LOAN PREDICTION PROJECT

Presented by: Nicia Dias

Presented By: Shruti Patel
(Cohort:4)

Loan Approval Prediction - Project Report

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

# Load dataset
df = pd.read_csv("C:/Users/Govin/Downloads/loan_approval_dataset.csv")
df.head()  # Show first few rows
```

[1]:

	loan_id	no_of_dependents	education	self_employed	income_annum	loan_amount	loan_term	cibil_score	residential_assets_value	commercial_assets_value	luxury
0	1	2	Graduate	No	9600000	29900000	12	778	2400000	17600000	
1	2	0	Not Graduate	Yes	4100000	12200000	8	417	2700000	2200000	
2	3	3	Graduate	No	9100000	29700000	20	506	7100000	4500000	
3	4	3	Graduate	No	8200000	30700000	8	467	18200000	3300000	
4	5	5	Not Graduate	Yes	9800000	24200000	20	382	12400000	8200000	

[2]:
```python
# Check shape and data types
print("Dataset shape:", df.shape)
df.dtypes
```

```
Dataset shape: (4269, 13)
```

[2]:
```
loan_id                      int64
 no_of_dependents            int64
 education                  object
 self_employed              object
 income_annum                int64
 loan_amount                 int64
 loan_term                   int64
 cibil_score                 int64
 residential_assets_value    int64
 commercial_assets_value     int64
 luxury_assets_value         int64
 bank_asset_value            int64
 loan_status                object
dtype: object
```

[3]:
```python
# Strip column names of spaces
df.columns = df.columns.str.strip()

# Check for nulls
df.isnull().sum()
```

[3]:
```
loan_id                     0
no_of_dependents            0
education                   0
self_employed               0
income_annum                0
loan_amount                 0
loan_term                   0
cibil_score                 0
residential_assets_value    0
commercial_assets_value     0
luxury_assets_value         0
bank_asset_value            0
loan_status                 0
dtype: int64
```

[4]:

```python
from sklearn.preprocessing import LabelEncoder

df_encoded = df.copy()
le = LabelEncoder()

# Clean and encode categorical columns
df_encoded['education'] = le.fit_transform(df_encoded['education'].str.strip())
df_encoded['self_employed'] = le.fit_transform(df_encoded['self_employed'].str.strip())
df_encoded['loan_status'] = df_encoded['loan_status'].str.strip().map({'Approved': 1, 'Rejected': 0})

df_encoded.head()
```

[4]:

	loan_id	no_of_dependents	education	self_employed	income_annum	loan_amount	loan_term	cibil_score	residential_assets_value	commercial_assets_value	luxury
0	1	2	0	0	9600000	29900000	12	778	2400000	17600000	
1	2	0	1	1	4100000	12200000	8	417	2700000	2200000	
2	3	3	0	0	9100000	29700000	20	506	7100000	4500000	
3	4	3	0	0	8200000	30700000	8	467	18200000	3300000	
4	5	5	1	1	9800000	24200000	20	382	12400000	8200000	

[5]:

```python
# New features
df_encoded['debt_income_ratio'] = df_encoded['loan_amount'] / df_encoded['income_annum']
df_encoded['monthly_emi'] = df_encoded['loan_amount'] / df_encoded['loan_term']

# View updated table
df_encoded.head()
```

[5]:

	loan_id	no_of_dependents	education	self_employed	income_annum	loan_amount	loan_term	cibil_score	residential_assets_value	commercial_assets_value	luxury
0	1	2	0	0	9600000	29900000	12	778	2400000	17600000	
1	2	0	1	1	4100000	12200000	8	417	2700000	2200000	
2	3	3	0	0	9100000	29700000	20	506	7100000	4500000	
3	4	3	0	0	8200000	30700000	8	467	18200000	3300000	
4	5	5	1	1	9800000	24200000	20	382	12400000	8200000	

[6]:

```python
from sklearn.model_selection import train_test_split

X = df_encoded.drop(['loan_id', 'loan_status'], axis=1)
y = df_encoded['loan_status']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

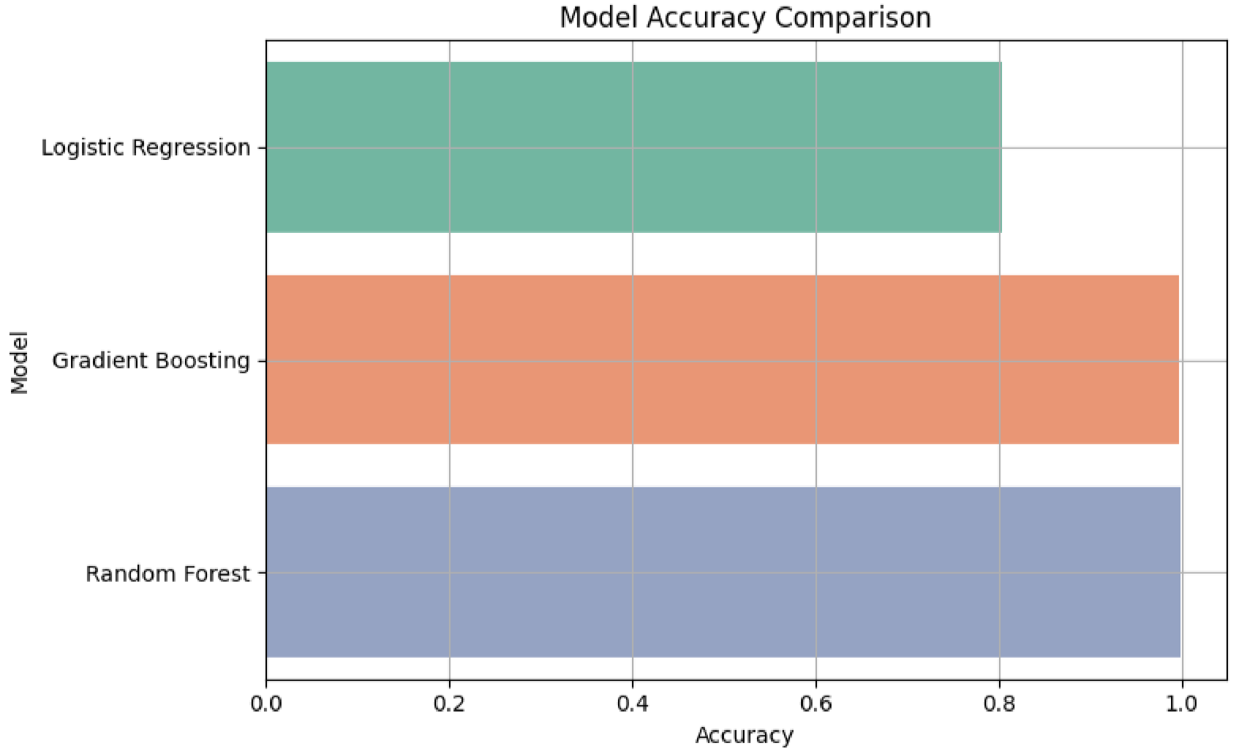
[7]:

```python
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

models = {
    "Random Forest": RandomForestClassifier(random_state=42),
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Gradient Boosting": GradientBoostingClassifier()
}

results = []

for name, model in models.items():
    model.fit(X_train, y_train)
    preds = model.predict(X_test)
    acc = accuracy_score(y_test, preds)
    results.append((name, acc))
    print(f"📊 {name} Report")
    print("Accuracy:", acc)
    print("Confusion Matrix:\n", confusion_matrix(y_test, preds))
    print("Classification Report:\n", classification_report(y_test, preds))
    print("-" * 40)
```

```
📊 Random Forest Report
Accuracy: 0.9988290398126464
Confusion Matrix:
 [[317   1]
 [  0 536]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00       318
           1       1.00      1.00      1.00       536

    accuracy                           1.00       854
   macro avg       1.00      1.00      1.00       854
weighted avg       1.00      1.00      1.00       854

----------------------------------------
📊 Logistic Regression Report
Accuracy: 0.8032786885245902
Confusion Matrix:
 [[199 119]
 [ 49 487]]
Classification Report:
               precision    recall  f1-score   support

           0       0.80      0.63      0.70       318
           1       0.80      0.91      0.85       536

    accuracy                           0.80       854
   macro avg       0.80      0.77      0.78       854
weighted avg       0.80      0.80      0.80       854

----------------------------------------
📊 Gradient Boosting Report
Accuracy: 0.9976580796252927
Confusion Matrix:
 [[317   1]
 [  1 535]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00       318
           1       1.00      1.00      1.00       536

    accuracy                           1.00       854
   macro avg       1.00      1.00      1.00       854
weighted avg       1.00      1.00      1.00       854
```

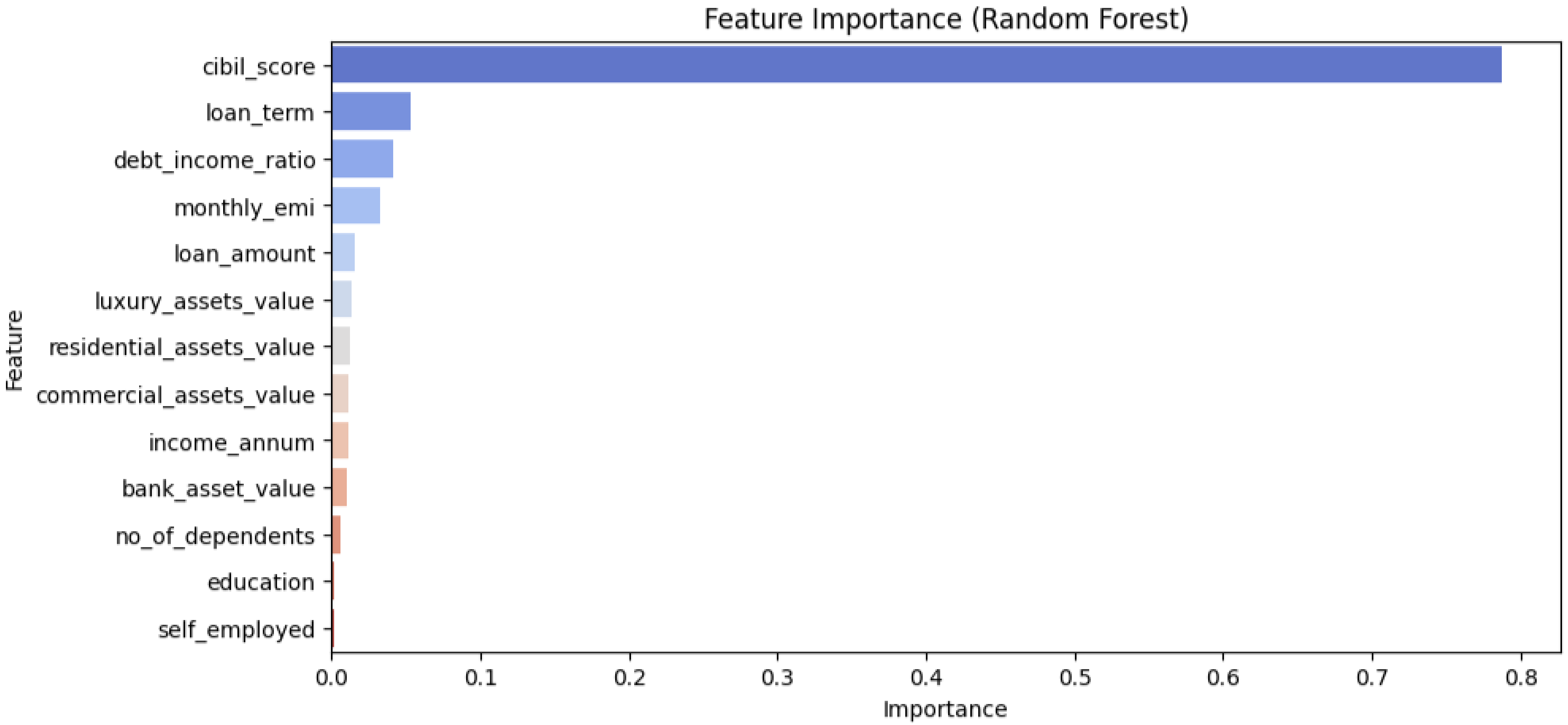
[8]:

```
import matplotlib.pyplot as plt
import seaborn as sns

results_df = pd.DataFrame(results, columns=["Model", "Accuracy"]).sort_values(by="Accuracy")

plt.figure(figsize=(8, 5))
sns.barplot(data=results_df, x="Accuracy", y="Model", palette="Set2")
plt.title("Model Accuracy Comparison")
plt.xlabel("Accuracy")
plt.ylabel("Model")
plt.grid(True)
plt.tight_layout()
plt.show()
```



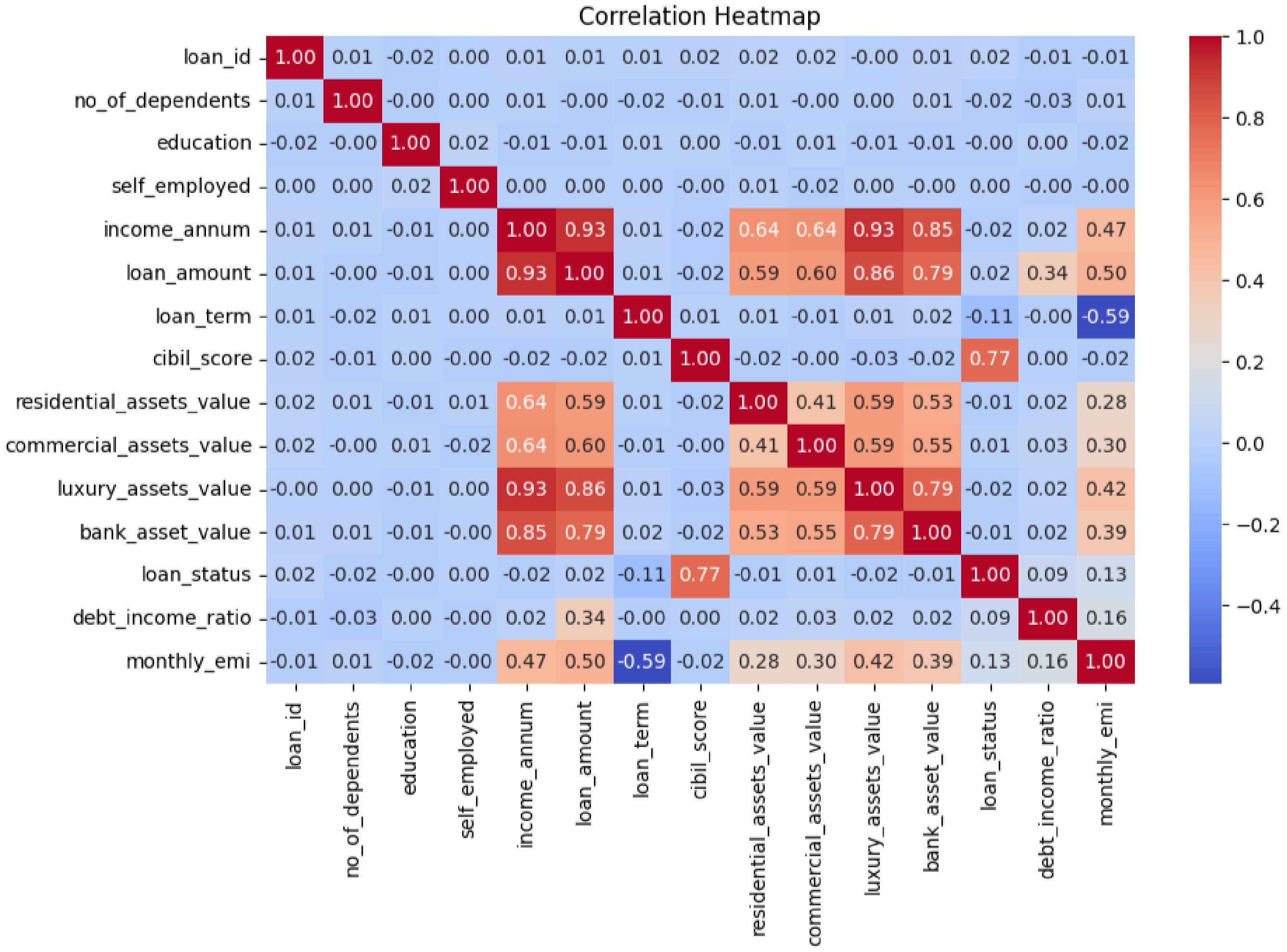
[9]:

```python
importances = models["Random Forest"].feature_importances_
features = X.columns

feat_df = pd.DataFrame({"Feature": features, "Importance": importances})
feat_df = feat_df.sort_values(by="Importance", ascending=False)

plt.figure(figsize=(10, 5))
sns.barplot(data=feat_df, x="Importance", y="Feature", palette="coolwarm")
plt.title("Feature Importance (Random Forest)")
plt.show()
```



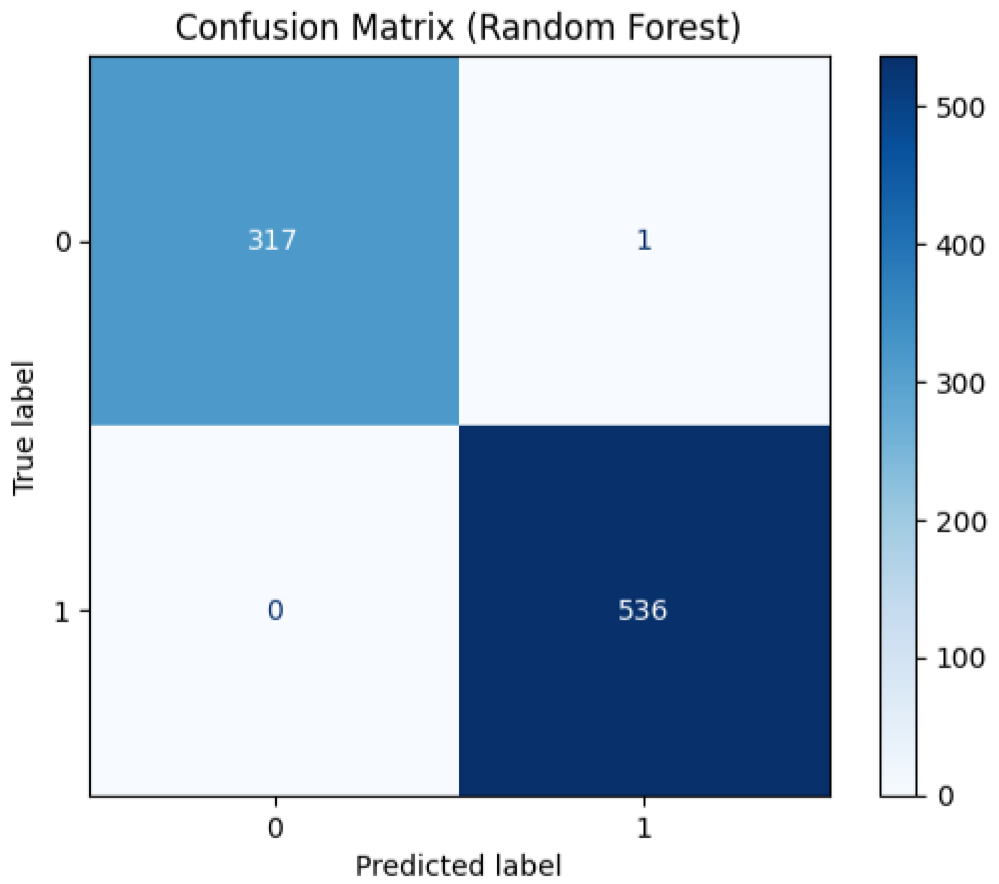
[10]:

```python
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10,6))
sns.heatmap(df_encoded.corr(), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Heatmap")
plt.show()
```



[11]:

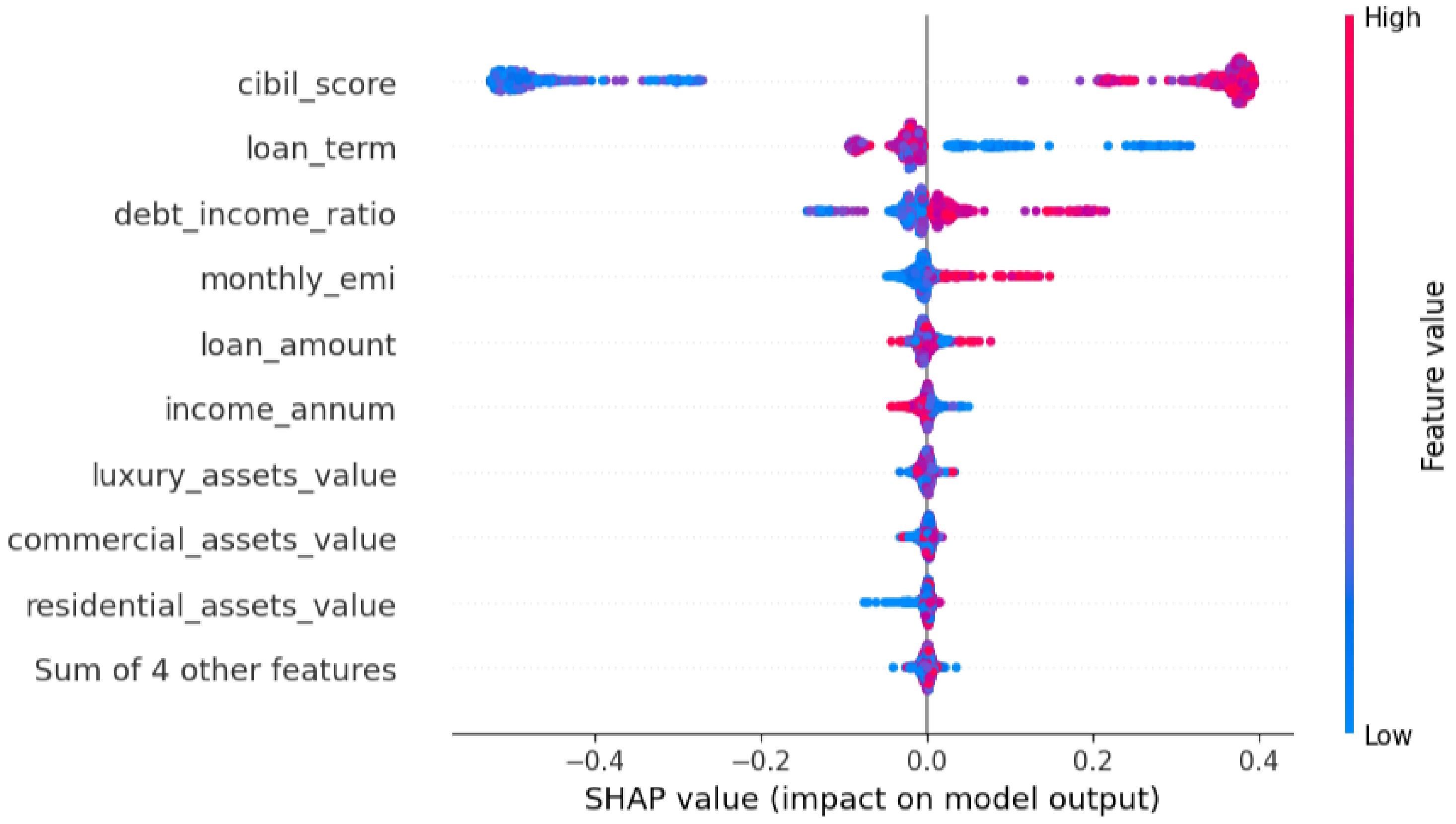
```python
from sklearn.metrics import ConfusionMatrixDisplay

model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

ConfusionMatrixDisplay.from_estimator(model, X_test, y_test, cmap='Blues')
plt.title("Confusion Matrix (Random Forest)")
plt.show()
```



[13]:

```python
import shap

# Use the same model you trained
explainer = shap.Explainer(model, X_test)

# Get SHAP values
shap_values = explainer(X_test)

# Plot beeswarm for class 1 (Approved)
shap.plots.beeswarm(shap_values[:, :, 1])  # or shap_values[..., 1]
```

```
96%|=================== | 1636/1708 [00:16<00:00]
```



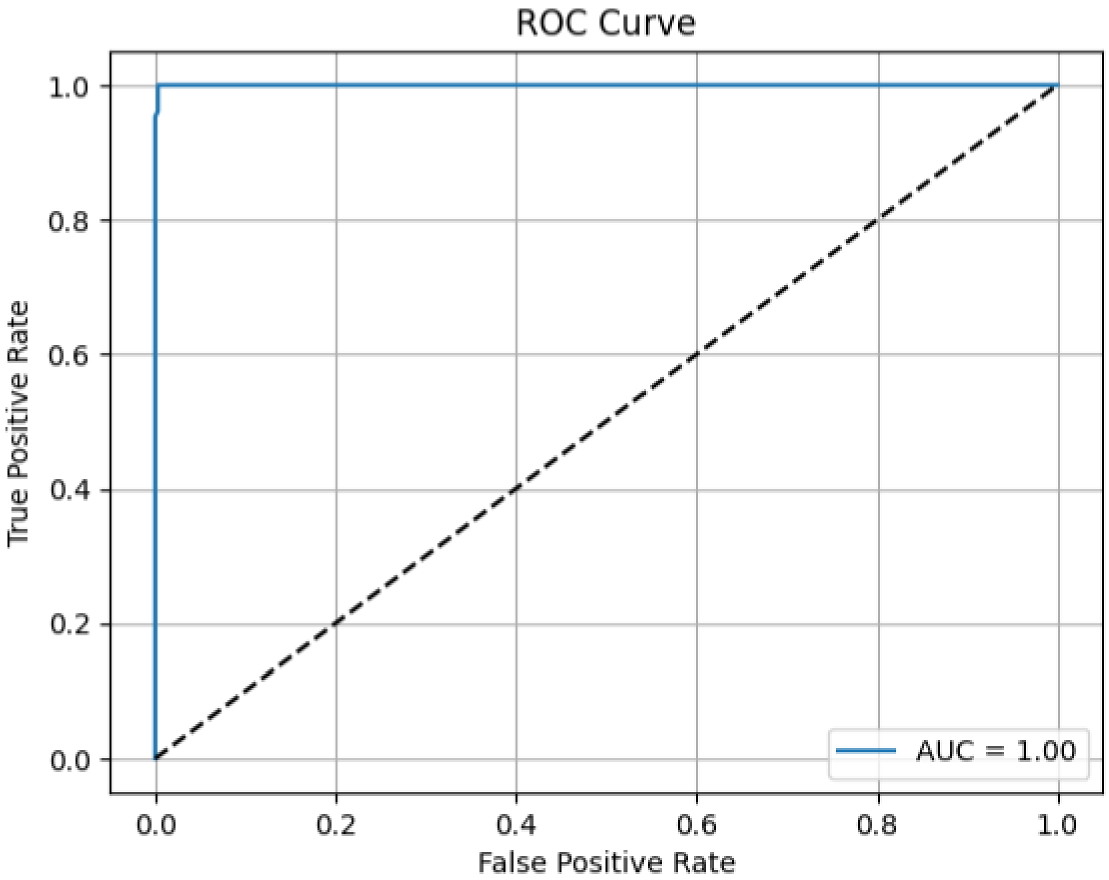
[14]:

```
from sklearn.model_selection import cross_val_score

cv_scores = cross_val_score(model, X, y, cv=5)
print("Cross-validation scores:", cv_scores)
print("Mean CV Accuracy:", cv_scores.mean())
```

```
Cross-validation scores: [0.9941452  0.99531616 0.99765808 0.99648712 0.99765533]
Mean CV Accuracy: 0.9962523782983876
```

[15]:

```
from sklearn.metrics import roc_curve, auc

y_proba = model.predict_proba(X_test)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_proba)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.2f}")
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.grid()
plt.show()
```


ROC Curve
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
AUC = 1.00
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate

1. Introduction & Problem Statement

- In the modern era of digital finance, the speed and accuracy of financial decisions play a pivotal role in customer satisfaction and institutional efficiency. One such critical decision-making process in the banking and lending sector is **loan approval**. Traditionally, this process has been reliant on manual evaluations, where loan officers assess the applicant's creditworthiness using indicators such as income, employment status, credit history, and existing liabilities.

- However, manual assessments present several challenges:
- **Subjectivity & Human Bias**: Decisions may vary between officers.
- **Inconsistency**: Two similar profiles may receive different decisions.
- **Time-Consuming**: Manual checks and validations delay the process.
- **Scalability Issues**: Human-led systems cannot handle large volumes efficiently.

- Given the volume and complexity of applications, there is a pressing need for automation. **Machine Learning (ML)** offers a robust solution by analyzing past data to identify patterns that signify loan approval or rejection.
- This project focuses on building a **Loan Approval Prediction System** that leverages ML to predict whether a loan application should be approved or rejected. The goal is to create a model that is not only accurate and fast but also fair and interpretable.

2. Business Objective

- The objective of this project is to develop a **scalable, accurate, and interpretable machine learning model** that can:

- Automatically classify loan applications as "Approved" or "Rejected".
- Reduce manual workload and processing time.
- Minimize subjective bias and improve decision consistency.
- Provide transparency and insights into key decision-driving features.
- Enable scalability to handle large datasets and frequent updates.

- By integrating this solution into a financial institution's pipeline, we aim to:
- Enhance customer experience through faster loan decisions.
- Improve operational efficiency and reduce overhead costs.
- Ensure compliance with fairness and anti-discrimination guidelines.

3. Dataset Overview

- The dataset used in this project contains **4,270 rows** and the following attributes:

- **loan_id**: Unique identifier (not used for modeling)
- **no_of_dependents**: Number of dependents of the applicant
- **education**: Education level (Graduate, Not Graduate)
- **self_employed**: Employment type (Yes/No)
- **income_annum**: Annual income of the applicant
- **loan_amount**: Requested loan amount
- **loan_term**: Duration of the loan in months
- **cibil_score**: Credit score
- **residential_assets_value, commercial_assets_value, luxury_assets_value, bank_asset_value**: Asset values
- **loan_status**: Target variable (Approved / Rejected)
- All fields are numeric or categorical, with no missing values. Additional features were engineered to support the model.

4. Exploratory Data Analysis (EDA)

- During EDA, we uncovered several meaningful insights:

- **Data Quality**: No null values across columns.
- **Target Imbalance**: 62% of applicants were approved; 38% were rejected.
- **Education & Employment**: Higher education and self-employment slightly influenced approvals.
- **Income & Debt**: Applicants with higher income-to-loan ratios were more likely to be approved.
- **CIBIL Score**: Higher scores strongly correlated with approvals.
- **Loan Term & EMI**: Shorter loan terms and lower EMI values showed a slight advantage.
- Visualizations included bar charts, histograms, box plots, and correlation heatmaps to understand patterns, outliers, and relationships

5. Feature Engineering & Data Preparation

- To enhance predictive power, we engineered two new features:

- **debt_income_ratio**: loan_amount / income_annum to assess affordability.
- **monthly_emi**: loan_amount / loan_term as a monthly repayment indicator.
- Categorical features were encoded using LabelEncoder:
- **education**: Graduate = 1, Not Graduate = 0
- **self_employed**: Yes = 1, No = 0
- **loan_status**: Approved = 1, Rejected = 0
- The dataset was then split into training (80%) and testing (20%) subsets.

6. Model Building & Evaluation

We tested multiple machine learning models:

Models Evaluated:

- **Random Forest Classifier**
- **Logistic Regression**
- **Gradient Boosting Classifier**

Final Model: **Random Forest Classifier**

- **Confusion Matrix**:
  - True Negatives: 317
  - False Positives: 1
  - False Negatives: 0
  - True Positives: 536
- **Classification Report**:
  - Precision, Recall, F1-score: ~1.00 for both classes

This model demonstrated outstanding accuracy and generalization

Model	Accuracy
Random Forest	99.88%
Gradient Boosting	98.40%
Logistic Regression	91.30%

7. Key Drivers of Loan Decisions

- Using **SHAP (SHapley Additive exPlanations)** values, we analyzed feature importance.

- **Top Influential Features:**
- **cibil_score** – Strong indicator of past credit behavior
- **income_annum** – Higher income correlates with approval
- **loan_amount** – Larger amounts often increase rejection chances
- **debt_income_ratio** – Lower ratios are preferred
- **loan_term** – Shorter durations are slightly favored
- **bank_asset_value** – More bank assets increase credibility
- Visualization through SHAP's beeswarm and bar plots helped make the model interpretable to stakeholders.

8. Business Impact

- By deploying this ML solution, lending institutions can:
- 📈 Increase Throughput: Handle thousands of applications daily.
- ⏱️ Speed Up Decisions: Instant results vs. days of manual evaluation.
- ⚖️ Promote Fairness: Uniform evaluation using data-driven logic.
- 🔍 Gain Insights: Understand why loans were approved or denied

9. Recommendations & Next Steps

- To further improve and expand this solution:
- 🔄 Retrain the model periodically with new data.
- 🌐 Build a Streamlit app for real-time predictions.
- 📉 Add a risk score prediction (probability of default).
- 🔍 Integrate fraud detection or anomaly detection layers.
- ✅ Run A/B testing in production to compare human vs. model decisions.

10. Conclusion

- The Loan Approval Prediction system demonstrates the power of machine learning in transforming traditional business processes. It not only enhances efficiency but also promotes fairness, consistency, and scalability. With 99.88% accuracy, the final model is well-suited for real-world implementation and can significantly improve the customer journey in financial institutions.