Flight Fare Price Prediction

LLD

Date: 25/10/2023

Team members:

- Yash Keshari
- Md Ehsanul Haque Kanan

Document Version Control

Date Issued	Version	Description	Author
15/10/2023	1	Initial LLD - V 0.1	Yash Keshari
24/10/2023	2	Major changes – V 0.2	Yash Keshari

Contents

1. Introduction

1.1. What is Low-Level design document?

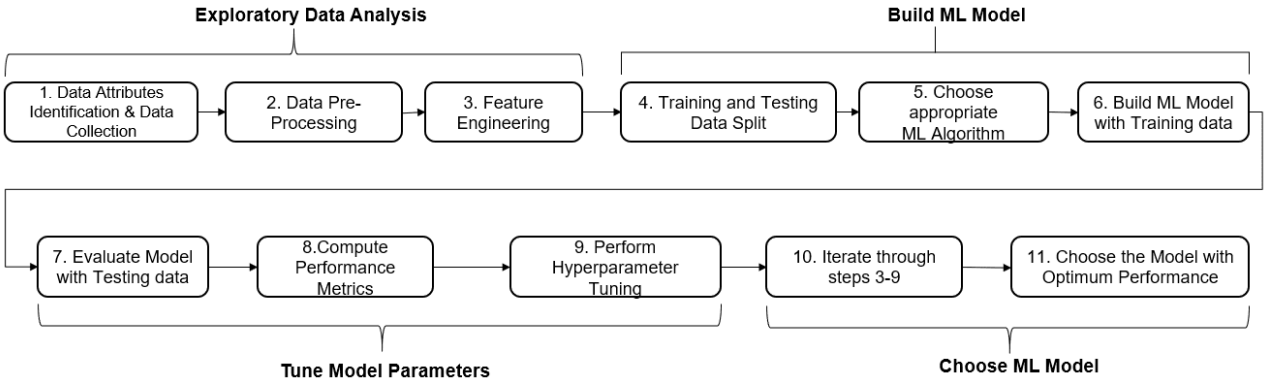
The goal of LLD or a low-level design document (LLD) is to give the internal logical design of the actual program code for Food Recommendation System. LLD describes the class diagrams with the methods and relations between classes and program specs. It describes the modules so that the programmer can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

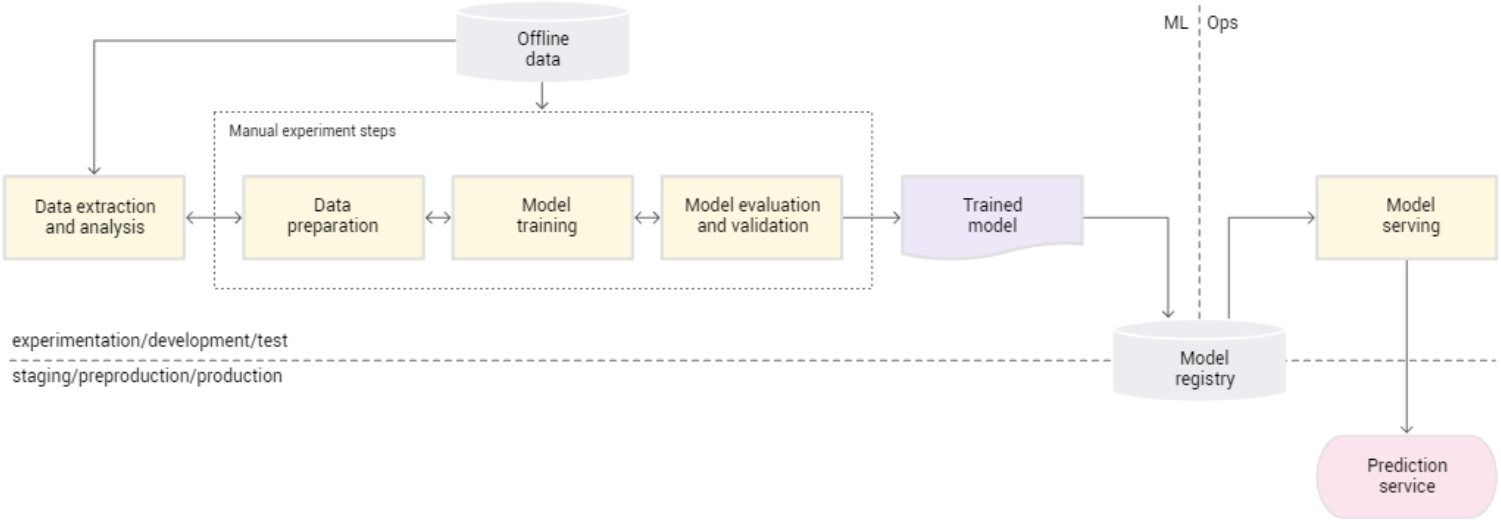
1.1. Scope

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.

2. Architecture

The architecture of the Flight Fare Price Prediction project is designed to deliver accurate and real-time fare estimates to travelers. The system collects data from various sources, processes it for modeling, and employs a Random Forest Regressor to predict fare prices. Users interact with the system through a user-friendly front-end, and the entire solution is deployed on a scalable cloud platform which is AWS for accessibility and responsiveness. This architecture ensures travelers receive up-to-date and precise fare predictions, enhancing their booking experience.





3. Architecture Description

- ## Data Description:

The project leverages a comprehensive dataset that encompasses various flight-related attributes, including airline details, routes, historical pricing data, and additional factors influencing fare prices.

- ## Web Scraping:

Real-time data, such as the latest fare prices, is gathered through web scraping from airline websites and other pertinent sources. This dynamic data collection ensures that users receive the most current fare estimates.

- ## Data Ingestion into the Artifact:

Processed data is ingested into the project's artifact repository for efficient storage and retrieval. This repository is a crucial resource for model training and user interactions.

- ## Export Data from Artifact:

Historical data can be exported from the artifact repository for model training and validation, ensuring the model is updated with the latest information.

- ## Data Pre-processing:

Data pre-processing encompasses tasks like scaling features, handling outliers, and encoding categorical variables to improve the quality of data used for model training.

- ## Data Transformation:

The collected data undergoes transformation to prepare it for modeling. This stage includes tasks like handling missing values, converting data types, and creating engineered features to enhance predictive accuracy. Feature engineering is performed to create and select relevant features that improve the model's ability to predict fare prices accurately.

- ## Model Building:

The central component of the architecture is the machine learning model development. A machine learning algorithm, such as the Random Forest Regressor, is employed to construct a predictive model that establishes relationships between flight features and fare prices.

- ## Model Evaluation:

The trained model is evaluated for its performance using various metrics, ensuring it meets the required accuracy standards. R2-score is employed in our case.

- ## Model Export:

Once validated, the model is exported for deployment, making it accessible to users for real-time fare predictions. We have used Joblib for this task.

- ## Front-end Development:

A user-friendly front-end application, such as a web or mobile app, is developed to facilitate user interactions. This interface enables users to input their flight details and receive fare predictions.

- Deployment (on AWS):

The entire system, including the predictive model and user interface, is deployed on a cloud platform, such as Amazon Web Services (AWS), for scalability and accessibility. This deployment allows users to access the service from various devices and locations.

4. Unit Test Cases

Test Case	Description and Steps
Data Pre-processing	1. Verify that missing values are handled correctly. 2. Check if categorical variable encoding is accurate. 3. Ensure feature scaling is applied as expected.
Feature Engineering	1. Test the creation of engineered features. 2. Verify that feature selection functions work accurately
Model Building	1. Evaluate the training of the Random Forest Regressor model with sample data. 2. Confirm that the model converges to a stable state. 3. Check that hyperparameters of the model are set correctly.
Data Export	1. Ensure the export of the trained model functions correctly and saves the model as intended.

User Interface	1. Verify that the user interface accepts user inputs for flight details. 2. Test the user input validation and ensure it handles invalid inputs appropriately.
Deployment (on AWS)	1. Test the deployment of the application on AWS and confirm it is accessible. 2. Ensure the system scales appropriately to handle user requests.
End-to-End Testing	1. Perform end-to-end testing where you input sample user data, run the model, and receive fare predictions to ensure the entire workflow functions correctly.
Error Handling	1. Verify that the system properly handles errors and exceptions, providing informative error messages to users or logs.