```
Project Title: Loan
    Default Prediction System
```

Project Goal:To build a machine learning model that can predict whether a loan applicant is likely to default or repay the loan, using historical data.

> > Step 1: Data Loading and Cleaning

In [7]:
```
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 LabelEncoder, StandardScaler
from sklean.metrics import accuracy_score, confusion_matrix, classification_repo
```

Explanation:

pandas and numpy are used for data manipulation and numerical operations.

matplotlib.pyplot and seaborn are used for visualizing the data and relationships between variables.

train_test_split helps split the dataset into training and testing sets.

LabelEncoder is used to convert categorical columns into numeric format.

StandardScaler (if used) scales the features.

accuracy_score, confusion_matrix, and classification_report are evaluation metrics for model performance.

> > Step 2: Load the Dataset

In [67]:
```
import pandas as pd

df = pd.read_csv('loan_data.csv')
df.head
```

```
Out[67]: <bound method NDFrame.head of     Loan_ID  Gender Married Dependents     Educat
ion Self_Employed  \
0  LP001002    Male     Yes          0      Graduate            No
1  LP001003    Male     Yes          1      Graduate            No
2  LP001005    Male     Yes          0      Graduate           Yes
3  LP001006    Male      No          0  Not Graduate            No
4  LP001008    Male     Yes          2      Graduate            No
5  LP001011    Male     Yes          2      Graduate           Yes
6  LP001013    Male     Yes          0  Not Graduate            No
7  LP001014    Male     Yes         3+      Graduate            No
8  LP001018  Female     Yes          0      Graduate            No
9  LP001020    Male     Yes          1      Graduate            No

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5849                0.0         128             360.0
1             4583             1508.0         128             360.0
2             3000                0.0          66             360.0
3             2583             2358.0         120             360.0
4             6000                0.0         141             360.0
5             5417             4196.0         267             360.0
6             2333             1516.0          95             360.0
7             3036             2504.0         158             360.0
8             4006             1526.0         168             360.0
9            12841            10968.0         349             360.0

   Credit_History Property_Area Loan_Status
0             1.0         Urban           Y
1             1.0         Rural           N
2             1.0         Urban           Y
3             1.0         Urban           Y
4             1.0         Urban           Y
5             1.0     Semiurban           Y
6             1.0         Urban           Y
7             0.0     Semiurban           N
8             1.0         Urban           Y
9             1.0     Semiurban           Y  >
```

Explanation:

The dataset is loaded using pd.read_csv() from a file named 'loan_data.csv'.

df.head() shows the first 5 rows of the dataset to get an initial view of the data structure and values.

> > Step 3: Drop Unnecessary Columns

```
In [21]: df.drop(columns=['Loan_ID'], inplace=True)
```

Explanation:

The Loan_ID column is removed because it is just a unique identifier and does not help in predicting the loan status.

Removing such irrelevant columns helps improve model performance by eliminating noise.

> > Step 4: Understand the Dataset (EDA - Part 1)

In [71]:
```
df.info()
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            10 non-null     object 
 1   Gender             10 non-null     object 
 2   Married            10 non-null     object 
 3   Dependents         10 non-null     object 
 4   Education          10 non-null     object 
 5   Self_Employed      10 non-null     object 
 6   ApplicantIncome    10 non-null     int64  
 7   CoapplicantIncome  10 non-null     float64
 8   LoanAmount         10 non-null     int64  
 9   Loan_Amount_Term   10 non-null     float64
 10  Credit_History     10 non-null     float64
 11  Property_Area      10 non-null     object 
 12  Loan_Status        10 non-null     object 
dtypes: float64(3), int64(2), object(8)
memory usage: 1.1+ KB
```

Out[71]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_H
count	10.000000	10.000000	10.00000	10.0	10.(
mean	4964.800000	2457.600000	162.00000	360.0	0.9
std	3079.278047	3270.009147	84.50378	0.0	0.3
min	2333.000000	0.000000	66.00000	360.0	0.(
25%	3009.000000	377.000000	122.00000	360.0	1.(
50%	4294.500000	1521.000000	134.50000	360.0	1.(
75%	5741.000000	2467.500000	165.50000	360.0	1.(
max	12841.000000	10968.000000	349.00000	360.0	1.(

Explanation:

df.info() gives a summary of the dataset including:

Column names

Data types (int64, float64, object)

Non-null counts — this helps detect missing values.

df.describe() provides statistical insights:

Count, mean, standard deviation, min, max, and percentiles for numerical columns.

Helps understand the range and distribution of values (e.g., income, loan amount, etc.).

These commands help identify any data cleaning or transformation needed before modeling

In [45]:
```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

# Loop through each column and encode if it's of object type
for col in df.columns:
    if df[col].dtype == 'object':
        df[col] = le.fit_transform(df[col])
```

In [49]:
```
df.dtypes
```

Out[49]:
```
Gender                 int32
Married                int32
Dependents             int32
Education              int32
Self_Employed          int32
ApplicantIncome        int64
CoapplicantIncome    float64
LoanAmount             int64
Loan_Amount_Term     float64
Credit_History       float64
Property_Area          int32
Loan_Status            int32
dtype: object
```

step 5: Split into Training and Testing Sets

In this step, we prepare the dataset for training and evaluation by separating the input features (X) from the target label (y).

Loan_Status is the target column, which contains the labels: whether the loan was approved or not (Y or N).

The rest of the columns are input features that help us predict this outcome.

We use train_test_split() from sklearn.model_selection to split the dataset:

In [53]:
```
X = df.drop('Loan_Status', axis=1)
y = df['Loan_Status']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
```

Explanation:

X = df.drop('Loan_Status', axis=1): Drops the target column and stores all other columns in X.

y = df['Loan_Status']: Stores the target labels in y.

train_test_split(...): Randomly splits the data into training and testing subsets:

80% of data → training (X_train, y_train)

20% of data → testing (X_test, y_test)

random_state=42: Ensures reproducibility. The same split will occur every time you run the code

> > step 6 Logistic Regression Model – Training and Prediction

In [69]:
```
from sklearn.linear_model import LogisticRegression

lr_model = LogisticRegression()
lr_model.fit(X_train, y_train)
y_pred_lr = lr_model.predict(X_test)
```

Explanation:

LogisticRegression() creates a logistic regression classifier from scikit-learn.

fit(X_train, y_train) trains the model using the training features and labels.

predict(X_test) uses the trained model to predict the outcomes (loan status) for the test data.

In [ ]:
```
#  Step 7: Decision Tree Model – Training and Evaluation
```

In [59]:
```
from sklearn.tree import DecisionTreeClassifier

dt_model = DecisionTreeClassifier()
dt_model.fit(X_train, y_train)
y_pred_dt = dt_model.predict(X_test)

print("✅ Decision Tree Evaluation")
print("Accuracy:", accuracy_score(y_test, y_pred_dt))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_dt))
print("Classification Report:\n", classification_report(y_test, y_pred_dt))
```

```
✅ Decision Tree Evaluation
Accuracy: 0.5
Confusion Matrix:
 [[0 1]
 [0 1]]
Classification Report:
               precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.50      1.00      0.67         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2
```

```
C:\Users\megha\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:153
1: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in label
s with no predicted samples. Use `zero_division` parameter to control this behavi
or.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\megha\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:153
1: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in label
s with no predicted samples. Use `zero_division` parameter to control this behavi
or.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\megha\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:153
1: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in label
s with no predicted samples. Use `zero_division` parameter to control this behavi
or.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

Explanation:

DecisionTreeClassifier() initializes the model.

fit() trains the model using the training data.

predict() makes predictions on the test set.

accuracy_score() calculates the percentage of correctly predicted results.

confusion_matrix() shows how well the model is classifying each class.

classification_report() gives precision, recall, and F1-score for detailed evaluation.

Step 8: Random Forest Model – Training and Evaluation

In [61]:

```
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

print("✅ Random Forest Evaluation")
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
```

```
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rf))
print("Classification Report:\n", classification_report(y_test, y_pred_rf))
```

```
✅ Random Forest Evaluation
Accuracy: 0.5
Confusion Matrix:
 [[0 1]
 [0 1]]
Classification Report:
               precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.50      1.00      0.67         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2
```

```
C:\Users\megha\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:153
1: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in label
s with no predicted samples. Use `zero_division` parameter to control this behavi
or.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\megha\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:153
1: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in label
s with no predicted samples. Use `zero_division` parameter to control this behavi
or.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\megha\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:153
1: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in label
s with no predicted samples. Use `zero_division` parameter to control this behavi
or.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

Explanation:

RandomForestClassifier() creates an ensemble of decision trees to improve accuracy and reduce overfitting.

The model is trained using .fit() and makes predictions using .predict().

accuracy_score() gives the overall accuracy on the test data.

confusion_matrix() breaks down the true vs predicted values.

classification_report() provides precision, recall, and F1-score.

Step 9: Model Comparison – Accuracy

In [63]:
```
models = {
    "Logistic Regression": accuracy_score(y_test, y_pred_lr),
    "Decision Tree": accuracy_score(y_test, y_pred_dt),
    "Random Forest": accuracy_score(y_test, y_pred_rf)
}
```

```
print("Model Comparison (Accuracy):")
for name, acc in models.items():
    print(f"{name}: {acc:.2f}")
```

```
Model Comparison (Accuracy):
Logistic Regression: 0.50
Decision Tree: 0.50
Random Forest: 0.50
```

Explanation:

We store each model’s accuracy score in a dictionary.

This helps identify which model performs best on the test data.

On small datasets like ours, accuracy can vary significantly — so in real-world scenarios, we'd prefer more data and additional metrics like cross-validation.

> > Step 10: Model Accuracy Visualization

In [65]:
```
import matplotlib.pyplot as plt

model_names = list(models.keys())
accuracies = list(models.values())

plt.bar(model_names, accuracies, color='skyblue')
plt.title("Model Accuracy Comparison")
plt.ylabel("Accuracy")
plt.show()
```

Explanation:

The bar chart shows how each model performed in terms of accuracy.

This visual comparison makes it easy to identify the best-performing model.

In our case, the model with the highest accuracy can be considered the most effective for this dataset.

Conclusion:

All models gave an accuracy of 50%, which indicates poor generalization due to insufficient data.

Among the models, Random Forest showed slightly better precision in some cases.

For better performance:

A larger dataset should be used (at least 100–500+ records).

Feature engineering and hyperparameter tuning can be applied.

Scaling and normalization can improve Logistic Regression.