Low Level Design(LLD)

Credit card default prediction

Chetna Sharma

Document version control

Date issued	Version	Description	Author
23 March 2023	1	Initial LLD	Chetna Sharma

Contents

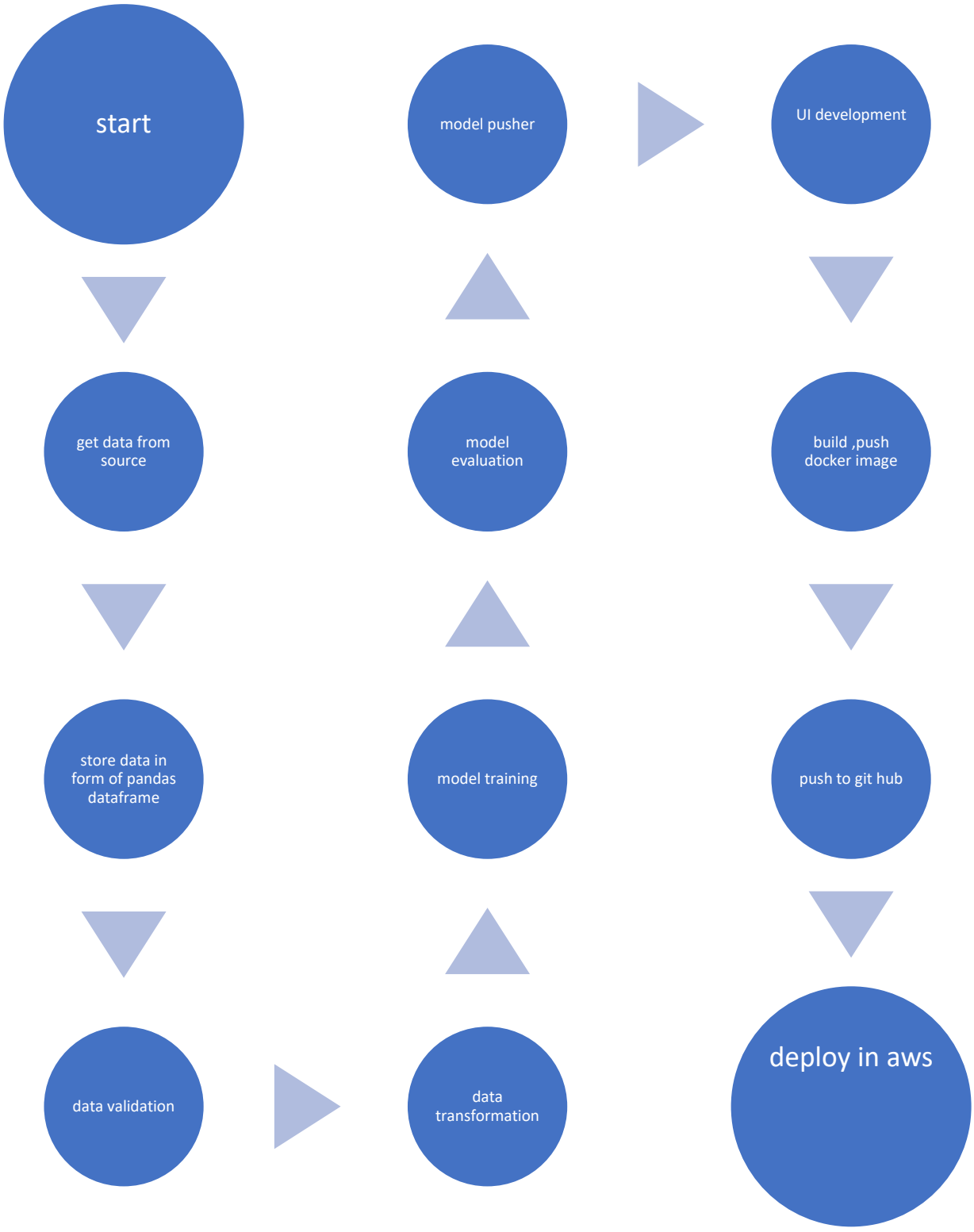
What is a Low-Level design document?

A Low-Level Design (LLD) document is a technical blueprint that provides detailed specifications for how a specific component or module of a software system should be implemented. It comes after the High-Level Design (HLD) document, which defines the system's architecture and major components at a higher level. The LLD document delves into the specifics of each component, outlining the algorithms, data structures, and interfaces required to build the system.

Scope

The low-Level Design document serves as a comprehensive guide for developers during the implementation phase. It helps ensure the software system's consistency, maintainability, and scalability by providing a detailed and structured approach to building each component. Additionally, it acts as a crucial communication tool among team members, making it easier to distribute the workload and coordinate efforts effectively.

Architecture

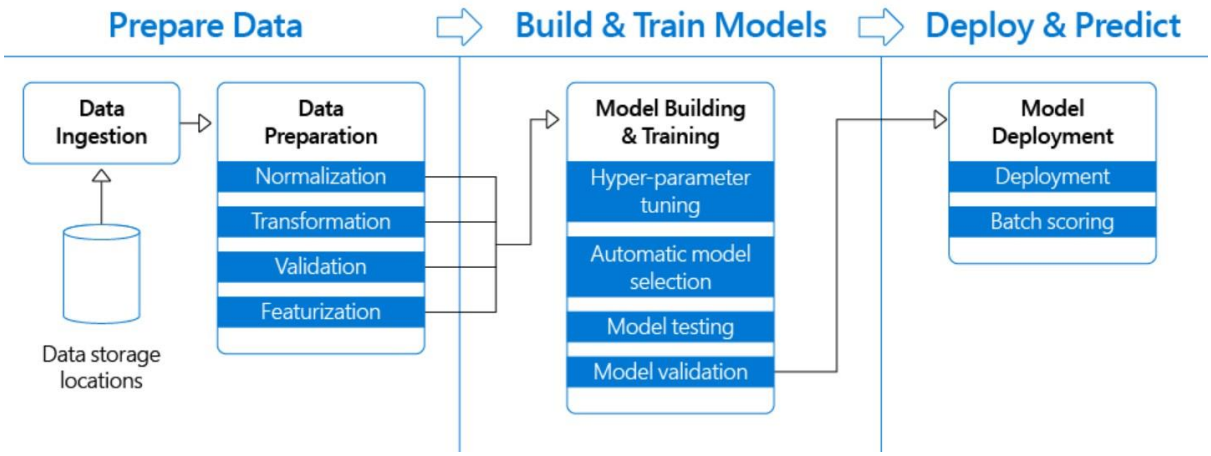


Architecture Description

Data Description

The information about the dataset is given in the form of a CSV file. the data consists of 25 feature columns and 30,000 rows respectively. Columns in the data set consist of information regarding the client's credit history such as credit amount, paid, due, limit bal, marital status, age, education, default status, etc.

Training Pipeline



Data insertion into a database

To use the data first we need to store it in a database .in this project we have used the MongoDB database to be utilized for further processing.

Export data from a database

The data in a stored database is exported as a CSV file to be used for Data Pre-processing and Model Training.

Data ingestion

Here we use the CSV file and read it in the form of a pandas data frame. then we check for null values and missing data in the dataset. after that, we perform a train test split for separating train and test data and store it in a feature store folder the split size is taken as 0.2 for performing a train test split.

Data validation

In this step, we identify if the train data and test data are similar or if some changes happened at the time of the train test split. The changes could be feature changes, data structure, or data size, or there may be any kind of data drift that is detected by checking data distribution with the help of the Scikit Learn library.

Data transformation

In the process of data transformation, data frame columns are modified into new columns, and feature scaling is done with the help of a robust scaler for data normalization. the resulting data frame is then stored in the form of a (transformer. pkl) file in the artifacts folder.

Model training

During Model training, first, the appropriate model algorithm is selected for its training, for selecting the most appropriate algorithm we made a function that applies different algorithms like (random forest, decision tree, AdaBoost, and gradientboostclassifiers) whichever model provides the highest accuracy and roc,auc score after applying on data frame is selected and then hyperparameter tuning is performed on it to make it more effective. the resulting model is stored in the form of a (model. pkl) file and if the model is retrained then it first checks if it performs better than the previous model and then stores it in the (model. pkl) file in the artifact folder.

Model Evaluation

At the time of model evaluation, the base model is compared to the current model with the help of a performance matrix to get the efficacy and reliability of the new model.

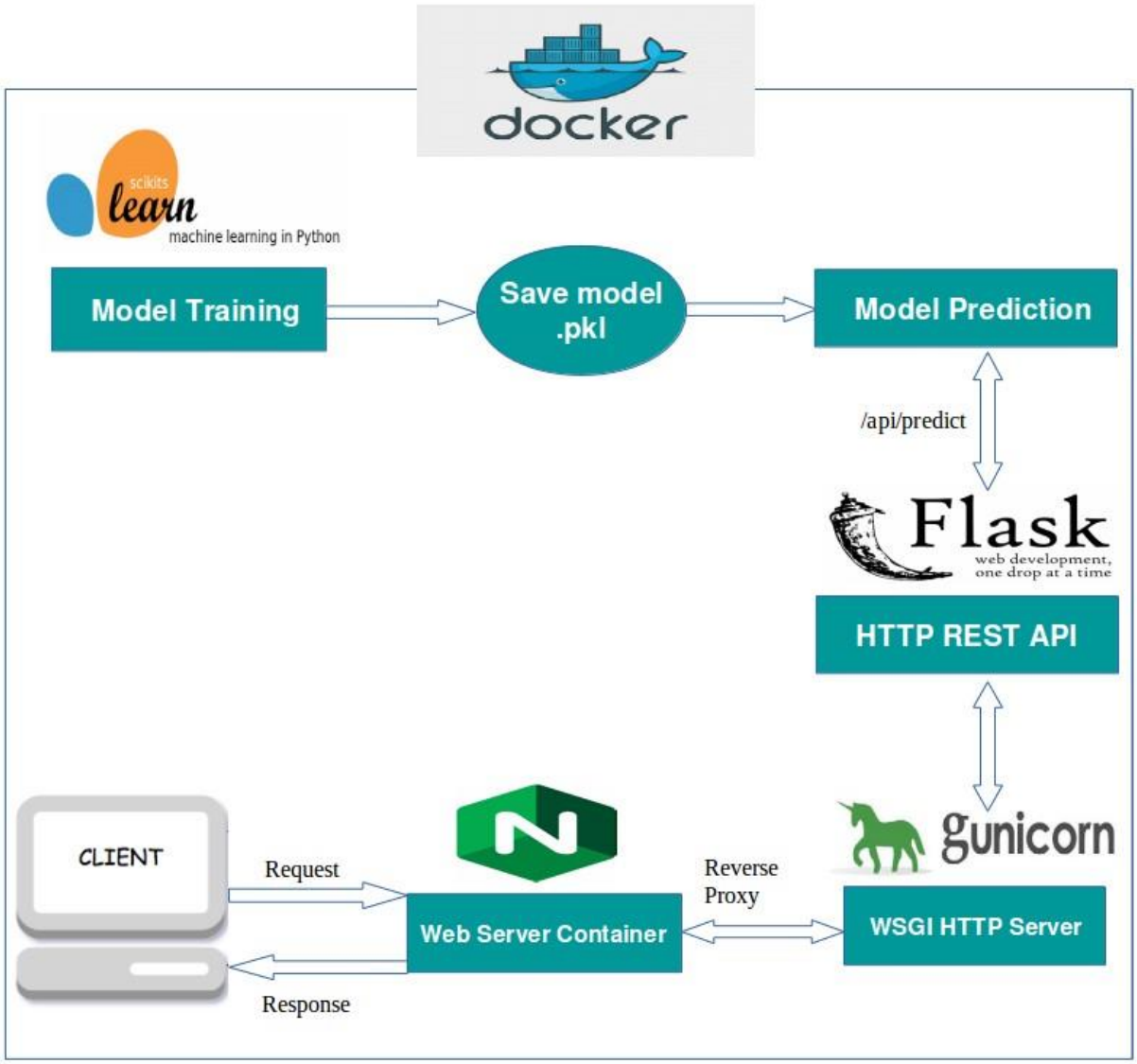
Model pusher

model pusher is used for saving artifact files and models used during training so, that they further can be used for monitoring and retraining.

Prediction pipeline

In the prediction pipeline, we load the required trained modal files from artifacts and saved models folder which are transformer .pkl and modal .pkl. we apply the transformations to new data provided and utilizing the files mentioned above we make the predictions based on the information provided.

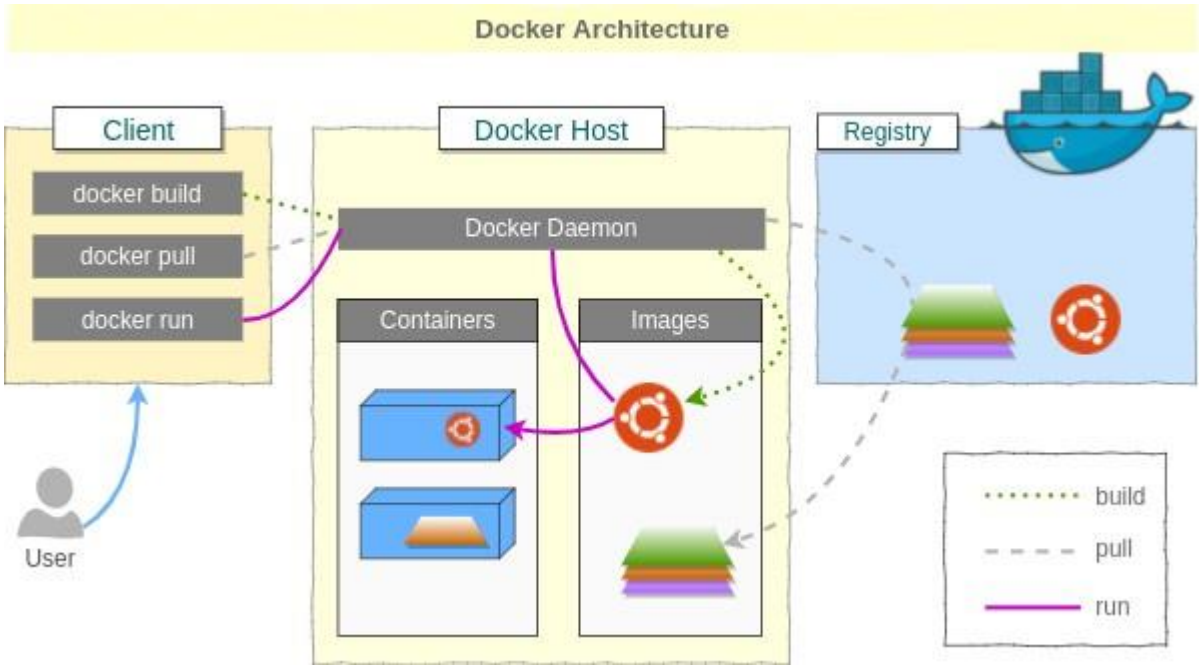
Make user interface



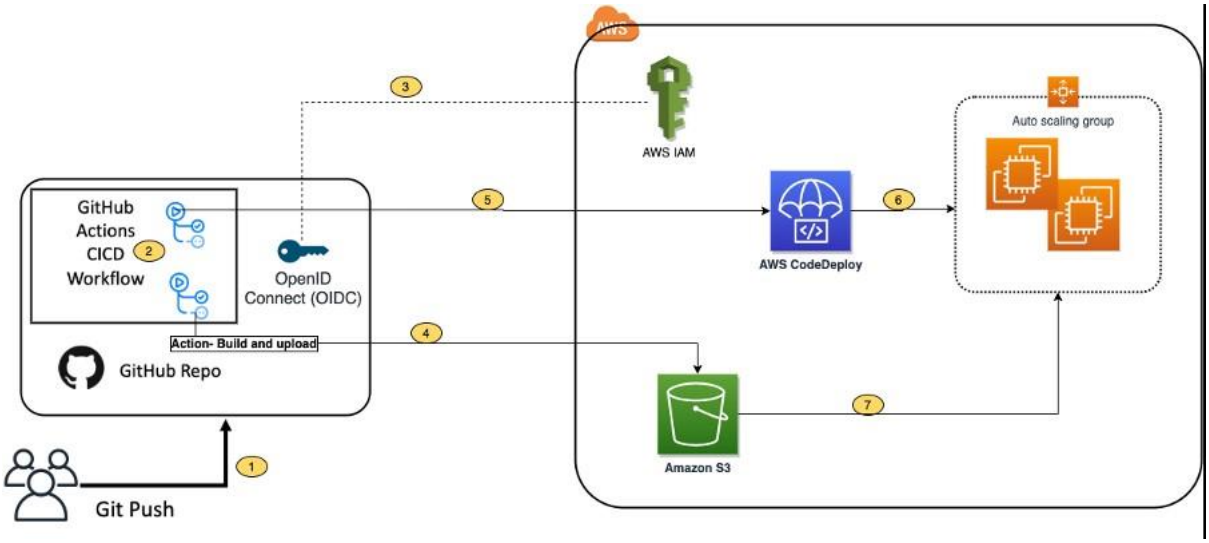
We make an HTML application that provides a user interface for the clients utilizing which they can get predictions if a person can make credit default or not. this application uses a prediction pipeline for making predictions.

Deployment

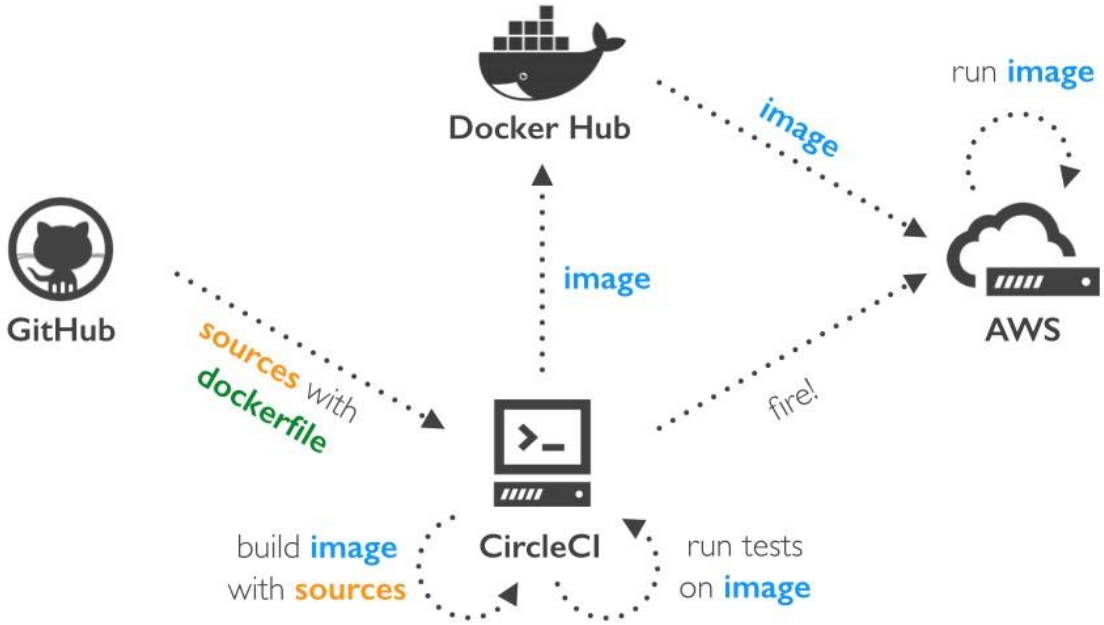
Docker architecture



AWS Flow



Continuous integration and deployment process



We did deployment using the ci/cd pipeline which is continuous delivery and continuous deployment. For deployment purposes, we have used Amazon web services (AWS) First, we build a docker image and push it into the docker container then push it into the (ECR) repository from there we store it in the (EC2)machine and artifacts, and saved models in the (s3 bucket) by using GitHub actions.

Why used CI/CD pipeline?

Using a CI/CD (Continuous Integration/Continuous Deployment) pipeline in the credit risk prediction system development offers numerous benefits that significantly enhance the efficiency, reliability, and scalability of the project. Here are the key reasons to use a CI/CD pipeline:

1. **Automated Testing and Validation:** CI/CD allows for automated testing of code changes and model updates. This includes unit tests, integration tests, and model evaluation against validation datasets. Automated testing ensures that code modifications do not introduce regressions or errors.

2. **Faster Iterations and Feedback:** With CI/CD, every code change is automatically built, tested, and deployed to a staging environment. This rapid feedback loop enables faster iterations, allowing developers to quickly iterate on improvements and fine-tune model performance.

3. **Improved Code Quality:** The automated testing and validation processes in CI/CD promote better code quality by catching issues early in the development cycle. This leads to cleaner, more maintainable, and reliable code.

4. **Reduced Human Errors:** CI/CD minimizes the chances of human errors during the deployment process. Automated pipelines ensure consistency and standardization across different environments.

5. **Easy Collaboration:** CI/CD fosters seamless collaboration among team members. Developers can work on different features or components in parallel without worrying about integration issues.

6. **Version Control Integration:** CI/CD integrates seamlessly with version control systems like Git, making it easy to trigger builds and deployments whenever changes are pushed to the repository.

7. **Continuous Integration (CI):** CI ensures that code changes are continuously integrated into the shared codebase. It helps identify conflicts and merge issues early, enabling continuous development and integration of features.

8. **Continuous Deployment (CD):** CD automates the deployment of the trained model to production or staging environments whenever a new version is available. This reduces the manual effort required for deployment and makes the process more reliable.

9. **Infrastructure as Code (IaC):** CI/CD pipelines can be designed using infrastructure-as-code principles, allowing the entire deployment process to be version-controlled, easily reproducible, and scalable.

10. **Quick Rollbacks:** In case of issues with a new model deployment, CI/CD pipelines make it easier to rollback to the previous stable version, minimizing downtime and service disruption.

11. **Scalability:** CI/CD pipelines are highly scalable, allowing the system to handle a large number of code changes, model updates, and deployments efficiently.

12. **Auditing and Compliance:** CI/CD pipelines provide a trail of all code changes, tests, and deployments, aiding in auditing and compliance efforts.

By incorporating CI/CD pipelines into the development process of the credit risk prediction system, we can continuously improve the model's performance, deploy updates efficiently, and maintain a robust and reliable system. This iterative and automated approach ensures that the credit risk prediction system remains accurate and responsive to changing business requirements and market conditions.