LOAN ELIGIBILITY PREDICTION PROJECT

Project Overview:

This project aims to predict the eligibility of loan applicants based on various factors such as credit score, income, loan history, and other demographic information. Using a Random Forest Classifier, the model was trained to determine whether a loan applicant is likely to be approved or denied.

Dataset:

- The dataset consists of loan applicants' details including features like credit score, loan amount, income, and loan status.
- Target Variable: Loan Status (0 = Not Eligible, 1 = Eligible)
- Features: Credit Score, Income, Loan Amount, Employment Status, etc.

Objective:

To build a machine learning model that can predict the loan eligibility of applicants and to improve decision-making for loan approval processes.

Tools & Libraries:

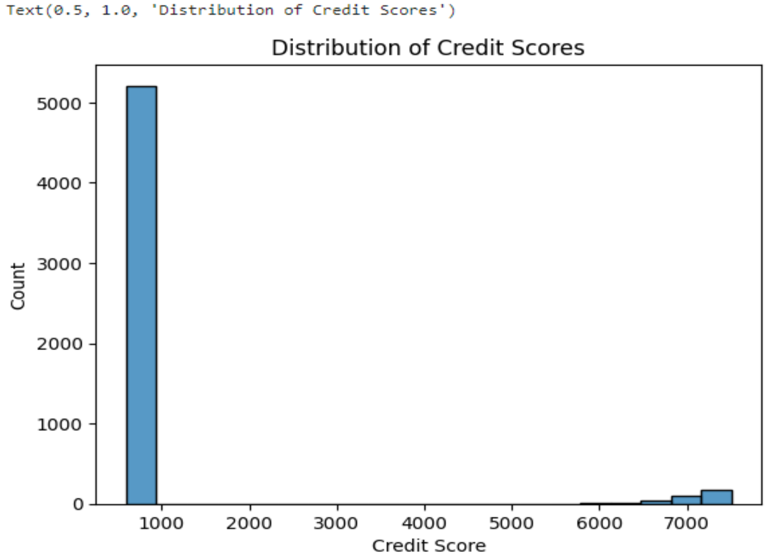
- Python for data analysis and machine learning
- Pandas for data manipulation
- Matplotlib and Seaborn for data visualization
- Scikit-learn for building and evaluating the machine learning model

Exploratory Data Analysis (EDA):

The plot shows the distribution of credit scores, helping to understand how applicants' credit scores vary.

```
[64]: sns.histplot(df['Credit Score'], bins = 20)
      plt.title('Distribution of Credit Scores')

[64]: Text(0.5, 1.0, 'Distribution of Credit Scores')
```

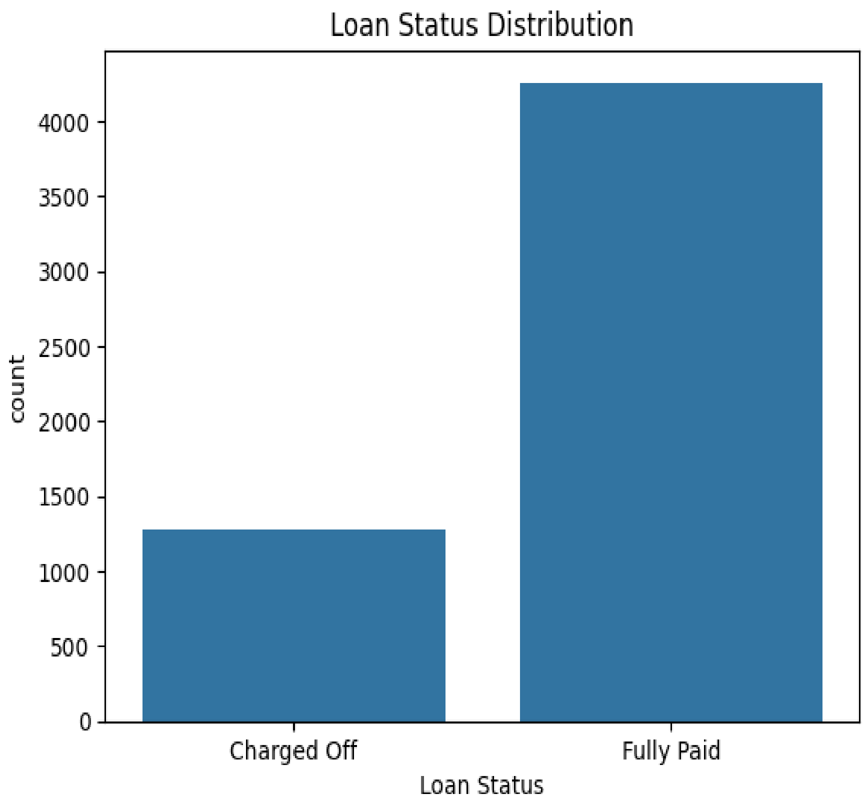


Loan Status Distribution:

This plot visualizes the proportion of applicants who are eligible and not eligible for loans.

[68]:
```python
sns.countplot(x = 'Loan Status', data = df)
plt.title('Loan Status Distribution')
```

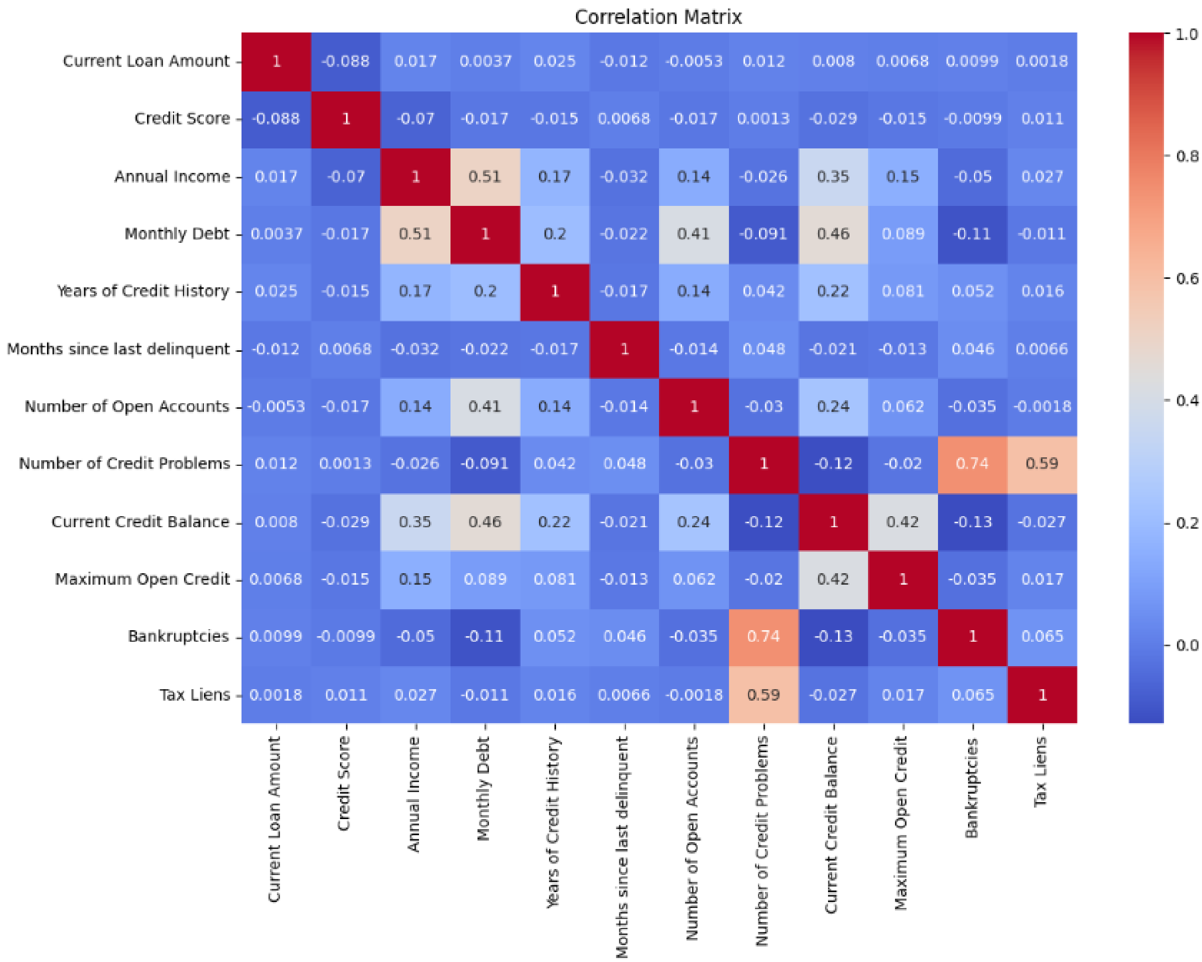
[68]: `Text(0.5, 1.0, 'Loan Status Distribution')`



The heatmap shows the correlation between numeric features, helping identify relationships between variables like credit score and loan amount.

[80]:

```
numeric_data = df.select_dtypes(include=['float64', 'int64'])
correlation_matrix = numeric_data.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



Accuracy: 81%

Class 0 (Not Eligible): F1-score of 0.35 (model struggles here)

Class 1 (Eligible): F1-score of 0.89 (model performs well here)

Mean CV score: 80.77%

```
model = RandomForestClassifier(random_state=42)
model.fit(X_train_encoded, y_train_encoded)

# Predictions and evaluation
y_pred = model.predict(X_test_encoded)
print(classification_report(y_test_encoded, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.22      0.35       257
           1       0.81      0.99      0.89       851

    accuracy                           0.81      1108
   macro avg       0.84      0.60      0.62      1108
weighted avg       0.82      0.81      0.76      1108
```

Model Tuning:

- **Best Parameters** (from hyperparameter tuning):
    - max_depth: None
    - min_samples_split: 5
    - n_estimators: 100

```
[96]: from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import LabelEncoder
import pandas as pd

# Encode the target variable
le = LabelEncoder()
y_train_encoded = le.fit_transform(y_train)

# Encode the features if there are categorical features
X_train_encoded = pd.get_dummies(X_train)

# Define parameter grid for tuning Random Forest model
param_grid = {
    'n_estimators': [50, 100],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5]
}

# Initialize Random Forest model
model = RandomForestClassifier(random_state=42)

# Perform GridSearchCV
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, scoring='accuracy', cv=3)
grid_search.fit(X_train_encoded, y_train_encoded)

# Print best parameters
print("Best parameters:", grid_search.best_params_)
```

```
Best parameters: {'max_depth': None, 'min_samples_split': 5, 'n_estimators': 100}
```

Results:

Classification Report:

Class	Precision	Recall	F1-Score	Support
0	0.88	0.22	0.35	257
1	0.81	0.99	0.89	851

- **Precision:** The model's ability to correctly predict each class.
  - Class 0 (not eligible): 88% precision, meaning 88% of predicted "not eligible" applicants were actually "not eligible."
  - Class 1 (eligible): 81% precision, meaning 81% of predicted "eligible" applicants were indeed eligible.
- **Recall:** The ability of the model to identify all relevant cases.

    - Class 0 (not eligible): 22% recall, meaning the model only captured 22% of actual "not eligible" applicants.
    - Class 1 (eligible): 99% recall, meaning the model was able to capture nearly all of the eligible applicants.
- **F1-Score:** The balance between precision and recall.
    - Class 0 has a low F1-score (0.35), while Class 1 has a high F1-score (0.89).
- **Overall Accuracy:** 81%. This means the model correctly predicted the loan status for 81% of the applicants.
- **Macro Average (average for both classes):** The model struggles with class 0, leading to a lower macro average for precision, recall, and F1-score.

Best Model Parameters (from Hyperparameter Tuning):

- **max_depth:** None (no limit on depth)
- **min_samples_split:** 5 (a split occurs only if a node has at least 5 samples)
- **n_estimators:** 100 (number of trees in the forest)

Cross-Validation Scores:

- Cross-validation scores: [0.8027, 0.8148, 0.8148, 0.7923, 0.8137]
    - These scores show a consistent performance across different data splits, with a range of ~79.2% to ~81.5%.
- Mean CV score: 80.77%
    - This confirms the model's performance is fairly stable and reliable across multiple folds.

Future Work:

1. **Handling Class Imbalance**: Techniques like oversampling, undersampling, or using SMOTE to balance the dataset.
2. **Model Optimization**: Explore other algorithms such as Gradient Boosting or XGBoost for better performance.
3. **Feature Engineering**: Additional features such as employment duration, loan history, or debt-to-income ratio might improve model accuracy.