# Loan Approval Prediction - Final Report

## 1. Problem Statement

Financial institutions receive large volumes of loan applications and must decide whether to approve or reject each one. Manual review is slow and may introduce inconsistent or biased decisions. This project builds an automated, data-driven machine learning pipeline to predict loan approval (approved / rejected) using applicant demographic and financial features.

## 2. Dataset

The dataset contains applicant-level records with demographic, credit and financial attributes. Typical columns include:
- loan_id, no_of_dependents, education, self_employed, income_annum, loan_amount, loan_term, cibil_score
- residential_assets_value, commercial_assets_value, luxury_assets_value, bank_asset_value, loan_status

Notes: The notebook includes a robust file loader that removes row-index prefixes and normalizes column names.

## 3. Methodology

The notebook implements a reproducible end-to-end pipeline with the following stages:
1. Smart dataset loading (Tkinter file picker; fallback path persistence)
2. Raw-file cleaning (removes leading row numbers which broke parsers)
3. Target detection and robust mapping of loan_status -> {0,1}
4. Feature engineering (example: debt-to-income estimate)
5. Preprocessing pipelines for numerical and categorical features (imputation, scaling, one-hot encoding)
6. Model training: Logistic Regression and Random Forest (pipelines)
7. Evaluation: Accuracy, F1, ROC-AUC, and confusion matrices
8. Best-model selection and export (best_loan_model.joblib)
9. Basic fairness check (disparate impact) if protected attributes exist.

## 4. Preprocessing & Features

Numeric columns are imputed with the median and standardized. Categorical columns are imputed with the most frequent value and one-hot encoded (the notebook uses scikit-learn's OneHotEncoder with sparse_output=False for compatibility). A sample engineered feature 'dti_estimate' (loan_amount / income_annum) is included when relevant columns exist.

## 5. Modeling & Evaluation

Two baseline models are trained in the notebook:
- Logistic Regression
- Random Forest

Each model is wrapped in a scikit-learn Pipeline that includes preprocessing. Models are evaluated on a held-out test set using the following metrics: Accuracy, F1-score, ROC-AUC, and confusion matrix. The

notebook selects the best model by prioritizing F1-score and AUC as a tiebreaker. The trained model is saved as 'best_loan_model.joblib' in the notebook folder.

## 6. How to run (local Jupyter)

1. Open the notebook file 'Loan_Approval_Prediction_final_v2.ipynb' in Jupyter Notebook or VS Code.
2. Install dependencies (if needed):
   pip install -r requirements.txt
3. Run cells from top to bottom. The first cell will open a file picker to select the dataset; the selection is remembered in 'dataset_path.txt'.
4. After training, the best model and evaluation outputs will be available in the notebook and the model file will be saved.

## 7. Files produced / included

- Loan_Approval_Prediction_final_v2.ipynb : Jupyter notebook with full pipeline
- best_loan_model.joblib : Saved best model (created after running the notebook)
- README.md : Project instructions
- requirements.txt : Python dependencies
- Loan_Approval_Report_final.pdf : This report

If your dataset is sensitive, avoid committing it to public repositories; include a README explaining how to obtain it instead.

## 8. Recommendations & Future Work

- Consider adding XGBoost or LightGBM for improved accuracy.
- Add SMOTE or class-weighting if classes are imbalanced.
- Implement SHAP explanations for model interpretability.
- Deploy the model via a simple FastAPI or Flask service for real-time predictions.
- Schedule periodic retraining to adapt to changing borrower behavior.

## 9. Notes & Contact

This report was generated automatically by the project pipeline creator. After running the notebook you can capture exact evaluation numbers (Accuracy, F1, ROC-AUC) and paste them into this report before final submission.

For help running the notebook or adding features (SHAP, XGBoost, API), request specific additions in the conversation.