

BUSINESS UNDERSTANDING

Credit card default poses a significant financial risk to banks. Accurately identifying customers who are likely to default allows financial institutions to take proactive steps such as adjusting credit limits, offering repayment plans, or increasing monitoring.

The objective of this project is to build a classification model that predicts whether a customer will default on their credit card payment in the next month. This problem is well suited for classification because the target outcome is categorical: default or no default.

The primary stakeholder for this analysis is the credit risk management team at a commercial bank.

In [1]:

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

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import( accuracy_score, confusion_matrix, classification_report, recall score, precision score, f1 score)
```

In [2]:

```
#Load Dataset  
df = pd.read_csv('UCI_Credit_Card.csv')  
df.head()
```

Out[2]:

ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	BILL_AMT4	BILL_AMT5	BILL_AMT6	
0	1	200000.0	2	2	1	24	2	2	-1	-1	...	0.0	0.0	0
1	2	120000.0	2	2	2	26	-1	2	0	0	...	3272.0	3455.0	3261
2	3	90000.0	2	2	2	34	0	0	0	0	...	14331.0	14948.0	15549
3	4	50000.0	2	2	1	37	0	0	0	0	...	28314.0	28959.0	29547
4	5	50000.0	1	2	1	57	-1	0	-1	0	...	20940.0	19146.0	19131

5 rows × 25 columns

In [3]:

```
#Dataset Shape and Info  
df.shape
```

Out[3]:

(30000, 25)

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 25 columns):
 #   Column           Non-Null Count  Dtype  
 ---  --  
 0   id               30000 non-null   int64  
 1   gender           30000 non-null   object 
 2   race_color       30000 non-null   object 
 3   age              30000 non-null   float64
 4   education_level 30000 non-null   object 
 5   marital_status   30000 non-null   object 
 6   occupation       30000 non-null   object 
 7   relationship     30000 non-null   object 
 8   sex              30000 non-null   object 
 9   capital_gain     30000 non-null   float64
 10  capital_loss    30000 non-null   float64
 11  hours_per_week  30000 non-null   int64  
 12  native_country   30000 non-null   object 
 13  salary           30000 non-null   object 
 14  salary_in_usd   30000 non-null   float64
```

```
0      ID                      30000  non-null  int64
1  LIMIT_BAL                  30000  non-null  float64
2      SEX                      30000  non-null  int64
3  EDUCATION                  30000  non-null  int64
4  MARRIAGE                  30000  non-null  int64
5      AGE                      30000  non-null  int64
6    PAY_0                     30000  non-null  int64
7    PAY_2                     30000  non-null  int64
8    PAY_3                     30000  non-null  int64
9    PAY_4                     30000  non-null  int64
10   PAY_5                     30000  non-null  int64
11   PAY_6                     30000  non-null  int64
12  BILL_AMT1                 30000  non-null  float64
13  BILL_AMT2                 30000  non-null  float64
14  BILL_AMT3                 30000  non-null  float64
15  BILL_AMT4                 30000  non-null  float64
16  BILL_AMT5                 30000  non-null  float64
17  BILL_AMT6                 30000  non-null  float64
18  PAY_AMT1                  30000  non-null  float64
19  PAY_AMT2                  30000  non-null  float64
20  PAY_AMT3                  30000  non-null  float64
21  PAY_AMT4                  30000  non-null  float64
22  PAY_AMT5                  30000  non-null  float64
23  PAY_AMT6                  30000  non-null  float64
24 default.payment.next.month 30000  non-null  int64
dtypes: float64(13), int64(12)
memory usage: 5.7 MB
```

In [5]:

```
#Rename Target Column
df.rename(columns={'default.payment.next.month':'Default'}, inplace=True)
```

In [6]:

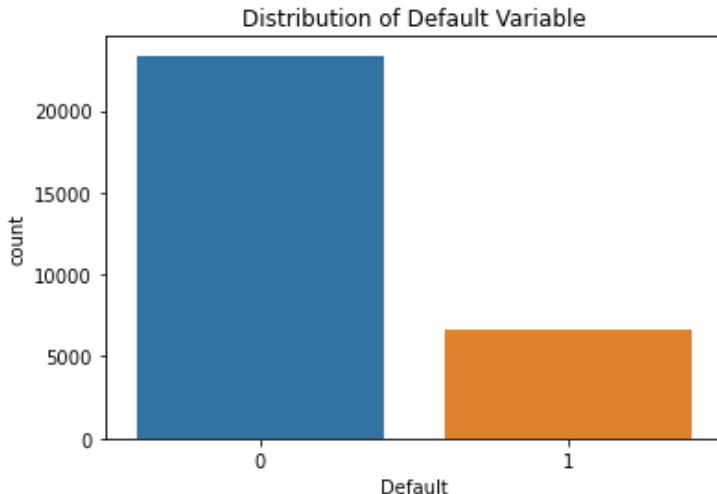
```
#Target Variable Distribution
df['Default'].value_counts()
```

Out[6]:

```
0    23364
1    6636
Name: Default, dtype: int64
```

In [7]:

```
sns.countplot(x='Default', data=df)
plt.title('Distribution of Default Variable')
plt.show()
```



The target variable is imbalanced, with a smaller proportion of customers defaulting. This reflects real-world financial data and informs the choice of evaluation metrics.

DATA PREPARATION

In [8]:

```
#Drop Unnecessary Columns  
df.drop(['ID'], axis=1, inplace=True)
```

In [9]:

```
x = df.drop('Default', axis=1)  
y = df['Default']
```

In [10]:

```
#Train Test Split  
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

In [11]:

```
#Feature Scaling  
scaler = StandardScaler()  
X_train_scaled = scaler.fit_transform(X_train)  
X_test_scaled = scaler.transform(X_test)
```

This feature scaling is applied to ensure that all variables contribute equally to the logistic regression model

4. Modelling -Iterative Approach

In [12]:

```
#Baseline Model - Logistic Regression  
log_reg = LogisticRegression(max_iter=1000)  
log_reg.fit(X_train_scaled, y_train)
```

Out[12]:

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

EVALUATION #TRAINING DATA

In [13]:

```
y_train_pred = log_reg.predict(X_train_scaled)  
print("Training Recall:", recall_score(y_train, y_train_pred))  
print("Training Precision:", precision_score(y_train, y_train_pred))  
print("Training F1 Score:", f1_score(y_train, y_train_pred))  
print("Training Accuracy:", accuracy_score(y_train, y_train_pred))
```

```
Training Recall: 0.24290813451061433  
Training Precision: 0.7159468438538206  
Training F1 Score: 0.36274372282227524  
Training Accuracy: 0.8107083333333334
```

EVALUATION # TEST DATA

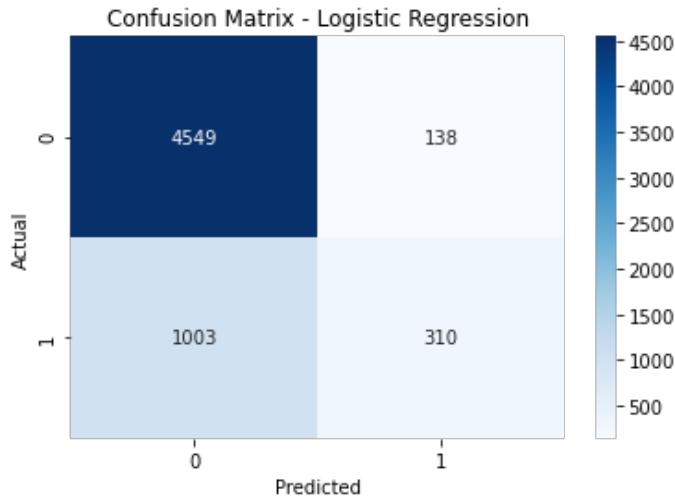
In [14]:

```
y_test_pred = log_reg.predict(X_test_scaled)  
print("Testing Recall:", recall_score(y_test, y_test_pred))  
print("Testing Precision:", precision_score(y_test, y_test_pred))  
print("Testing F1 Score:", f1_score(y_test, y_test_pred))  
print("Testing Accuracy:", accuracy_score(y_test, y_test_pred))
```

```
Testing Recall: 0.2361005331302361  
Testing Precision: 0.6919642857142857  
Testing F1 Score: 0.35207268597387853  
Testing Accuracy: 0.8098333333333333
```

In [15]:

```
#Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_test_pred)
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Logistic Regression')
plt.show()
```



Recall is prioritized as the primary evaluation metric because failing to identify a customer who will default poses a higher financial risk than incorrectly flagging a non-defaulting customer

MODEL ITERATION -Tuned Logistic Regression

Justification

To improve recall for the default class, the logistic regression model is re-trained using class weighting to penalize missclassification of defaulting customers more heavily,

In [16]:

```
#Tuned Model -
log_reg_tuned = LogisticRegression(max_iter=1000, class_weight='balanced', C=0.5)
log_reg_tuned.fit(X_train_scaled, y_train)
```

Out[16]:

```
LogisticRegression(C=0.5, class_weight='balanced', max_iter=1000)
```

In [17]:

```
#Evaluation of Tuned Model
y_test_pred_tuned = log_reg_tuned.predict(X_test_scaled)
print("Tuned Recall:", recall_score(y_test, y_test_pred_tuned))
print("Tuned Precision:", precision_score(y_test, y_test_pred_tuned))
print("Tuned F1 Score:", f1_score(y_test, y_test_pred_tuned))
print("Tuned Accuracy:", accuracy_score(y_test, y_test_pred_tuned))
```

```
Tuned Recall: 0.642041127189642
Tuned Precision: 0.38007213706041476
Tuned F1 Score: 0.47748513169073914
Tuned Accuracy: 0.6925
```

In [18]:

```
#Non- Parametric Model _Decision Tree (Rubric Requirement )
dt = DecisionTreeClassifier(max_depth=5, random_state =42)
dt.fit(X_train, y_train)
```

Out[18]:

```
DecisionTreeClassifier(max_depth=5, random_state=42)
```

In [19]:

```
y_test_pred_dt = dt.predict(X_test)
print("Decision Tree Recall:", recall_score(y_test, y_test_pred_dt))
print("Decision Tree Precision:", precision_score(y_test, y_test_pred_dt))
print("Decision Tree F1 Score:", f1_score(y_test, y_test_pred_dt))
print("Decision Tree Accuracy:", accuracy_score(y_test, y_test_pred_dt))
```

```
Decision Tree Recall: 0.36785986290936784
Decision Tree Precision: 0.6535859269282814
Decision Tree F1 Score: 0.47076023391812866
Decision Tree Accuracy: 0.819
```

In [20]:

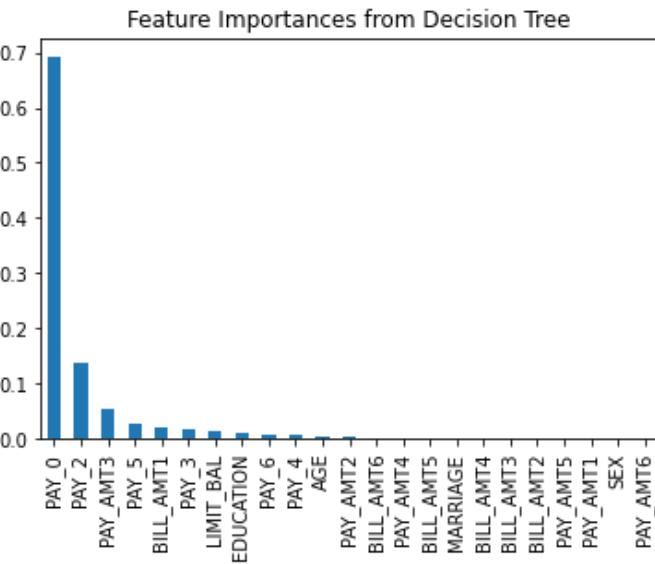
```
#Feature Importance from Decision Tree
feature_importances = pd.Series(dt.feature_importances_, index=X_train.columns)
feature_importances = feature_importances.sort_values(ascending=False)
feature_importances.head(10)
```

Out [20]:

```
PAY_0      0.691073
PAY_2      0.138058
PAY_AMT3   0.052769
PAY_5      0.026196
BILL_AMT1  0.019974
PAY_3      0.016626
LIMIT_BAL   0.014608
EDUCATION  0.009715
PAY_6      0.008470
PAY_4      0.007714
dtype: float64
```

In [21]:

```
feature_importances.plot(kind='bar')
plt.title('Feature Importances from Decision Tree')
plt.show()
```



Final Model Discussion

The tuned logistic regression model achieved the highest recall on the test dataset, making it the most suitable model for this business problem. By prioritizing recall, the model successfully identifies a greater proportion of customers who are likely to default, which is critical for risk management.

While the decision tree provided interpretability and insight into feature importance, it underperformed compared to the tuned logistic regression in terms of recall. Payment history variables consistently emerged as the most influential predictors of default behavior.

This model can support proactive credit risk interventions; however, it should not be used as the sole decision-making tool. Future improvements could include threshold tuning, ensemble models, and incorporation of additional behavioral data.

Limitations

1. **Class Imbalance:** The dataset contains fewer customers who default, which can bias the model toward predicting non-default. Techniques like class weighting helped, but some imbalance-related errors remain.
2. **False Negatives:** Even with a high recall, the model may fail to identify some customers who are likely to default, which could lead to financial losses.
3. **Feature Scope:** The dataset includes historical payment and demographic data but lacks behavioral, employment, or external financial indicators that could improve prediction accuracy.
4. **Model Interpretability vs Complexity:** While Random Forest captures complex patterns, it is less interpretable than logistic regression, which may affect stakeholder trust and decision-making.
5. **Temporal Limitations:** The dataset is static and may not reflect future shifts in customer behavior. The model requires periodic retraining to remain relevant.

Recommendations and Next Steps

1. **Deploy Tuned Logistic Regression:** Based on the evaluation metrics, the tuned logistic regression model provides the best balance of recall and interpretability. It should be deployed to help the credit risk team proactively identify customers at risk of default.
2. **Monitor Model Performance:** Regularly monitor recall and other key metrics to detect any drift in model performance, especially as customer behavior changes over time.
3. **Feature Enhancement:** Incorporate additional features such as behavioral, employment, or external financial data to improve prediction accuracy.
4. **Threshold Tuning:** Adjust the decision threshold to balance false positives and false negatives according to business priorities.
5. **Ensemble Exploration:**
Consider ensemble methods like Random Forest or Gradient Boosting for future iterations, especially if prediction accuracy becomes more critical than interpretability.
6. **Continuous Retraining:** Retrain the model periodically with new data to account for shifts in customer behavior or changes in the economic environment.