Low-Level Design

Stores Sales Prediction

|  |  |
| --- | --- |
| Written By | Avinash Durugkar |
| Version | 1.0 |
| Date | 17-05-2023 |

**Document Change Control Record**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Comments** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

### Reviews:

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Reviewer** | **Comments** |
|  |  |  |  |

### Approval Status:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Review**  **Date** | **Reviewed By** | **Approved By** | **Comments** |
|  |  |  |  |  |

**Index**

|  |  |
| --- | --- |
| **Content** | **Page No** |
| 1. Introduction | 4 |
| 1.1 What is Low-Level Design Document | 4 |
| 1.2 Scope | 4 |
| Architecture | 5 |
| 2. Architecture Description | 5 |
| 2.1 Data Description | 5 |
| 2.2 Data Gathering | 6 |
| 2.3 Raw Data Validation | 6 |
| 2.4 Data Transformation | 6 |
| 2.5 New Feature Generation | 7 |
| 2.6 Data Pre-processing | 7 |
| 2.7 Feature Engineering | 7 |
| 2.8 Parameter Tuning | 7 |
| 2.9 Model Building | 7 |
| 2.10 Model Saving | 7 |
| 2.11 Flask setup | 8 |
| 2.12 GitHub | 8 |
| 2.13 Deployment | 8 |
| 3. Unit Test Cases | 9 |

**1. Introduction**

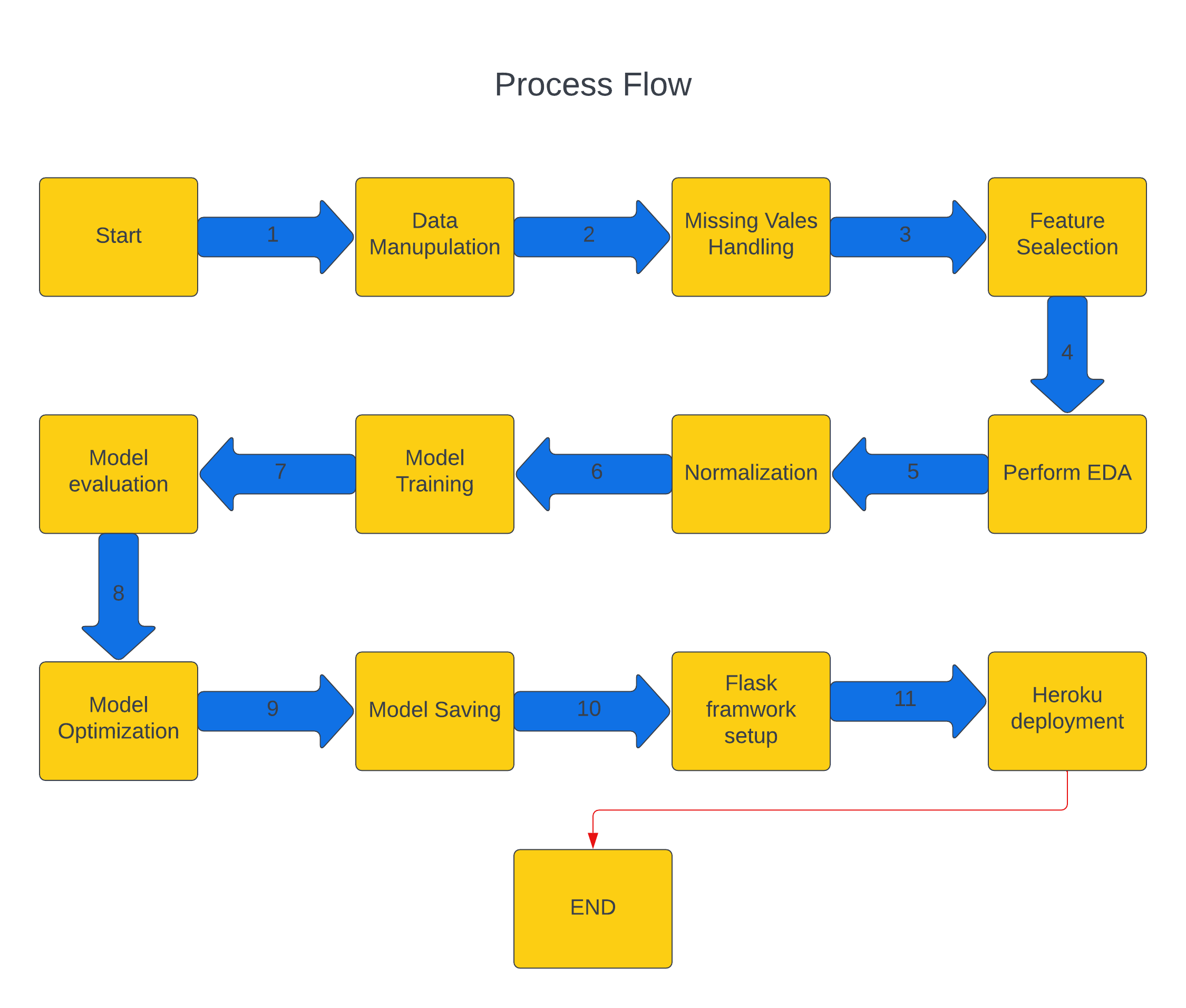
**1.1 What is Low-Level Design Document.**

The goal of LLD or a low-level design document (LLDD) is to give the internal logical design of the actual program code for **‘Credit card defaulter’**. LLD describes the class diagrams with the methods and relations between classes and program specs. It describes the modules so that the programmer can directly code the program from the document.

**1.2 Scope**

Low-level design (LLD) is a component-level design process that follows a step-by-step refinement process. This process can be used for designing data structures, required software architecture, source code, and ultimately, performance algorithms. Overall, the data organization may be defined during requirement analysis and then refined during data design work.

**Architecture**

****

**2. Architecture Description**

**2.1 Data Description**

Given is the variable name, variable type, the measurement unit, and a brief description. The concrete compressive strength is the regression problem. The order of this listing corresponds to the order of numerals along the rows of the database.

# change dataset

|  |  |  |
| --- | --- | --- |
| Name | Data Type | Measurement |
| ID | Integer | ID of each client |
| LIMIT\_BAL | float | Amount of given credit in NT dollars (includes individual and family/supplementary credit |
| SEX | integer | Gender (1=male, 2=female) |
| EDUCATION | integer | 1=graduate school, 2=university, 3=high school, 4=others, 5=unknown, 6=unknown |
| MARRIAGE | Integer | Marital status (1=married, 2=single, 3=others) |
| AGE | Integer | Age in years |
| PAY\_0 | Integer | Repayment status in September, 2005 (-1=pay duly, 1=payment delay for one month, 2=payment delay for two months, … 8=payment delay for eight months, 9=payment delay for nine months and above) |
| PAY\_2 | Integer | Repayment status in August, 2005 (scale same as above) |
| PAY\_3 | Integer | Repayment status in July, 2005 (scale same as above) |
| PAY\_4 | integer | Repayment status in June, 2005 (scale same as above) |
| PAY\_5 | integer | Repayment status in May, 2005 (scale same as above) |
| PAY\_6 | integer | Repayment status in April, 2005 (scale same as above) |
| BILL\_AMT1 | float | BILL\_AMT1: Amount of bill statement in September, 2005 (NT dollar) |
| - BILL\_AMT2 | float | - BILL\_AMT2: Amount of bill statement in August, 2005 (NT dollar) |
| - BILL\_AMT3 | float | - BILL\_AMT3: Amount of bill statement in July, 2005 (NT dollar) |
| - BILL\_AMT4 | float | - BILL\_AMT4: Amount of bill statement in June, 2005 (NT dollar) |
| - BILL\_AMT5 | float | - BILL\_AMT5: Amount of bill statement in May, 2005 (NT dollar) |
| - BILL\_AMT6 | float | - BILL\_AMT6: Amount of bill statement in April, 2005 (NT dollar) |
| PAY\_AMT1 | float | Amount of previous payment in September, 2005 (NT dollar) |
| PAY\_AMT2 | float | Amount of previous payment in August, 2005 (NT dollar) |
| PAY\_AMT3 | float | Amount of previous payment in July, 2005 (NT dollar) |
| PAY\_AMT4 | float | Amount of previous payment in June, 2005 (NT dollar) |
| PAY\_AMT5 | float | Amount of previous payment in May, 2005 (NT dollar) |
| PAY\_AMT6 | float | Amount of previous payment in April, 2005 (NT dollar) |
| default.payment.next.month: Default payment (1=yes, 0=no) | Integer | default.payment.next.month: Default payment (1=yes, 0=no) |

**2.2 Data Gathering**

Data source: <https://www.kaggle.com/datasets/uciml/default-of-credit-card-clients-dataset>

Train and Test data are stored in .csv format.

**2.3 Raw Data Validation**

Once the data has been loaded, several validations need to be performed before proceeding with any further operations, particularly in the case of credit card defaulters. These validations include checking for a zero-standard deviation across all columns and identifying any columns with complete missing values. These checks are crucial because attributes with these characteristics do not contribute to the prediction of credit card defaulters.

For instance, if an attribute has a zero-standard deviation, it means that all the values in that attribute are the same, resulting in a mean of zero. This indicates that regardless of whether the sales increase or decrease, that particular attribute will remain unchanged. On the other hand, missing data in the training dataset can significantly diminish the power and accuracy of a predictive model, potentially leading to a biased outcome. Without properly analysing the behaviour and relationships with other variables, incorrect predictions may result.

Hence, it is imperative to validate and address these issues to ensure the reliability and effectiveness of credit card defaulter prediction models.

**2.4 Data Transformation**

Before storing the data in the database, data transformation is necessary to ensure that it can be easily inserted into the database in a suitable format. In the case of credit card defaulters, there are missing values in the 'Item Weight' and 'Outlet Type' attributes. To address this issue, these missing values are filled with appropriate data types in both the training set and the test set. This step ensures that the data is complete and compatible for further analysis and modelling.

**2.5 Data Pre-processing**

During the data pre-processing phase for credit card defaulters, certain adjustments and mappings are made to improve the quality and relevance of the dataset. Specifically, for the 'EDUCATION' and 'MARRIAGE' attributes, values such as 5, 6, and 0 are encountered, which lack a clear description. To handle these cases, these values are grouped under the category of "Others" (4), as they do not fit into the existing defined categories.

**2.6 Feature Engineering**

After pre-processing it was found that some of the attributes are not important to the item sales for the particular outlet. So those attributes are removed. There are some columns that needs to be dropped as they don't seem to help in our analysis

**Model Selection:**

Accuracy: Accuracy measures the proportion of correctly classified instances out of the total number of instances. It provides a general overview of the model's performance but can be misleading when classes are imbalanced.

Precision: Precision calculates the proportion of true positive predictions (correctly identified positive instances) out of all positive predictions. It focuses on the accuracy of positive predictions and helps assess the model's ability to avoid false positives.

Recall (Sensitivity or True Positive Rate): Recall measures the proportion of true positive predictions out of all actual positive instances. It indicates the model's ability to identify positive instances and helps assess the occurrence of false negatives.

F1 Score: The F1 score combines precision and recall into a single metric that balances both measures. It provides a harmonic mean of precision and recall, giving equal weight to both metrics. F1 score is useful when there is an uneven class distribution.

Specificity (True Negative Rate): Specificity measures the proportion of true negative predictions (correctly identified negative instances) out of all actual negative instances. It helps evaluate the model's ability to identify negative instances and avoid false positives.

ROC Curve (Receiver Operating Characteristic Curve): The ROC curve visualizes the trade-off between true positive rate (sensitivity) and false positive rate (1 - specificity) at various classification thresholds. It provides an overall assessment of the model's performance across different thresholds.

**2.8 Parameter Tuning**

Parameters are tuned using GridSearchCV. The parameters are tuned on Random Forest Classifier.

**2.9 Model Building**

After applying various pre-processing techniques, including scaling and hyperparameter tuning, the pre-processed data is utilized by a Random Forest model. The evaluation results demonstrate the model's remarkable performance, achieving an impressive score of 0.84 when employing the RandomForestClassifier() function. This outcome underscores the effectiveness of the random forest methodology, as it surpasses alternative models such as XGBClassifier and AdaBoostClassifier, which achieved scores of 0.82 and 0.75 respectively. Therefore, the Random Forest model stands out as the superior choice for this specific problem.

**2.10 Model Saving**

Model is saved using dill library in `. pkl` format.

**2.11 Flask Setup for Data Extraction**

After saving the model in .pkl file format we then create an app.py flask web framework (Written in python) and then we render the home.html template and use request to extract all the form selection selected by the user and then we predict the sale price by using the selected records by the user.

**2.13 GitHub**

The whole project directory will be pushed into the GitHub repository.

GitHub Project link: <https://github.com/adurugkar/Credit_card_defaulters>

**2.14 Deployment**

The cloud environment was set up and the project was deployed from GitHub into the Heroku cloud platform.

WebApp link -

**3. Unit Test Cases.**

|  |  |  |
| --- | --- | --- |
| **Test Case Description** | **Pre-Requisite** | **Expected Result** |
| Verify whether the Application URL is  accessible to the user | 1. Application URL  should be defined | Application URL should be  accessible to the user |
| Verify whether the Application loads completely for the user when the URL is accessed | 1. Application URL is accessible 2. Application is deployed | The Application should load completely for the user when the URL is accessed |
| Verify whether a user is able to see input fields while opening the application | 1. Application is accessible 2. The user is able to see the input fields | Users should be able to see input fields on logging in |
| Verify whether a user is able to enter the input values. | 1. Application is accessible 2. The user is able to see the input fields | The user should be able to fill the input field |
| Verify whether a user gets predict button to submit the inputs | 1. Application is accessible 2. The user is able to see the input fields | Users should get Submit button to submit the inputs |
| Verify whether a user is presented with recommended results on clicking submit | 1. Application is   accessible   1. The user is able to see the input fields. 2. The user is able to see the submit button | Users should be presented with recommended results on clicking submit |
| Verify whether a result is in accordance with the input that the user has entered | 1. Application is accessible 2. The user is able to see the input fields. 3. The user is able to see the submit button | The result should be in accordance with the input that the user has entered |