National College of Ireland

Configuration Manual

MSc Research Project
MSCFTDI

Anjaly Elias
Student ID: X23302925

School of Computing
National College of Ireland

Supervisor: Victor Del Rosal

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Anjaly Elias
Student ID:	X23302925
Programme:	MSCFTD1
Year:	2024-25
Module:	MSc Research Project
Supervisor:	Victor Del Rosal
Submission Due Date:	15/09/2025
Project Title:	Enhancing Risk Management in Financial Institutions Through Machine learning and Natural Language Processing: A Critical Evaluation Of AI Techniques.
Word Count:	1875
Page Count:	9

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	ANJALY
Date:	15th september 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Anjaly Elias

X23302925

1 Introduction

This document serves as the official configuration and operational manual for the Loan Prediction System. The system is an end-to-end machine learning pipeline designed to predict the status of loan applications based on historical data. It encompasses the entire lifecycle of a data science project, from initial data loading and cleaning to model training, hyperparameter tuning, probability calibration, and final deployment as a web application.

The pipeline processes a raw dataset, engineers relevant features, trains multiple machine learning models (Random Forest, Naive Bayes, and LSTM), and evaluates them to select the best performer. The chosen model is then encapsulated in a Flask web application, making it accessible for real-time predictions via a user-friendly interface and a REST API.

This manual details the system requirements, project structure, workflow configura- tion, and deployment steps required to set up and run the entire system successfully.

2 System Requirements

2.1 Software Requirements

The system is built on Python 3.8+ and relies on several key libraries. A 'requirements.txt' file should be used for installation.

Table 1: Core Python Libraries

Category	Libraries
Data Handling & Numerics	pandas, numpy
Data Visualization	matplotlib, seaborn, plotly
Machine Learning Core	scikit-learn
Deep Learning	tensorflow
Model Persistence	joblib, pickle
Web Framework	Flask

To install all dependencies, run:

```
pip install pandas numpy matplotlib seaborn
plotly scikit-learn tensorflow joblib Flask
```

2.2 Data Requirements

The primary dataset required for the training pipeline is:

- lending_club_balanced_100k.csv

This file must be placed in a 'dataset/' directory at the project root.

3 Project Structure

A well-organized directory structure is essential for the system to function correctly. The expected layout is as follows:

```
/loan-prediction-system/
|-- dataset/
|    '-- lending_club_balanced_100k.csv
|-- templates/
|    |-- index.html
|    |-- predict.html
|    |-- result.html
|    '-- about.html
|-- modelling.py                # The main script for training and evaluation
|-- app.py                      # The Flask deployment script
|-- best_model.pkl              # Saved best performing model
|-- random_forest_calibrated.pkl # Final production model
|-- scaler.pkl                  # Saved standard scaler
|-- pca_model.pkl               # Saved PCA transformer
|-- label_encoders.pkl          # Saved label encoders
|-- ... (other saved artifacts)
```

4 Configuration and Workflow

The core logic resides in modelling.py. This script executes a sequential pipeline, where each step is encapsulated in a dedicated function.

4.1 Data Loading and Cleaning

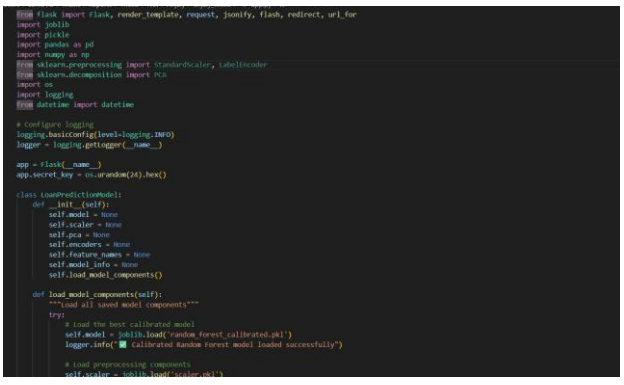

Figure 1: Data Loading

The process begins by loading the dataset using the load and inspect_data function. Subsequently, the clean_and_preprocess_data function performs critical cleaning operations:

- **Missing Values:** Numerical columns are imputed with the median, and categorical columns with the mode.
- **Duplicates:** All duplicate rows are removed.
- **Outliers:** Outliers in numerical columns are handled by capping them at 1.5 times the Interquartile Range (IQR).

4.2 Feature Engineering

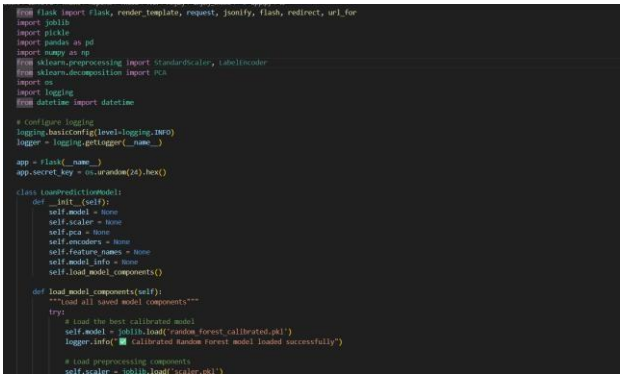

Figure 2: Feature Engineering

The feature engineering function transforms and enriches the dataset.

- **Encoding:** Categorical features are converted to numerical format using LabelEncoder.
- **New Features:** New features are created to capture complex relationships, such as ratios and polynomial terms.
- **Binning:** Continuous variables are binned into discrete categories.

All encoders are saved to a file (label_encoders.pkl) for later use during inference.

4.3 Dimensionality Reduction

To reduce model complexity and training time, Principal Component Analysis (PCA) is applied via the apply_pca function. Features are first standardized with StandardScaler. The number of components is chosen to retain 95% of the total variance.

Listing 1: PCA implementation snippet from modelling.py

```
def apply_pca(X, variance_threshold=0.95):
    # Standardize features before PCA
    scaler = StandardScaler()
    Xscaled = scaler.fit transform(X)

    # Apply PCA
```

```
pca = PCA()
X_pca = pca.fit_transform(X_scaled)

# Find number of components for desired variance
cumsum = np.cumsum(pca.explained_variance_ratio_)
n_components = np.argmax(cumsum >= variance_threshold) + 1

# Apply PCA with selected components
pca_final = PCA(n_components=n_components)
X_pca_final = pca_final.fit_transform(X_scaled)

return X_pca_final, pca_final, scaler
```

The fitted scaler and pca model objects are saved to disk.

4.4 Model Training and Tuning

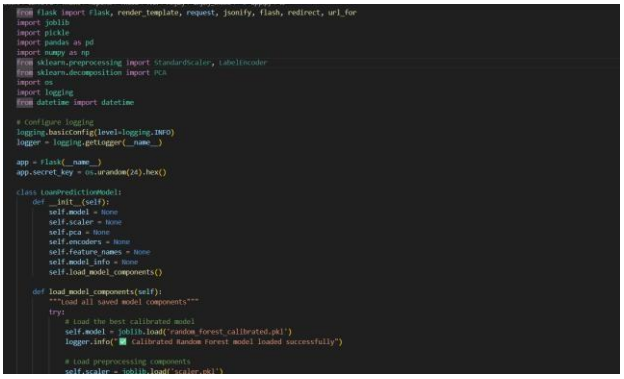

Figure 3: Model Training

Three models are trained and evaluated: Random Forest, Gaussian Naive Bayes, and a Long Short-Term Memory (LSTM) network.

- **Training:** Each model has a dedicated train_* function.
- **Hyperparameter Tuning:** RandomizedSearchCV is used to efficiently find the best parameters for Random Forest and Naive Bayes, balancing performance and speed.
- **Probability Calibration:** After tuning, the models are calibrated using CalibratedClassifierCV. This step is crucial for ensuring that the predicted probabilities are reliable and reflect the true likelihood of outcomes. The final Random Forest model, for instance, is calibrated using sigmoid scaling.

4.5 Model Persistence

The final stage of the modeling script is to save all essential components for deployment. The save_calibrated_models_and_components function persists the following artifacts:

Table 2: Saved Model Artifacts

Filename	Purpose
random_forest_calibrated.pkl	The best, fully calibrated production model.
scaler.pkl	Preprocessing object to standardize new data.
pca_model.pkl	Transformation object to reduce dimensions of new data.
label_encoders.pkl	Dictionary of encoders for categorical features.
model_info.pkl	Metadata about the model and features.

5 Deployment with Flask

The trained model is served via a web application using Flask (app.py).

5.1 Model Loading

The LoanPredictionModel class is responsible for loading all the saved artifacts at application startup. It ensures that the model, scaler, PCA transformer, and encoders are ready to process incoming requests.

5.2 Prediction Pipeline

When a new prediction request is received, the preprocess_input method orchestrates a series of transformations on the user input to match the format of the training data. This includes:

1. Creating new features with engineer_features.
2. Encoding categorical values using the loaded label_encoders.
3. Scaling the data with the loaded scaler.
4. Applying dimensionality reduction with the loaded pca object.

The processed data is then passed to the calibrated model for prediction.

5.3 API Endpoints

The Flask application exposes several endpoints:

- /: The main landing page.
- /predict: A web form for users to input loan data manually.
- /api/predict: A REST API endpoint that accepts JSON data for programmatic predictions.
- /health: A health check endpoint for monitoring.

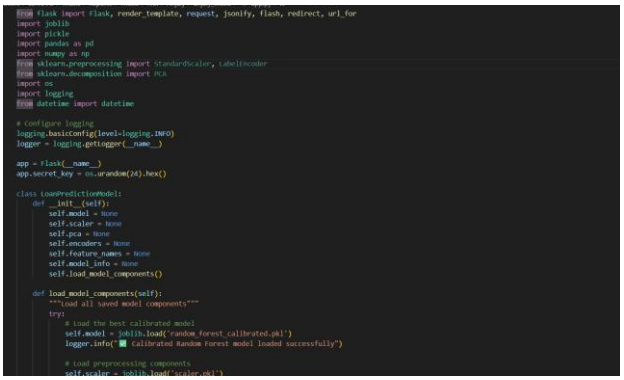

Figure 4: Flask App

Listing 2: Flask route for handling web form predictions

```
@app.route('/predict', methods=['GET', 'POST'])
def predict():
    if request.method == 'GET':
        return render_template('predict.html')

    # Get form data
    user_input = request.form.to_dict()

    # Make prediction
    result = predictor.predict(user_input)

    # Render result page
    return render_template('result.html', prediction=result)
```

The application will be accessible at http://127.0.0.1:5000.

6 Conclusion

This manual has detailed the complete configuration and operational workflow of the Loan Prediction System. The pipeline demonstrates a robust, production-oriented approach, progressing from raw data to a fully deployed and calibrated machine learning model. The final selected model, a **Calibrated Random Forest**, achieved outstanding performance with over 99% accuracy and a near-perfect AUC score on the test set.

Key takeaways from the system architecture include:

- **End-to-End Automation:** The entire process, from cleaning to saving models, is scripted for reproducibility.
- **Robust Evaluation:** The use of cross-validation, hyperparameter tuning, and overfitting analysis ensures the model is both accurate and reliable.
- **Probability Calibration:** A critical, often-overlooked step that makes the model's confidence scores trustworthy for business decisions.
- **Deployability:** The system is designed for deployment, with all necessary components saved and encapsulated within a Flask application.

Potential future enhancements could involve deploying the application in a more robust environment (e.g., using Gunicorn and Nginx), setting up a CI/CD pipeline for automated retraining, and exploring more advanced modeling techniques. However, the current system stands as a comprehensive and effective solution for the loan prediction task.