

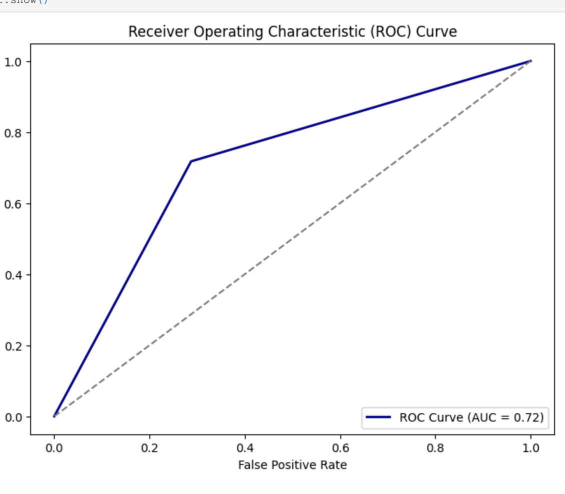
Checking accuracy metric on Test data

```
In [14]:
pred1=model1.predict(X_test)
accuracy = accuracy_score(y_test, pred1)
f1 = f1_score(y_test, pred1)
print(f"Accuracy: {accuracy:.2f}")
print(f"F1 score: {f1:.2f}")
```

```
Accuracy: 0.72
F1 score: 0.72
```

ROC Curve and AUC score

```
In [15]:
roc_auc = roc_auc_score(y_test, pred1)
fpr, tpr, thresholds = roc_curve(y_test, pred1)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkblue', lw=2, label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```



Artificial Neural Network Model

```
In [16]:
model2 = Sequential()

model2.add(Dense(512,input_dim=X_train.shape[1],activation='relu',kernel_initializer='he_normal'))
model2.add(Dropout(0.2))

model2.add(Dense(512,activation='relu',kernel_initializer='he_normal'))
model2.add(Dropout(0.2))

model2.add(Dense(256,activation='relu',kernel_initializer='he_normal'))

model2.add(Dense(64,activation='relu',kernel_initializer='he_normal'))

model2.add(Dense(1, activation='sigmoid'))

model2.compile(optimizer=Adam(learning_rate =0.001),loss='binary_crossentropy',metrics=['accuracy'])

early_stopping = EarlyStopping(monitor='val_accuracy',patience=10,restore_best_weights=True)

model2.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 512)	25,088
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 512)	262,656
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 256)	131,328
dense_3 (Dense)	(None, 64)	16,448
dense_4 (Dense)	(None, 1)	65

Total params: 435,585 (1.66 MB)
Trainable params: 435,585 (1.66 MB)
Non-trainable params: 0 (0.00 B)

```
In [17]:
history = model2.fit(x= X_train, y= y_train, validation_data=(X_test,y_test), batch_size= 256, epochs=1000, verbose=0,
          callbacks=[early_stopping])
```

Predicting accuracy for ANN model

```
In [18]:
pred2= model2.predict(X_test)
pred2_binary = (pred2 > 0.5).astype(int)  # Convert probabilities to binary predictions (0 or 1)
accuracy = accuracy_score(y_test, pred2_binary)
f1 =f1_score(y_test, pred2_binary)
print(f"Accuracy: {accuracy:.2f}")
display(HTML(f"<h3>Model Accuracy: {np.round(accuracy,2)}</h3>"))
print(f"F1 Score: {f1:.2f}")
display(HTML(f"<h3>Model F1 score: {np.round(f1,2)}</h3>"))
```

```
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Accuracy: 0.83
```

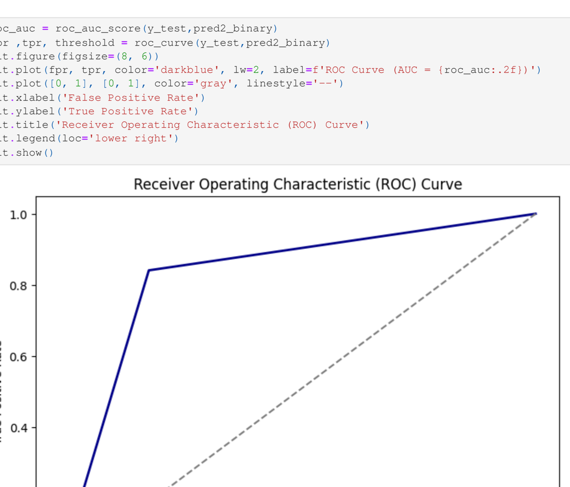
Model Accuracy: 0.83

```
F1 Score: 0.83
```

Model F1 score: 0.83

ROC curve and AUC score

```
In [19]:
roc_auc = roc_auc_score(y_test,pred2_binary)
fpr ,tpr, threshold = roc_curve(y_test,pred2_binary)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkblue', lw=2, label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```



Function for predicting risk using the ANN model

```
In [20]:
new_data=pd.DataFrame(0,index=[0],columns=X_train.columns)
def predict_credit_risk(
    Gender, Work_phone, Own_car, Own_property,Phone, Email, Unemployed, Num_children, Num_family,
    Total_income, Age, Years_employed, Income_type, Account_length, Housing_type,
    Occupation_type, Education_type,Family_status
    ):
    # Gender
    if Gender == "Male":
        new_data.iloc[:,6] = 1

    # Work phone
    if Work_phone == "Yes":
        new_data.iloc[:,9] = 1

    # Own car
    if Own_car == "Yes":
        new_data.iloc[:,7] = 1

    # Own_property
    if Own_property == "Yes":
        new_data.iloc[:,8] = 1
    # Phone
    if Phone == "Yes":
        new_data.iloc[:,10] = 1

    # Email
    if Email == "Yes":
         new_data.iloc[:,11] = 1

    # Unemployed
    if Unemployed == "Yes":
        new_data.iloc[:,12] = 1

    # Income type
    if Income_type == "Working":
        new_data.iloc[:,16] = 1
    elif Income_type == "Pensioner":
        new_data.iloc[:,13] = 1
    elif Income_type == "State servant":
        new_data.iloc[:,14] = 1
    elif Income_type == "Student":
        new_data.iloc[:,15] = 1

    # Education type
    if Education_type == "Higher education":
        new_data.iloc[:,17] = 1
    elif Education_type == "Secondary_secondary special":
        new_data.iloc[:,20] = 1
    elif Education_type == "Incomplete higher":
        new_data.iloc[:,18] = 1
    elif Education_type == "Lower secondary":
        new_data.iloc[:,19] = 1

    # Family status
    if Family_status == "Married":
        new_data.iloc[:,21] = 1
    elif Family_status == "Separated":
        new_data.iloc[:,22] = 1
    elif Family_status == "Single_unmarried":
        new_data.iloc[:,23] = 1
    elif Family_status == "Widow":
        new_data.iloc[:,24] = 1

    # Housing type
    if Housing_type == "Rented apartment":
        new_data.iloc[:,28] = 1
    elif Housing_type == "With parents":
        new_data.iloc[:,29] = 1
    elif Housing_type == "Municipal apartment":
        new_data.iloc[:,26] = 1
    elif Housing_type == "House_apartment":
        new_data.iloc[:,25] = 1
    elif Housing_type == "Office apartment":
        new_data.iloc[:,27] = 1

    # Occupation type
    if Occupation_type == "Laborers":
        new_data.iloc[:,37] = 1
    elif Occupation_type == "Core staff":
        new_data.iloc[:,32] = 1
    elif Occupation_type == "Managers":
        new_data.iloc[:,39] = 1
    elif Occupation_type == "Sales staff":
        new_data.iloc[:,44] = 1
    elif Occupation_type == "Drivers":
        new_data.iloc[:,33] = 1
    elif Occupation_type == "High skill tech staff":
        new_data.iloc[:,35] = 1
    elif Occupation_type == "Medicine staff":
        new_data.iloc[:,40] = 1
    elif Occupation_type == "IT staff":
        new_data.iloc[:,36] = 1
    elif Occupation_type == "Cleaning staff":
        new_data.iloc[:,30] = 1
    elif Occupation_type == "Cooking staff":
        new_data.iloc[:,31] = 1
    elif Occupation_type == "HR staff":
        new_data.iloc[:,34] = 1
    elif Occupation_type == "Low-skill Laborers":
        new_data.iloc[:,38] = 1
    elif Occupation_type == "Realty agents":
        new_data.iloc[:,43] = 1
    elif Occupation_type == "Security staff":
        new_data.iloc[:,46] = 1
    elif Occupation_type == "Waiters_barmen staff":
        new_data.iloc[:,47] = 1
    elif Occupation_type == "Other":
        new_data.iloc[:,41] = 1
    elif Occupation_type == "Private service staff":
        new_data.iloc[:,42] = 1
    elif Occupation_type == "Secretaries":
        new_data.iloc[:,45] = 1


    # Normalize numeric inputs
    new_data.iloc[:,0] = (Num_children - X_train['Num_children'].describe()[1]) / X_train['Num_children'].describe()[2]
    new_data.iloc[:,1] = (Num_family - X_train['Num_family'].describe()[1]) / X_train['Num_family'].describe()[2]
    new_data.iloc[:,2] = (Account_length - X_train['Account_length'].describe()[1]) / X_train['Account_length'].describe()[2]
    new_data.iloc[:,3] = (((Total_income)/11.75) - X_train['Total_income'].describe()[1]) / X_train['Total_income'].describe()[2]
    new_data.iloc[:,4] = (Age - X_train['Age'].describe()[1]) / X_train['Age'].describe()[2]
    new_data.iloc[:,5] = (Years_employed - X_train['Years_employed'].describe()[1]) / X_train['Years_employed'].describe()[2]

     # Predict credit risk
    pred2 = model2.predict(new_data)

    # Convert prediction to binary class
    prediction = (pred2 > 0.5).astype(int)

    if prediction == 1:
        prediction_final = "High "
    else:
        prediction_final = "Low "

    return prediction_final
```

Let us try to predict on a given input

```
In [21]:
pred_datapoint=predict_credit_risk(
    Gender="Male",
    Work_phone="Yes",
    Own_car="Yes", Phone="Yes",
    Email="Yes",
    Unemployed="No",
    Num_children=0,
    Num_family=4,
    Total_income=200000,
    Age=22, Years_employed=1,
    Income_type="Working",
    Account_length=12,
    Housing_type="With parents",
    Occupation_type="IT staff",
    Education_type="Higher education",
    Family_status="Single_unmarried",
    Own_property="Yes")
display(HTML(f"<h3>Credit card user risk : {pred_datapoint}</h3>"))
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
<ipython-input-20-68cdc8e09a73>:116: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  new_data.iloc[:,0] = (Num_children - X_train['Num_children'].describe()[1]) / X_train['Num_children'].describe()[2]
<ipython-input-20-68cdc8e09a73>:116: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  new_data.iloc[:,0] = (Num_children - X_train['Num_children'].describe()[1]) / X_train['Num_children'].describe()[2]
<ipython-input-20-68cdc8e09a73>:117: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  new_data.iloc[:,1] = (Num_family - X_train['Num_family'].describe()[1]) / X_train['Num_family'].describe()[2]
<ipython-input-20-68cdc8e09a73>:117: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  new_data.iloc[:,1] = (Num_family - X_train['Num_family'].describe()[1]) / X_train['Num_family'].describe()[2]
<ipython-input-20-68cdc8e09a73>:118: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  new_data.iloc[:,2] = (Account_length - X_train['Account_length'].describe()[1]) / X_train['Account_length'].describe()[2]
<ipython-input-20-68cdc8e09a73>:118: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  new_data.iloc[:,2] = (Account_length - X_train['Account_length'].describe()[1]) / X_train['Account_length'].describe()[2]
<ipython-input-20-68cdc8e09a73>:119: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  new_data.iloc[:,3] = (((Total_income)/11.75) - X_train['Total_income'].describe()[1]) / X_train['Total_income'].describe()[2]
<ipython-input-20-68cdc8e09a73>:119: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  new_data.iloc[:,3] = (((Total_income)/11.75) - X_train['Total_income'].describe()[1]) / X_train['Total_income'].describe()[2]
<ipython-input-20-68cdc8e09a73>:120: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  new_data.iloc[:,4] = (Age - X_train['Age'].describe()[1]) / X_train['Age'].describe()[2]
<ipython-input-20-68cdc8e09a73>:120: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  new_data.iloc[:,4] = (Age - X_train['Age'].describe()[1]) / X_train['Age'].describe()[2]
<ipython-input-20-68cdc8e09a73>:121: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  new_data.iloc[:,5] = (Years_employed - X_train['Years_employed'].describe()[1]) / X_train['Years_employed'].describe()[2]
<ipython-input-20-68cdc8e09a73>:121: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  new_data.iloc[:,5] = (Years_employed - X_train['Years_employed'].describe()[1]) / X_train['Years_employed'].describe()[2]
```

Credit card user risk : High

Conclusion

Summmary Findings

- **We have a clear higher accuracy in ANN model than Logistic model.**
- **We have a clear higher F1 score in ANN model than Logistic model.**
- **We have a clear higher ROC AUC in ANN model than Logistic model.**

Higher Accuracy: The ANN model has demonstrated superior accuracy compared to the Logistic Regression model. This indicates that the ANN model is more proficient at correctly classifying both the default and non-default classes overall.

Higher F1 Score: The ANN model's higher F1 score suggests it performs better in balancing precision and recall, which is crucial in imbalanced datasets where one class (e.g., high risk) may be less frequent. This indicates that the ANN model is more effective at minimizing both false positives and false negatives, which is especially important in credit card risk prediction where the cost of misclassification can be high.

Higher ROC AUC: The ANN model also exhibits a higher ROC AUC score, reflecting its enhanced ability to discriminate between the high risk and low risk classes across different classification thresholds. A higher ROC AUC confirms that the ANN model is more effective at distinguishing between high-risk and low-risk customers.

Discussion

Model Complexity and Capability:

- ANN Advantages: The superior performance of the ANN model can be attributed to its ability to capture complex, non-linear relationships within the data. ANNs are designed to learn intricate patterns through their multiple layers and neurons, which makes them more flexible and capable of modeling complex interactions between features. Logistic Regression Limitations: Logistic Regression, while simpler and more interpretable, is inherently limited to modeling linear relationships. It may not capture the complexities and non-linearities in the data as effectively as an ANN.

Interpretability vs. Performance:

- ANN Trade-offs: Despite its higher performance metrics, the ANN model comes with increased complexity, which can make it harder to interpret. This "black box" nature of ANNs can be a drawback in contexts where model interpretability is crucial for regulatory or decision-making purposes. Logistic Regression Strengths: Logistic Regression's main advantage lies in its transparency and ease of interpretation. It provides clear insights into how individual features affect the probability of default, which can be valuable for understanding and explaining model predictions.

Implementation and Cost:

- Computational Resources: The ANN model typically requires more computational resources and longer training times compared to Logistic Regression. This increased cost can be a factor in deployment, especially in real-time or resource-constrained environments. Practical Considerations: If real-time prediction and lower computational cost are critical, Logistic Regression might still be preferred despite its lower performance. On the other hand, if maximizing predictive accuracy is the primary goal, and resources are available, the ANN model is the better choice. Data Handling:

Recommendations

- **For High Accuracy and Complex Data:** If your priority is achieving the highest possible accuracy and your data exhibits complex patterns, the ANN model is the better option. Ensure that you have the computational resources and tools for model training and deployment.
- **For Interpretability and Simplicity:** If interpretability, ease of implementation, and computational efficiency are more important, Logistic Regression remains a viable choice. It provides valuable insights into the model's decision-making process, which can be crucial in a financial context.

```
In [22]:
!jupyter nbconvert --execute --to html /content/drive/MyDrive/credit_approval/credit_card_risk.ipynb
```

```
[NbConvertApp] Converting notebook /content/drive/MyDrive/credit_approval/credit_card_risk.ipynb to html
2024-09-11 17:44:40.926457: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:485] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT when one has already been registered
2024-09-11 17:44:40.981458: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:8454] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when one has already been registered
2024-09-11 17:44:40.997291: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1452] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already been registered
2024-09-11 17:44:43.294953: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
[NbConvertApp] Writing 667129 bytes to /content/drive/MyDrive/credit_approval/credit_card_risk.html
```