High Level Design (HLD)

Credit Card Default Prediction

Revision Number: 1.0

Last date of revision: 25/09/2023

# Document Version Control

| Date Issued | Version | Description | Author |
| --- | --- | --- | --- |
| 25/09/2023 | 1 | Initial HLD – v1.0 | Abraham Audu |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Table of Contents**

[Document Version Control 2](#_heading=h.gjdgxs)

[**Abstract 4**](#_heading=)

[**1. Introduction 5**](#_heading=)

[1.1 Why this High-Level Design Document? 5](#_heading=)

[1.2 Scope 5](#_heading=h.d1qeboog8101)

[**2. General Description 5**](#_heading=h.ggkmsajlo83h)

[2.1 Product Perspective 5](#_heading=h.47rs17phkyud)

[2.2 Problem Statement 5](#_heading=h.2m4jh7xphb00)

[2.3 Proposed Solution 5](#_heading=h.a24luzxqjj4y)

[2.4 Further Improvements 6](#_heading=h.9tipoxtfyl22)

[2.5 Technical Requirement 6](#_heading=h.6e6nbndd0z0)

[2.6 Data Requirements 6](#_heading=h.u8e5flsgmxuv)

[2.7 Tools Used 6](#_heading=h.eeubfnt9ay1j)

[2.8 Constraints 6](#_heading=h.uq8kg0k4hxoi)

[2.9 Assumptions 6](#_heading=h.jy1d8hqituqs)

[**3. Design Details 7**](#_heading=h.4o154eianxnu)

[3.1a Development Process Flow 7](#_heading=h.2izcvvqfv1k2)

[3.1b Deployment Process Flow 7](#_heading=h.gegq5e7jn42o)

[3.2 Event Log 7](#_heading=h.7vujpp5x3o0f)

[3.3 Error Handling 8](#_heading=h.754hode35qq)

[**4. Performance 8**](#_heading=h.83j129w5xneh)

[4.1 Reusability 8](#_heading=h.k9trcoggc87)

[4.2 Application Compatibility 8](#_heading=h.va0xe28dsx4t)

[4.3 Resource Utilisation 8](#_heading=h.yci7e8mbhjs5)

[4.4 Deployment 8](#_heading=h.rwu1iljkevt8)

[**5. Conclusion 8**](#_heading=h.j1a2hkradlcn)

# Abstract

Banking institutions offer credit to customers as one of their major businesses and this comes with some associated credit risk. WIth millions of customers, it is important to have an automated system which can filter through customer information and financial history to determine the probability that a card owner will default in order to limit their access to current or future credit services and minimise risk to the issuing organisation. This project covers the implementation of a machine learning based platform which is able to take a credit card owner’s information and financial history and predict the probability that they will be credit defaulters.

# Introduction

## Why this High-Level Design Document?

The purpose of this High-Level Design Document is to add necessary detail to this Credit Card Default Prediction project to serve as a suitable background for the coding aspect of this project. This document also captures conceptual design propositions to guide code requirements and interaction between modules.

This document will:

* Present design aspects and define them in detail
* Describe the user interface being implemented
* Describe the development hardware and software
* Include design features and the architecture of the project
* List and describe the non-functional attributes such as:
  + Security
  + Reliability
  + Maintainability
  + Portability
  + Reusability
  + Application compatibility
  + Resource utilisation
  + Serviceability

## 1.2 Scope

This document covers the structure of the system, including application architecture, application flow and technology architecture. Non-technical to mildly technical terms have been used in order to facilitate easy understanding of the contents of this document.

# 2. General Description

## 2.1 Product Perspective

The Credit Card Default Prediction solution is a machine learning-based classification model which will help us determine the probability of a credit card owner with some use history to default on their card payments.

## 2.2 Problem Statement

To create a machine learning solution which is able to use credit card user information and payment history to determine the probability that a card owner will be a defaulter in monthly payments.

## 2.3 Proposed Solution

The solution proposed here is a simple web app which allows the user to upload a credit card owner’s historical data and receive a probability score that the said credit card owner will default on future monthly payments.

## 2.4 Further Improvements

The solution can utilise multiple models to ensure the fairest judgement based on historical data since finance is a sensitive aspect.

## 2.5 Technical Requirement

The primary requirement is that the user is able to have an interface to enter credit card owner details and payment history and obtain the probability that the said user will default on their monthly payments.

## 2.6 Data Requirements

Data used to develop the models which power the platform is in CSV (comma separated values) format and the final solution also accepts CSV data to make inferences.

To develop the models, data with thousands of rows of records is needed to ensure that the model has access to a wide range of scenarios and also to ensure that accounting for class imbalance – which is characteristic of financial data – does not drastically affect the quality of the processed data.

## 2.7 Tools Used

Python and frameworks such as NumPy, Pandas and Scikit-learn were used in the model development phase. Other tools used are:

* Visual Studio Code as IDE
* Matplotlib and Seaborn for visualisation
* FastAPI for backend development
* Streamlit for frontend development
* Git/GitHub for version control and remote code backup
* PyTest for unit testing

## 2.8 Constraints

Friendly user-interface requiring no knowledge of the underlying processes to obtain desired results

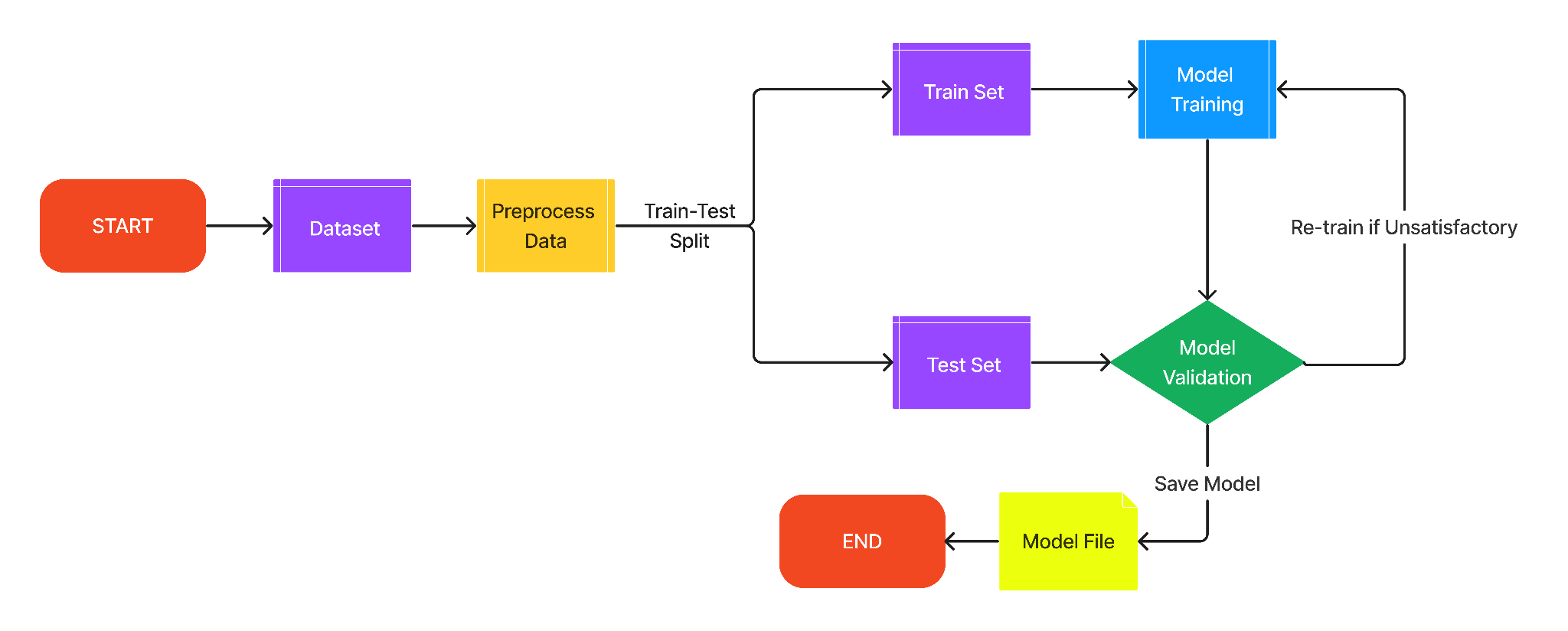
## 2.9 Assumptions

* The objective of the project remains as described in section 2.2
* Machine learning approach is required to achieve desired result
* The user input follows the same format as training data

# 3. Design Details

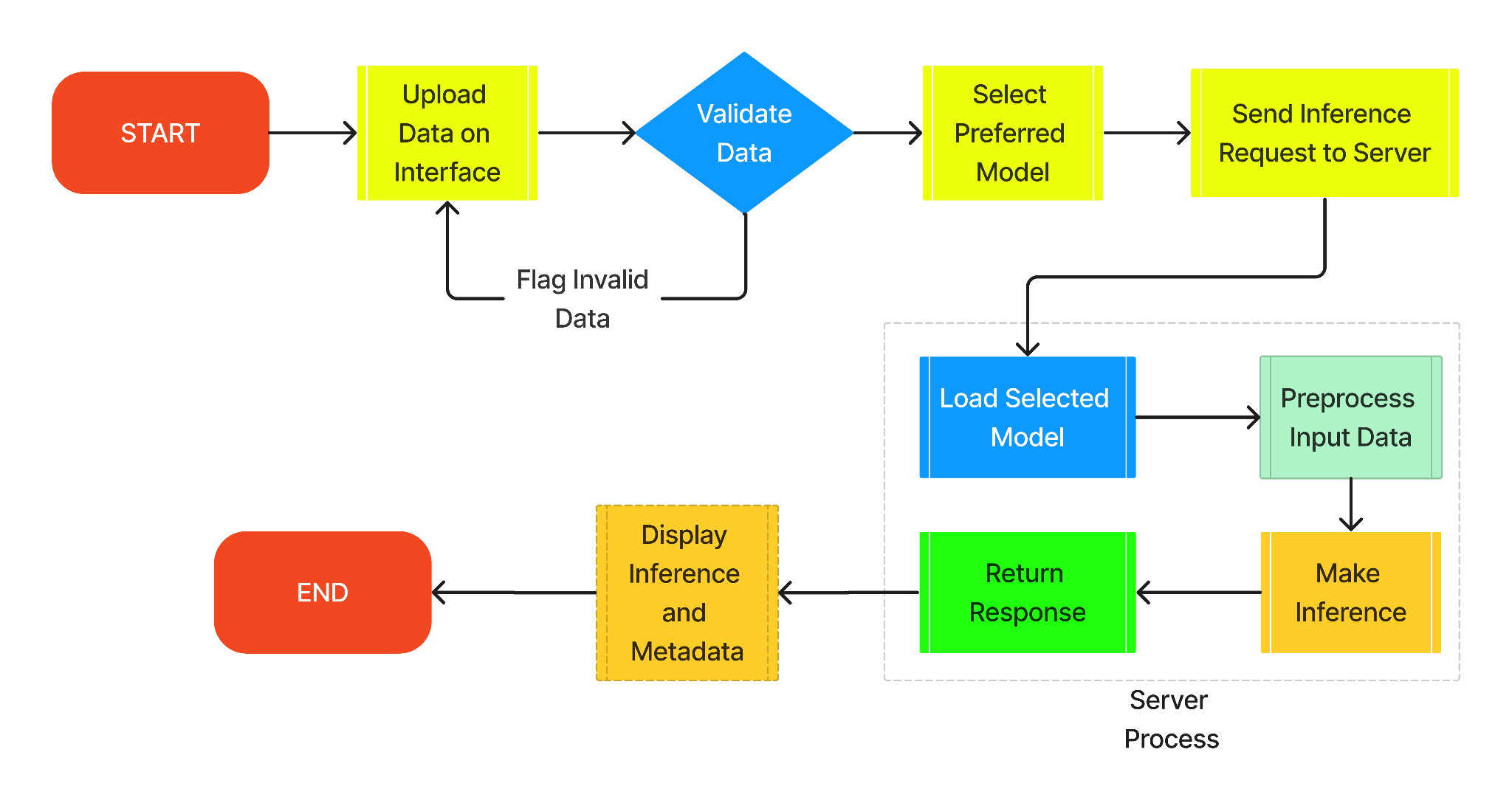
## 3.1a Development Process Flow

The diagram below illustrates the development workflow of the Credit Card Default Prediction system



## 3.1b Deployment Process Flow

The diagram below illustrates the deployment workflow of the solution when interacting via the frontend of the application, as well as the underlying API endpoint process.



## 3.2 Event Log

The solution developed has logging implemented at all levels from the data and model pipeline, backend server process and frontend interface to monitor activity at different logging levels. The loggers for the different layers log information to separate files.

## 3.3 Error Handling

Exception handlers have been implemented in sections of the code base which have dynamic behaviour so as to catch errors and perform specific action to handle the error and keep the process running.

For fatal errors in the development phase, the error messages are logged, and for errors in the deployment phase, an appropriate error message is relayed to the user interface for appropriate action to be taken.

# 4. Performance

The models developed have an average accuracy of ~71% which whilst not being the absolute best, is still better than a guess and can serve as an initial filter for obvious cases before human judgement can come in for the final decision.

In addition the model is able to make inference within milliseconds and the overall process between the interface and server also completes within milliseconds in local deployment.

## 4.1 Reusability

Sections of the preprocessing pipeline have been made reusable for both the training and inference generation phases

## 4.2 Application Compatibility

The application pipeline is built in python and can be run on any machine with python installed. In addition, the backend is separate from the frontend and can be called with any user interface redesign as long as the structure for the API call is maintained.

## 4.3 Resource Utilisation

The size of the underlying models range from ~200KB to ~30MB, indicating that the models are lightweight and can be deployed even on a very basic server for maintained quick inference times

## 4.4 Deployment

To demonstrate the functionality of the solution built, it has been deployed locally using FastAPI to build a prediction endpoint and Streamlit for the web user interface.

### 4.4.1 Local Deployment Details

Code Repository: <https://github.com/theabrahamaudu/credit_card_default_predictor>

Steps:

* Clone repository
* Create virtual environment
* Open terminal
* Install requirements: python -m pip install -r requirements.txt
* Start FastAPI server: python -m src/main
* Start Streamlit frontend: streamlit run src/streamlit\_frontend.py

Refer to README.md on the code base for more details.

### 4.4.2 Cloud Deployment

Server deployed on Render: <https://credit-card-predictor-api.onrender.com>



Frontend deployed on Streamlit: <https://creditcarddefaultapp.streamlit.app/>



# 5. Conclusion

The Credit Card Default Predictor app will take a credit card owner’s information and financial history and determine the probability that the credit card owner will default on their monthly card payment.