Importing Packages

In [1]:
```python
import pandas as pd
```

Data Loading

In [2]:
```python
df = pd.read_csv("dataset/loan.csv") # Loading the dataset "loan.csv"
```

In [3]:
```python
df.head() # Checking contents from the loaded dataset. The "default_status" column
          # where FALSE = 0 and TRUE = 1
```

Out[3]:

	loan_type	loan_amount	interest_rate	loan_term	employment_type	income_level	credi
0	Car Loan	16795	0.051852	15	Self-employed	Medium	
1	Personal Loan	1860	0.089296	56	Full-time	Medium	
2	Personal Loan	77820	0.070470	51	Full-time	Low	
3	Car Loan	55886	0.062155	30	Full-time	Low	
4	Home Loan	7265	0.070635	48	Part-time	Low	

In [4]:
```python
df.info() # Checking for null values and data type
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   loan_type        5000 non-null   object 
 1   loan_amount      5000 non-null   int64  
 2   interest_rate    5000 non-null   float64
 3   loan_term        5000 non-null   int64  
 4   employment_type  5000 non-null   object 
 5   income_level     5000 non-null   object 
 6   credit_score     5000 non-null   int64  
 7   gender           5000 non-null   object 
 8   marital_status   5000 non-null   object 
 9   education_level  5000 non-null   object 
 10  default_status   5000 non-null   int64  
dtypes: float64(1), int64(4), object(6)
memory usage: 429.8+ KB
```

Data Transformation

In [5]:
```
df['default_status'].unique() # Checking for data type of each column and converted
```

Out[5]:
```
array([0, 1], dtype=int64)
```

In [6]:
```
# Conversion of data from categorical to numerical
df['loan_type'] = df['loan_type'].map({'Car Loan':0, 'Personal Loan':1, 'Home Loan'
df['employment_type'] = df['employment_type'].map({'Self-employed':2, 'Full-time':1
df['income_level'] = df['income_level'].map({'Medium':1, 'Low':0, 'High':2}).astype
df['gender'] = df['gender'].map({'Male':1, 'Female':0}).astype('int')
df['marital_status'] = df['marital_status'].map({'Single':0, 'Married':1, 'Divorced
df['education_level'] = df['education_level'].map({'Master':2, 'Bachelor':1, 'High
```

In [7]:
```
df.info() # Checking for converted data type. To proceed with modeling, all data ty
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   loan_type        5000 non-null   int32  
 1   loan_amount      5000 non-null   int64  
 2   interest_rate    5000 non-null   float64
 3   loan_term        5000 non-null   int64  
 4   employment_type  5000 non-null   int32  
 5   income_level     5000 non-null   int32  
 6   credit_score     5000 non-null   int64  
 7   gender           5000 non-null   int32  
 8   marital_status   5000 non-null   int32  
 9   education_level  5000 non-null   int32  
 10  default_status   5000 non-null   int64  
dtypes: float64(1), int32(6), int64(4)
memory usage: 312.6 KB
```

In [8]:
```
df.head() # Rreviewing data content. Data are on different ranges. Applying feature
```

Out[8]:

	loan_type	loan_amount	interest_rate	loan_term	employment_type	income_level	credi
0	0	16795	0.051852	15	2	1	
1	1	1860	0.089296	56	1	1	
2	1	77820	0.070470	51	1	0	
3	0	55886	0.062155	30	1	0	
4	2	7265	0.070635	48	0	0	

In [9]:
```
# Setting data for X and y
X = df.drop('default_status', axis=1)
y = df['default_status']
```

In [10]:
```
X.shape
```

Out[10]: (5000, 10)

In [11]:
```
y.shape
```

Out[11]: (5000,)

Feature Scaling

In [12]:
```
# Setting cols as variable for feature scaling
cols = ['loan_amount', 'interest_rate', 'loan_term', 'credit_score']
```

In [13]:
```
# Using RobustScaler for scaling
from sklearn.preprocessing import RobustScaler
st = RobustScaler()
X[cols] = st.fit_transform(X[cols])
```

In [14]:
```
# Cheking for the scaled value
X
```

Out[14]:

	loan_type	loan_amount	interest_rate	loan_term	employment_type	income_level	c
0	0	-0.656992	-1.334365	-0.869565	2	1	
1	1	-0.954917	0.470603	0.913043	1	1	
2	1	0.560343	-0.436900	0.695652	1	0	
3	0	0.122801	-0.837700	-0.217391	1	0	
4	2	-0.847098	-0.428934	0.565217	0	0	
...	...	...	...	...	...	...	
4995	0	-0.235089	-0.455318	0.956522	2	2	
4996	1	-0.015819	-1.114868	0.652174	0	1	
4997	2	-0.842888	-0.738553	1.000000	1	2	
4998	0	0.060363	0.741455	-1.000000	2	1	
4999	1	0.825274	0.206698	0.739130	2	0	

5000 rows × 10 columns

Model Training using Random Forest Classifier

Random Forest Classifier

In [15]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

In [16]:
```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier()
```

In [17]:
```
clf.fit(X_train, y_train)
```

Out[17]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [18]:
```
y_pred = clf.predict(X_test)
```

Feature importance

In [19]:
```
pd.DataFrame(clf.feature_importances_,
             index = X_train.columns,
             columns=['Importance']).sort_values('Importance', ascending=False)
```

Out[19]:

	Importance
interest_rate	0.200989
loan_amount	0.200648
credit_score	0.193229
loan_term	0.152956
education_level	0.053153
loan_type	0.050723
income_level	0.044471
marital_status	0.041698
employment_type	0.040090
gender	0.022044

The feature importance indicates the significance of each of the features in a model. Interest rate and Loan amount has the most influence in these features which has the highest importance value at approximately 19.9%. The second most important feature is the credit score, with an importance value just a little behind the top 2 around 19.2%. Larger loan amounts impact approval chances. Next is the loan term which has importance value of 15.4%. The rest are below 6% which means they have lesser impact when it comes to predicting if a borrower will default.

In [20]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

Out[20]:
```
0.7976
```

The model's accuracy on the test data is around 80.08% which is considerably good. Testing with cross validation score for more comparison.

In [21]:
```
from sklearn.model_selection import cross_val_score
cross_val_score(clf, X_train, y_train, cv=10)
```

Out[21]:
```
array([0.80266667, 0.79733333, 0.8       , 0.80266667, 0.80266667,
       0.79733333, 0.8       , 0.8       , 0.79733333, 0.8       ])
```

Cross-validation results indicate consistent performance across different folds.