Low Level Design (LLD)

Backorder Prediction

Written By	Anurag Bisen , Shashank Bhat
Document Version	0.3
Last Revised Date	27 – sep-2021

Document Control:

Change Record:

Version	Date	Author	Comments
0.1	23 / Sep/2021	Anurag Bisen	Introduction & Architecture defined
0.2	24 / Sep -2021	Shashank Bhat	Architecture & Architecture Description appended and updated
0.3	26 /Sep/2021	Anurag Bisen	Unit Test Cases defined and appended
0.4	30/oct/2021	Anurag Bisen	Technology stack

Reviews:

Version	Date	Reviewer	Comments
1.0	29 /Sep/2021	Shashank Bhat	Document Content , Version Control and Unit Test Cases to be added

Approval Status:

Version	Review Date	Reviewed By	Approved By	Comments
1.1	10/Oct/2021	Shashank Bhat		Final

Contents

1. Introduction:

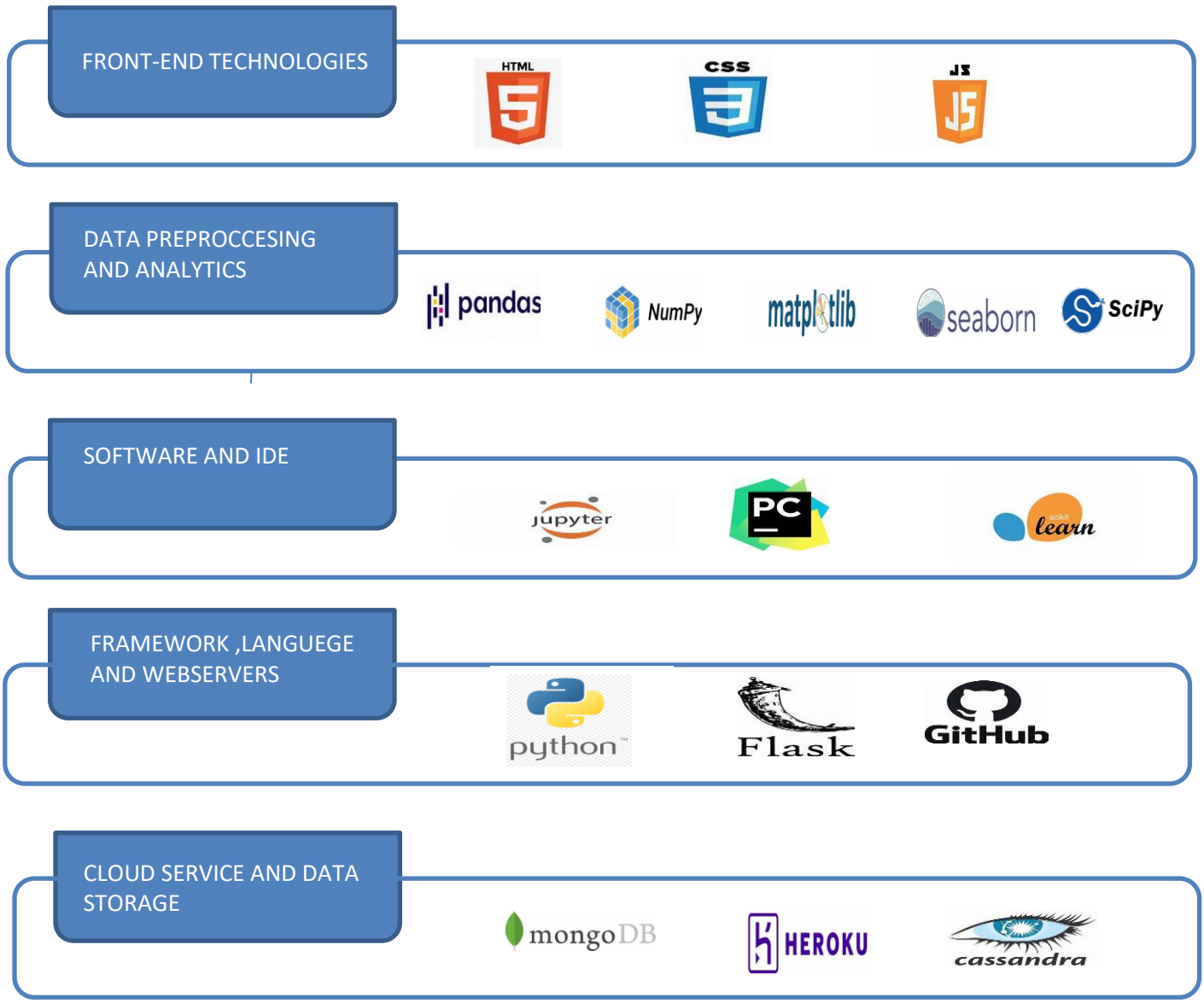
1.1. What is Low-Level design document?

The goal of LLD or a low-level design document (LLDD) is to give the internal logical design of the actual program code for Food Recommendation System. 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.

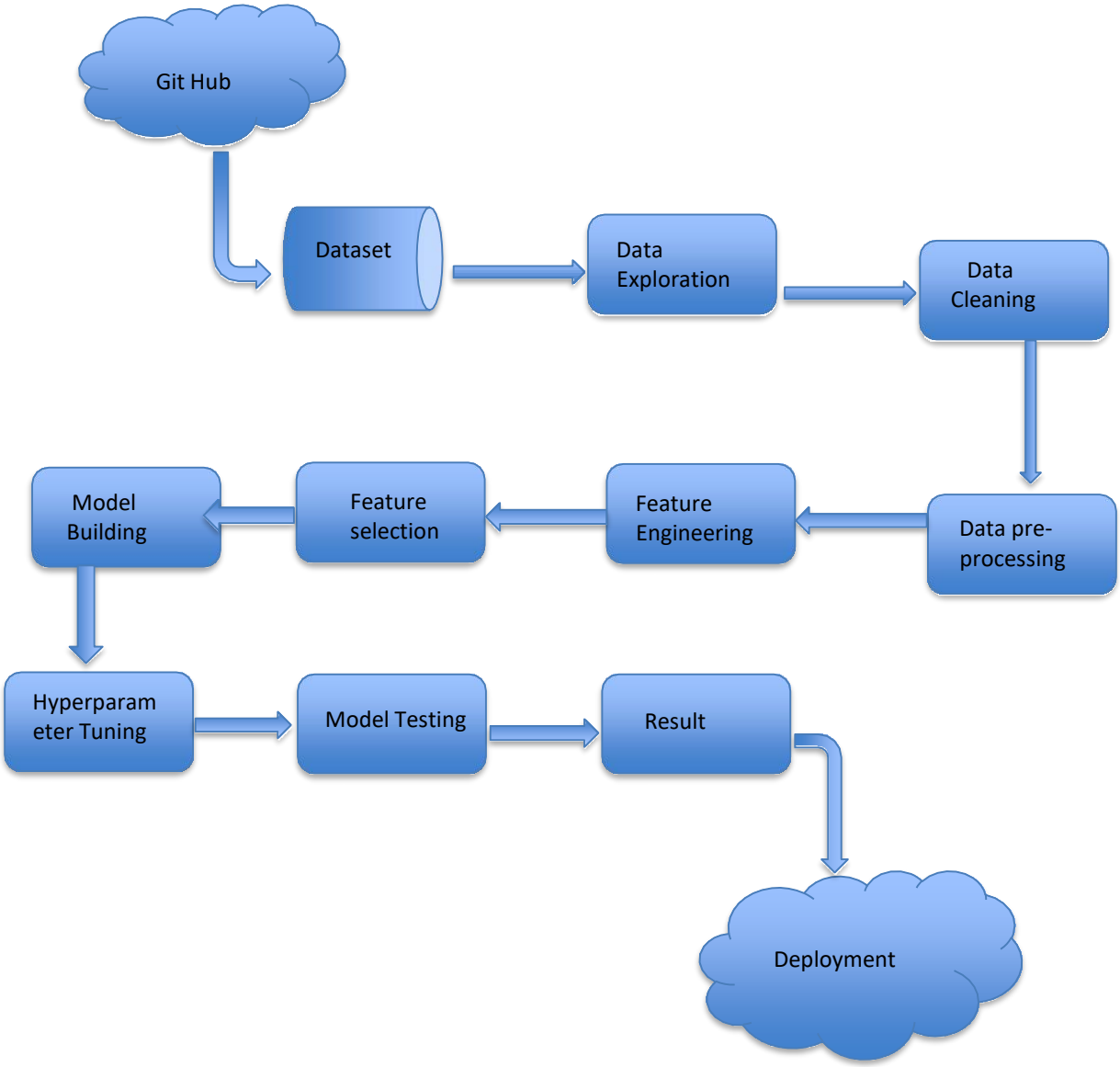
1.2. 1.Scope:

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

1.1.2 Technology Stack :



2.
Architecture



3. Architecture Description:

3.1. Data Description:

The dataset of this research has been published in Kaggle (Currently the page is not available). It is divided into the training and testing datasets.
Data Source: https://github.com/rodrigosantis1/backorder_prediction/blob/master/dataset.rar
The dataset is highly imbalanced which should be addressed for accurate predictions by the model each dataset contains 23 attributes with 1,687,862 and 242,077 observations for the training and testing sets, respectively.

3.2. Data Transformation:

In the Transformation Process, we will convert our original dataset which is in CSV Format to pandas data frame after reading dataset we have two pandas data frame train and test concatenated into one data frame.

3.3. Exploratory Data Analysis:

Exploratory Data Analysis, or EDA, is an important step in any Data Analysis or Data Science project EDt. A is the process of investigating the dataset to discover patterns, and anomalies (outliers), and form hypotheses based on our understanding of the dataset.

3.4 Data Insertion into Database:

a. Database Creation and connection - Create a database with name passed. If the database is already created, open the connection to the database.
b. Table creation in the database.
c. Insertion of files in the table.

3.5. Export Data from Database:

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

3.6. Data Cleaning:

Data cleansing or data cleaning is the process of detecting and correcting (or removing) corrupt or inaccurate records from a record set, table, or database and refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data.

3.7 Data Pre-processing:

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model. When creating a machine learning project, it is not always a case that we come across the clean and formatted data.

3.8 Feature Engineering:

Feature engineering is the process of using domain knowledge of the data to create features that make machine learning algorithms work. If feature engineering is done correctly, it increases the predictive power of machine learning algorithms by creating features from raw data that help facilitate the machine learning process.

3.9 Feature Selection:

A feature selection algorithm can be seen as the combination of a search technique for proposing new feature subsets, along with an evaluation measure which scores the different feature subsets. The simplest algorithm is to test each possible subset of features finding the one which minimizes the error rate. This is an exhaustive search of the space, and is computationally intractable for all but the smallest of feature sets.

3.10 Model Building:

A machine learning model is built by learning and generalizing from training data, then applying that acquired knowledge to new data it has never seen before to make predictions and fulfill its purpose. Lack of data will prevent you from building the model, and access to data isn't enough.

3.11 Random Forest:

A random forest classifier, a random forest is a Meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

3.12 DecisionTreeClassifier:

Decision tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.

3.13 XGBoost Model:

XGBoost Model for Classification XGBoost is short for Extreme Gradient Boosting and is an efficient implementation of the stochastic gradient boosting machine learning algorithm. The stochastic gradient boosting algorithm, also called gradient boosting machines or tree boosting, is a powerful machine learning technique that performs well or even best on a wide range of challenging machine learning problems.

3.14 Light Gradient Boost Model:

Light Gradient Boosted Machine, or LightGBM for short, is an open-source library that provides an efficient and effective implementation of the gradient boosting algorithm. LightGBM extends the gradient boosting algorithm by adding a type of automatic feature selection as well as focusing on boosting examples with larger gradients. This can result in a dramatic speedup of training and improved predictive performance.

3.15. Hyperparameter Tuning:

While model parameters are learned during training — such as the slope and intercept in a linear regression — hyperparameters must be set by the data scientist before training. In the case of a random forest, hyperparameters include the number of decision trees in the forest and the number of features considered by each tree when splitting a node.

3.16. Model Validation:

For machine learning systems, we should be running model evaluation and model tests in parallel. Model evaluation covers metrics and plots which summarize performance on a validation or test dataset. Model testing involves explicit checks for behaviors that we expect our model to follow.

3.17. Deployment:

We will be deploying the model in Heroku cloud platform.

3.17.01. Heroku:

Heroku is a cloud platform as a service supporting several programming languages. One of the first cloud platforms, Heroku has been in development since June 2007, when it supported only the Ruby programming language, but now supports Java, Node.js, Scala, Clojure, Python, PHP, and Go.

3.17.02. Flask:

Flask is a micro web framework written in python. Here we use flask to create an API that allows to send data, and receive a prediction as a response. Flask supports extensions that can add application features as if they were implemented in Flask itself.

4. Database:

Cassandra DB is used to retrieve, insert, delete and update the customer input database. Flask is a micro web framework written in python. Here we use the flask to create an API that allows sending data, and receive a prediction as a response. Flask API's act as an interface between the user, ML- Model and DB. Users interact with the deployed model on HEROKU and the result will be returned to the user by API's. The Flask API will collect all the customer input data store in Cassandra DB.

4. 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 the User is able to sign Up in the application.	1. Application is accessible	The User should be able to sign up In the application.
Verify whether user is able to Successfully login to the application.	1. Application is accessible 2. User is signed up to the application	User should be able to successfully Login to the application.
Verify whether user is able to see input fields on logging in.	1. Application is accessible 2. User is signed up to the application 3. User is logged in to the application	User should be able to see input fields on logging in.
Verify whether user is able to edit all input fields.	1. Application is accessible 2. User is signed up to the application 3. User is logged in to the application	User should be able to edit all input Fields.
Verify whether user gets Submit button to submit the inputs.	1. Application is accessible 2. User is signed up to the application 3. User is logged in to the application	User should get Submit button to submit the inputs.
Verify whether user is presented with predicted results on clicking Submit.	1. Application is accessible 2. User is signed up to the application 3. User is logged in to the application	User should be presented with predicted results on clicking Submit.
Verify whether the predicted results are in accordance to the Inputs user made.	1. Application is accessible 2. User is signed up to the application 3. User is logged in to the application	The predicted results should be in accordance to the selections user made