BANK LOAN APPROVAL PREDICTION – LOGISTIC REGRESSION PROJECT

```
Predict whether a bank should approve a customer's loan
Goal: Predict Approved = 1 or Not Approved = 0 using customer
features.
```

Project Covers:

```
Clean dataset loading
Feature selection
Train/test split
Scaling
Training (Logistic Regression)
Evaluation (Accuracy, Precision, Recall, F1)
Confusion Matrix + Classification Report
Decision Thresholds
AUC-ROC Curve
Prediction function
```

IMPORT LIBRARIES

In [4]:
```
# pandas: Used for data handling and analysis
import pandas as pd

# numpy: Used for numerical operations
import numpy as np

# matplotlib.pyplot: Used for basic data visualization
import matplotlib.pyplot as plt

# seaborn: Built on matplotlib, used for advanced & attractive plots
import seaborn as sns

# train_test_split: Used to divide data into training & testing sets
# Ensures model is tested on unseen data
from sklearn.model_selection import train_test_split

# StandardScaler: Used to scale features (mean = 0, std = 1)
# Important for models like Logistic Regression
from sklearn.preprocessing import StandardScaler

# LogisticRegression: A classification algorithm
```

```python
# Used to predict binary outcomes (Yes/No, 0/1)
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import (
    accuracy_score,  # Accuracy: Out of all predictions, how many were correct
    precision_score,  # Precision: Out of all YES predictions, how many were a
    recall_score,  # Recall: Out of all actual YES cases, how many did the mod
    f1_score,  # F1 Score: A single score that balances Precision and Recall
    confusion_matrix,  # A table that shows Correct and wrong predictions (YES
    classification_report,  # Shows Accuracy, Precision, Recall, and F1 togeth
    roc_curve,  # A graph that shows how well the model separates YES and NO
    roc_auc_score # A number that tells how good the model is at separating cl
);
```

LOAD DATASET

In [6]:

```python
df = pd.read_csv("bank_loan.csv")

print("Dataset Loaded Successfully!")

print(df.head())
```

```
Dataset Loaded Successfully!
   Age  Income  CreditScore EmploymentStatus  Approved
0   59   40358          812       Unemployed         0
1   49   23267          595       Unemployed         0
2   35  102745          619         Salaried         1
3   63  109588          871    Self-employed         0
4   28   58513          648         Salaried         1
```

In [7]:

```python
print("=================== DATA OVERVIEW ====================")

# Shape of the dataset
print("\nShape:", df.shape)

# Data types and info
print("\nDataset Info:")
df.info()

# Check missing values
print("\nMissing Values in Each Column:")
print(df.isnull().sum());
```

```
=================== DATA OVERVIEW ====================

Shape: (500, 5)

Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 5 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   Age               500 non-null    int64 
 1   Income            500 non-null    int64 
 2   CreditScore       500 non-null    int64 
 3   EmploymentStatus  500 non-null    object
 4   Approved          500 non-null    int64 
dtypes: int64(4), object(1)
memory usage: 19.7+ KB

Missing Values in Each Column:
Age                 0
Income              0
CreditScore         0
EmploymentStatus    0
Approved            0
dtype: int64
```

In [8]:

```python
# Count each Employment Status category
print("\nEmployment Status Distribution:")
print(df["EmploymentStatus"].value_counts())

# Preview the last 10 rows
print("\nLast 10 Rows of the Dataset:")
print(df.tail(10));
```

```
Employment Status Distribution:
EmploymentStatus
Salaried         289
Self-employed    136
Unemployed        75
Name: count, dtype: int64

Last 10 Rows of the Dataset:
     Age  Income  CreditScore EmploymentStatus  Approved
490   43   32219          870    Self-employed         0
491   37   20235          662    Self-employed         0
492   46   62929          856    Self-employed         0
493   28  147309          644         Salaried         1
494   49   87444          469         Salaried         1
495   46  159639          655         Salaried         1
496   30  121834          490         Salaried         0
497   46  104555          721         Salaried         1
498   54   93698          306       Unemployed         0
499   61  144450          432         Salaried         0
```

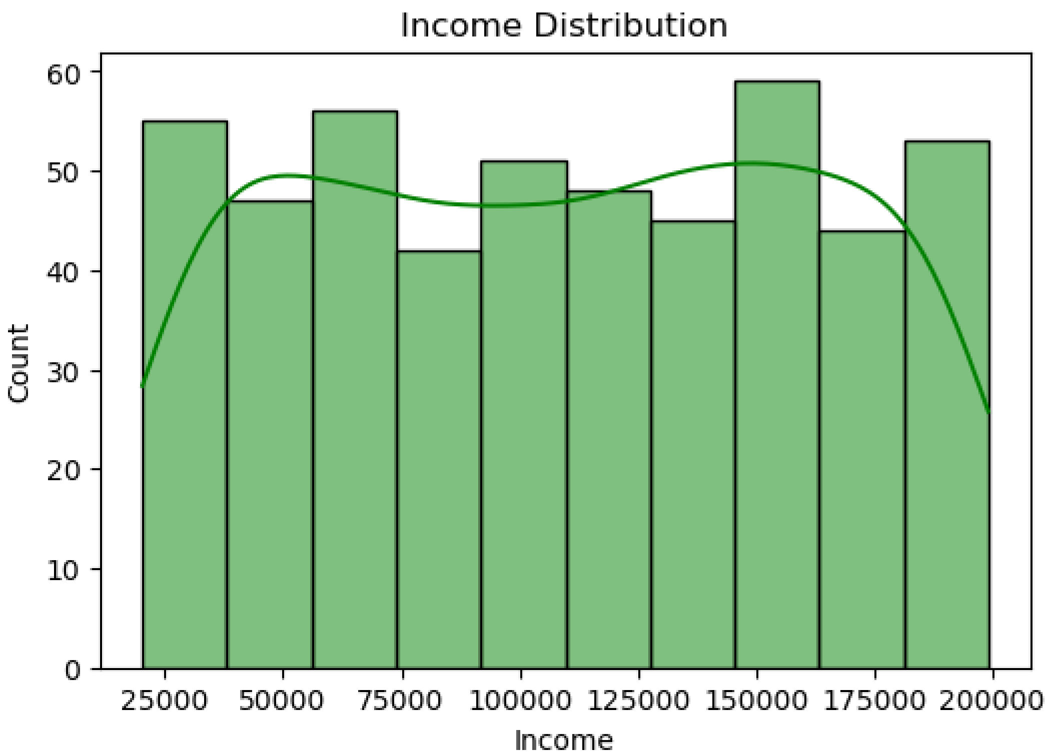
In [9]:

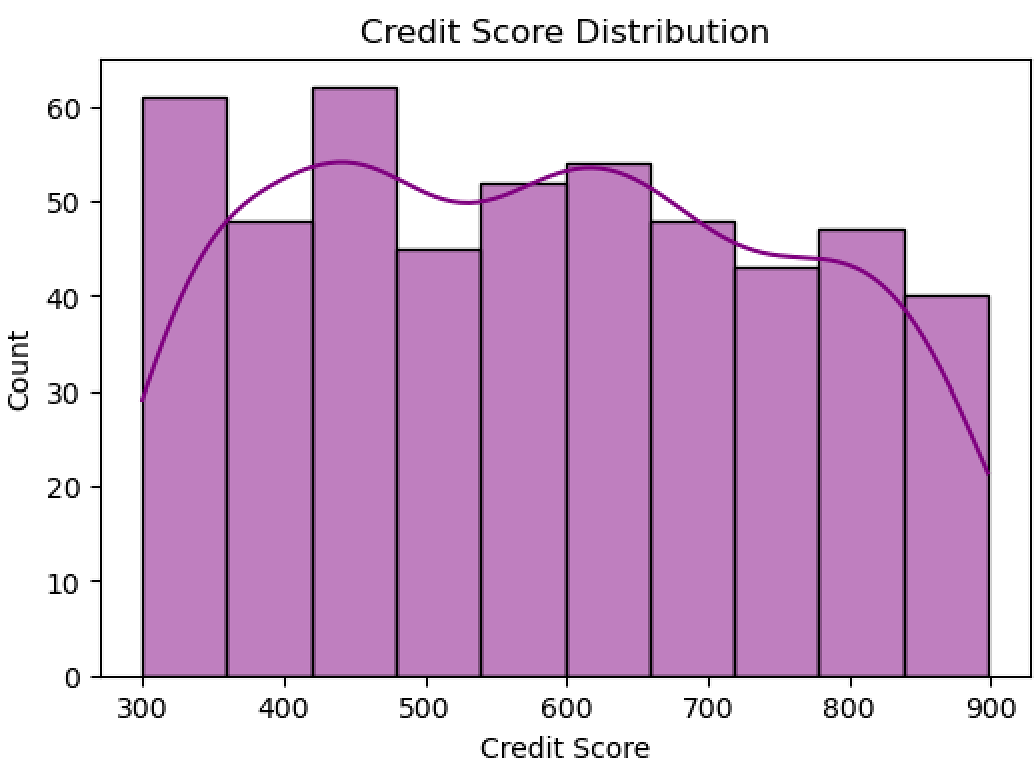
```
# Summary statistics for numerical features
print("\nSummary Statistics:")
print(df.describe())

# Unique values in Employment Status
print("\nUnique Employment Status Values:")
print(df["EmploymentStatus"].unique())
```

```
Summary Statistics:
              Age         Income  CreditScore    Approved
count  500.000000     500.000000   500.000000  500.000000
mean    43.116000  109045.696000   583.490000    0.502000
std     12.733217   52974.205023   171.614805    0.500497
min     21.000000   20235.000000   300.000000    0.000000
25%     32.000000   62479.250000   440.000000    0.000000
50%     44.000000  109228.500000   585.500000    1.000000
75%     53.000000  156195.000000   723.500000    1.000000
max     64.000000  199208.000000   898.000000    1.000000

Unique Employment Status Values:
['Unemployed' 'Salaried' 'Self-employed']
```

In [10]:

```
# ------------------------------
plt.figure(figsize=(6,4))
sns.histplot(df["Income"], kde=True, color="green")
plt.title("Income Distribution")
plt.xlabel("Income")
plt.ylabel("Count")
plt.show()
# ------------------------------
plt.figure(figsize=(6,4))
sns.histplot(df["CreditScore"], kde=True, color="purple")
plt.title("Credit Score Distribution")
plt.xlabel("Credit Score")
plt.ylabel("Count")
plt.show()
```

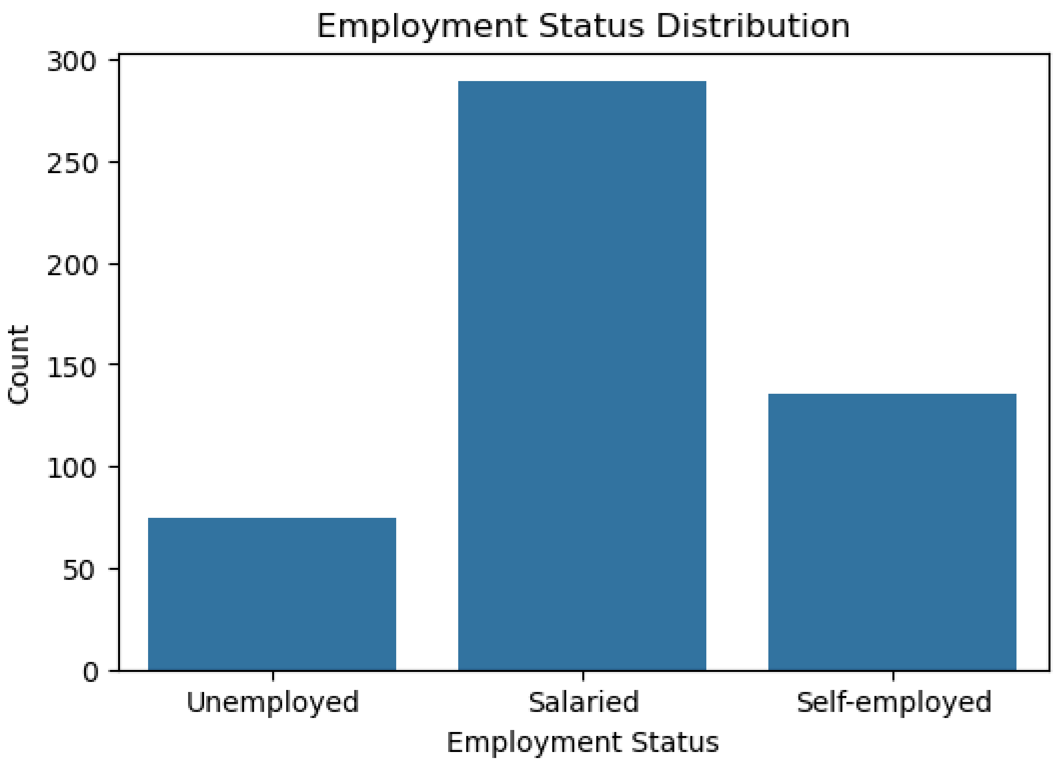




In [11]:

```
# 4. Employment Status Count Plot
plt.figure(figsize=(6,4))
sns.countplot(x="EmploymentStatus", data=df)
plt.title("Employment Status Distribution")
plt.xlabel("Employment Status")
plt.ylabel("Count")
```

```
plt.show()
```



FEATURE SELECTION & CLEANING

In [13]:
```
# Select important features
df = df[["Age", "Income", "CreditScore", "EmploymentStatus", "Approved"]]
```

Encode Employment Status (numeric encoding)

In [15]:
```
mapping = {"Unemployed": 0, "Self-employed": 1, "Salaried": 2}
df["EmploymentEncoded"] = df["EmploymentStatus"].map(mapping);
```

In [16]:
```
# Final Feature / Target
X = df[["Age", "Income", "CreditScore", "EmploymentEncoded"]]
y = df["Approved"];

print("\nFinal Features:\n", X.head())
```

```
Final Features:
    Age  Income  CreditScore  EmploymentEncoded
0   59   40358          812                  0
1   49   23267          595                  0
2   35  102745          619                  2
3   63  109588          871                  1
4   28   58513          648                  2
```

TRAIN–TEST SPLIT

In [18]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42, stratify=y
);
print("Train size:", X_train.shape)
print("Test size:", X_test.shape)
```

```
Train size: (375, 4)
Test size: (125, 4)
```

FEATURE SCALING

In [20]:
```
scaler = StandardScaler()
# Create a scaler that standardizes data (mean = 0, std = 1)

X_train_scaled = scaler.fit_transform(X_train)
# Learn scaling from training data and apply it

X_test_scaled = scaler.transform(X_test)
# Apply the SAME scaling to test data (no learning)
```

TRAIN LOGISTIC REGRESSION MODEL

In [22]:
```
model = LogisticRegression(max_iter=1000) # Improves accuracy
# Create a Logistic Regression model

model.fit(X_train_scaled, y_train)
# Train the model using the scaled training data and their correct labels

print("Model Training Completed!")
# Confirm that training is finished
```

```
Model Training Completed!
```

MODEL PREDICTIONS

In [24]:
```
y_pred = model.predict(X_test_scaled)
# Predict class labels (0 or 1) for the test data

y_prob = model.predict_proba(X_test_scaled)[:, 1]
# Get probability of class 1 (loan approved) for each test sample
```

EVALUATION METRICS

In [69]:
```
print("\n================ MODEL EVALUATION ================")

print("Accuracy :", accuracy_score(y_test, y_pred));
# How many total predictions were correct
```

```
print("Precision:", precision_score(y_test, y_pred));
# Out of all predicted YES, how many were actually YES (controls false positiv

print("Recall   :", recall_score(y_test, y_pred));
# Out of all actual YES, how many the model correctly found (controls false ne

print("F1 Score :", f1_score(y_test, y_pred));
# Balance of Precision and Recall in one score
```

```
================ MODEL EVALUATION ================
Accuracy : 0.624
Precision: 0.6212121212121212
Recall   : 0.6507936507936508
F1 Score : 0.6356589147286822
```

Model Evalution

    Accuracy = 0.62 (62%)
    Out of 100 loan predictions, about 62 are correct.

    Precision = 0.62 (62%)
    When the model says “YES – loan approved”, it is correct 62
    times out of 100.

    Recall = 0.65 (65%)
    Out of all people who should get the loan, the model correctly
    finds 65% of them.

    F1 Score = 0.63 (63%)
    Overall balanced performance between Precision and Recall.

    Percentages help us understand model performance easily,
    regardless of dataset size.

CONFUSION MATRIX + CLASSIFICATION REPORT

In [78]:
```
print("\n=============== CONFUSION MATRIX ===============")
print(confusion_matrix(y_test, y_pred));
# Shows TP, FP, TN, FN to understand correct and incorrect predictions
```

```
=============== CONFUSION MATRIX ===============
[[37 25]
 [22 41]]
```

Confusion Matrix

    37 → Correctly predicted NO

25 → Wrongly predicted YES (but actually NO)
22 → Wrongly predicted NO (but actually YES)
41 → Correctly predicted YES

In [83]:
```
print("\n=========== CLASSIFICATION REPORT =============")
print(classification_report(y_test, y_pred));
# Shows Precision, Recall, F1-score, and support for each class
```

```
=========== CLASSIFICATION REPORT =============
              precision    recall  f1-score   support

           0       0.63      0.60      0.61        62
           1       0.62      0.65      0.64        63

    accuracy                           0.62       125
   macro avg       0.62      0.62      0.62       125
weighted avg       0.62      0.62      0.62       125
```

Class 0 (NO / Loan Rejected)

Precision 63% → When model says NO, it is correct 63 times out of 100
Recall 60% → Found 60% of actual NO cases
F1-score 61% → Balanced score for NO class
Support 62 → Total 62 NO records

Class 1 (YES / Loan Approved)

Precision 62% → When model says YES, it is correct 62 times out of 100
Recall 65% → Found 65% of actual YES cases
F1-score 64% → Balanced score for YES class
Support 63 → Total 63 YES records

Accuracy = 62% → Model predicted 62 correct out of 100
Macro Avg → Simple average of both classes
Weighted Avg → Average based on number of records

Decision Threshold Analysis

Decision Threshold Analysis means deciding at what probability value the model should say YES or NO.
≥ 0.5 → YES
< 0.5 → NO

In [88]:
```python
threshold = 0.5; # Equal importance to YES and NO | Balanced datasets

print("\nDefault Threshold =", threshold)

print("Prediction Probabilities (first 10):")

print(y_prob[:20]);  # Shows the first 10 probability values predicted by the
```

```
Default Threshold = 0.5
Prediction Probabilities (first 10):
[0.32716288 0.68291553 0.52806798 0.675654   0.38418969 0.71675992
 0.69488846 0.63420475 0.59271349 0.5549581  0.45405383 0.57621246
 0.718221   0.08868347 0.85116035 0.36293882 0.27502011 0.64420197
 0.18260534 0.24739485]
```

AUC–ROC Curve

AUC–ROC tells us how well a model can separate positive and negative cases.
It measures how good the model is at distinguishing YES vs NO.

In [95]:
```python
# AUC tells how well the model separates Yes vs No.

# Step 1: Calculate False Positive Rate, True Positive Rate and Thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
# fpr = how many wrong positives at each threshold
# tpr = how many correct positives at each threshold
# thresholds = different probability cut-off values

# Step 2: Calculate AUC score
auc = roc_auc_score(y_test, y_prob)
# AUC tells how well the model separates class 0 and class 1
# Higher AUC = better model

print("\nAUC Score:", auc)
```

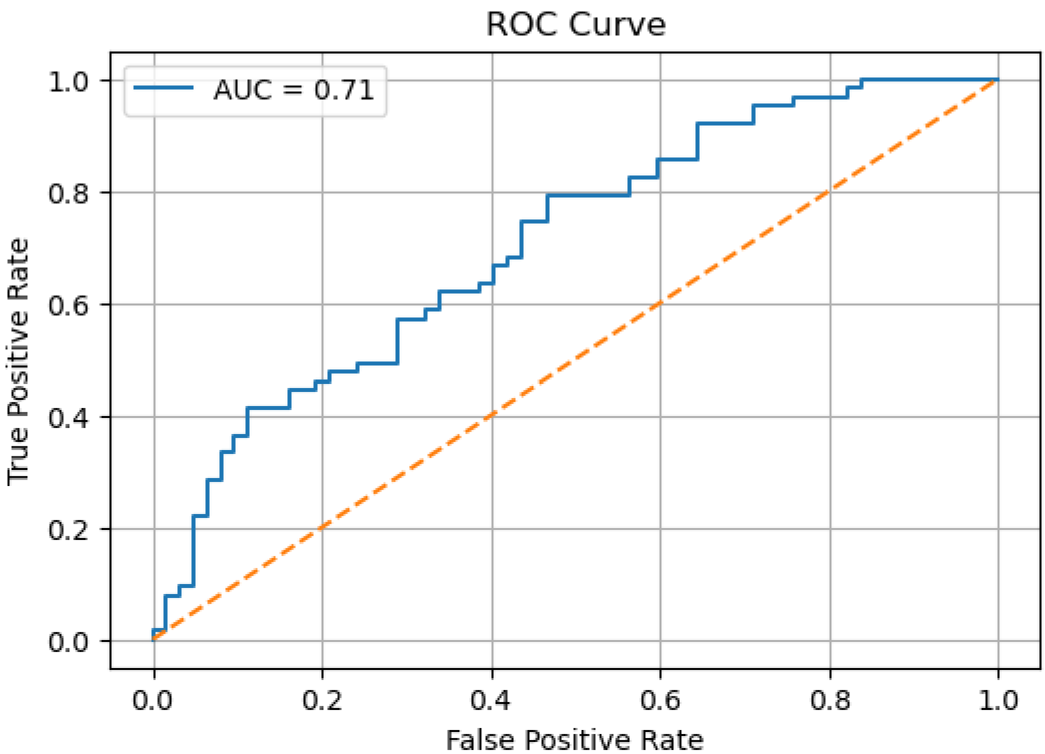
```
AUC Score: 0.7073732718894009
```

AUC Score = 0.71

Means 71% chance the model will correctly separate YES vs NO

In [99]:
```python
plt.figure(figsize=(6,4))
plt.plot(fpr, tpr, label=f"AUC = {auc:.2f}")
# Plot the ROC curve

plt.plot([0,1], [0,1], '--') # Draws a random guessing line | Used for compari

plt.xlabel("False Positive Rate");
plt.ylabel("True Positive Rate");
```

```
plt.title("ROC Curve")
plt.legend()
plt.grid(True)
plt.show()
```



PREDICTION FUNCTION

In [119…

```
def predict_loan(age, income, credit, employment):

    # 1. Put the customer data into a DataFrame
    row = pd.DataFrame([[age, income, credit, mapping[employment]]],
                       columns=["Age", "Income", "CreditScore", "EmploymentEnc

    # 2. Scale the data (same scaling used during training)
    row_scaled = scaler.transform(row);

    # 3. Get probability of loan approval
    prob = model.predict_proba(row_scaled)[0][1]

    # 4. Final prediction (0 or 1) using threshold 0.5
    pred = 1 if prob >= 0.5 else 0;

    # 5. Print simple output
    print(f"\nLoan Approval Probability = {prob:.2f}")

    if pred == 1:
        print("Loan Approved")
    else:
```

```python
        print("Loan Not Approved")

predict_loan(25, 60000, 800, "Salaried")
predict_loan(55, 30000, 500, "Unemployed")
```

```
Loan Approval Probability = 0.57
Loan Approved

Loan Approval Probability = 0.12
Loan Not Approved
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: