Low Level Design

FLIGHT FARE PREDICTION SYSTEM

Written By	GOBIKRISHNAN,SRIRAM,SHINY
Document Version	0.3
Last **Revised** Date	22– May -2024

Document Control

Change Record:

Version	Date	Author	Comments
0.1	17 – May - 2024	GOBI KRISHNAN	Introduction & Architecture defined
0.2	18 – May - 2024	SRIRAM	Architecture & Architecture Description appended and updated
0.3	19 – May - 2024	SHINY	Unit Test Cases defined and appended

Reviews:

Version	Date	Reviewer	Comments
0.2	22 – May - 2024		Document Content , Version Control and Unit Test Cases to be added

Approval Status:

Version	Review Date	Reviewed By	Approved By	Comments

Contents

1. Introduction 1
1.1. What is Low-Level design document? 1
1.2. Scope 1
2. Architecture 2
3. Architecture Description 3
3.1. Data Description 3
3.2. Data Preprocessing 3
3.3. Model Training 3
3.4. Model Evaluation 4
3.5. Model Serialization 4
3.6. Model Deployment 4
3.7. Prediction Service 5
3.8 Monitoring and Logging 6
4 Unit Test Cases 7

1. Introduction

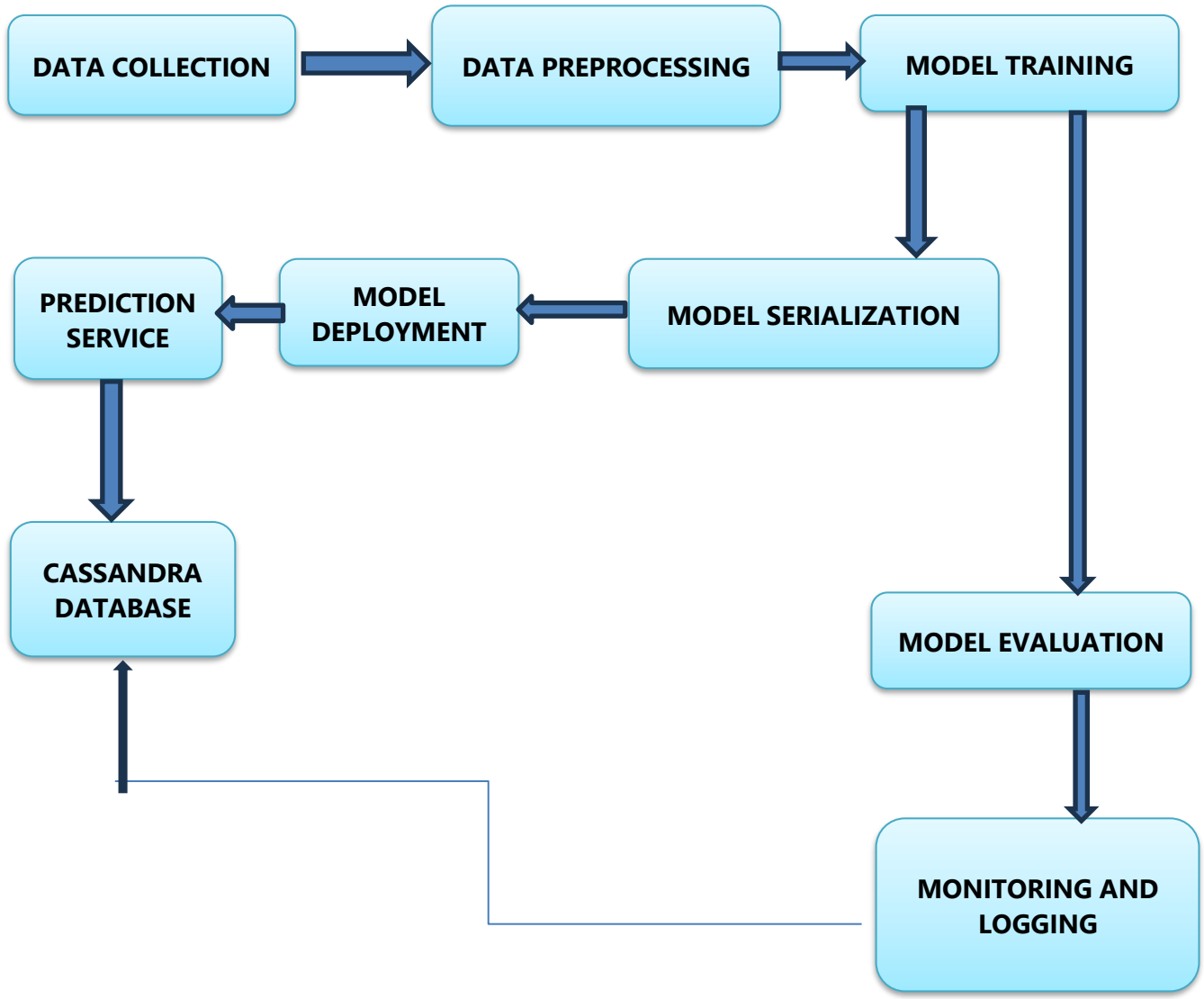
1.1. What is Low-Level design document?

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. 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



3. Architecture Description

3.1. Data Collection

Data collection is the initial and crucial step in the flight fare prediction project. In this project, we have collected data from Kaggle, a popular platform for datasets and machine learning competitions. The dataset from Kaggle includes various features relevant to predicting flight fares, such as flight details, airline information, travel dates, and more.

3.2. Data Preprocessing

Data preprocessing for the flight fare prediction project involved several critical steps to ensure the dataset was clean, consistent, and suitable for model training. Initially, all rows with missing values were removed to maintain data integrity. The Date_of_Journey, Dep_Time, and Arrival_Time columns were converted into datetime formats, from which day, month, hour, and minute features were extracted, and the original columns were subsequently dropped. The Duration column was standardized to include both hours and minutes, then split into separate Duration_hours and Duration_mins columns. Categorical data, such as Airline, Source, and Destination, were encoded using one-hot encoding, with less frequent categories grouped under 'Other' to reduce complexity. The Total_Stops column, representing an ordinal variable, was encoded numerically. Unnecessary columns like Route and Additional_Info were dropped to streamline the dataset. Finally, the processed features were combined into a single dataframe, and an ExtraTreesRegressor was used to determine feature importances, ensuring that the most relevant features were highlighted for model training. This thorough preprocessing pipeline prepared the dataset for accurate and efficient flight fare predictions using machine learning techniques.

3.3. Model Training

For the flight fare prediction project, model training was conducted using two powerful machine learning algorithms: the Random Forest Regressor and the XGBoost Regressor. Initially, the Random Forest Regressor was employed by importing the necessary module from sklearn.ensemble, followed by initializing and training the model on the preprocessed training dataset (X_train for features and y_train for target fares). The Random Forest algorithm, being an ensemble learning method, builds multiple decision trees and aggregates their results to enhance prediction accuracy and stability. In parallel, the XGBoost Regressor, a highly efficient implementation of gradient boosting, was utilized. By importing the XGBRegressor from the xgboost library, the model was similarly initialized and trained using the same training dataset. XGBoost is renowned for its performance and speed, combining the outputs of numerous weak learners to improve predictive accuracy. Both models were subsequently evaluated on the testing dataset to determine their effectiveness, providing a robust approach to accurately predict flight fares by leveraging the strengths of ensemble methods and gradient boosting. This dual-model training strategy ensures reliable and precise fare predictions for the final deployed system.

3.4. Model Evaluation

Model evaluation is a crucial step in the flight fare prediction project to ensure that the trained models perform well on unseen data and generalize effectively. After training the Random Forest Regressor and XGBoost Regressor, their performance was assessed using the test dataset. The evaluation involved calculating key metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R^2). These metrics provide insights into the models' accuracy and reliability. MAE measures the average absolute errors between the predicted and actual values, while MSE and RMSE evaluate the average squared differences and their square root, respectively. R^2 indicates the proportion of variance in the target variable that is predictable from the features. By comparing these metrics, we determine which model performs better on the test data, with lower MAE, MSE, and RMSE values indicating better performance, and higher R^2 values signifying a stronger model fit. This evaluation ensures that the final deployed system, whether using the Random Forest or XGBoost model, is both accurate and reliable in predicting flight fares.

3.5. Model Serialization

Model serialization, a critical step in deploying machine learning models, involves saving trained models to disk for later use without the need for retraining. In the flight fare prediction project, serialization was achieved using Python's joblib or pickle module. For both the Random Forest Regressor and XGBoost Regressor, serialization followed a similar process. With joblib, the trained models were serialized using the dump function, saving them as binary files with the .pkl extension. Alternatively, pickle allowed for serialization by opening a file in binary writing mode and using the dump function to save the models. These serialized models can then be stored on disk and later loaded into production environments or applications for making predictions on new data. This serialization process ensures that the trained models retain their learned parameters and can be efficiently deployed without the overhead of retraining, enabling real-time prediction capabilities for the flight fare prediction system.

3.6. Model Deployment

Model deployment is the process of integrating trained machine learning models into production systems to make predictions on new data. For the flight fare prediction project, deploying the trained Random Forest Regressor and XGBoost Regressor models involves several steps to ensure seamless integration and efficient performance. Firstly, the serialized models, saved as binary files using joblib or pickle, are loaded into the deployment environment. This typically involves setting up a server or cloud-based infrastructure capable of hosting the models and handling incoming prediction requests. Next, an application programming interface (API) is developed to expose the prediction functionality of the models. This API receives input data, preprocesses it if necessary, and then passes it to the loaded models for prediction. The predicted flight fares are returned as output to the calling application or user interface. Additionally, the deployment setup includes monitoring mechanisms to track model performance, handle scalability, and ensure reliability in real-time

prediction scenarios. Regular updates and maintenance are also essential to keep the deployed models accurate and up-to-date with changing data patterns. By following these deployment practices, the flight fare prediction models can seamlessly integrate into production environments, providing accurate predictions to users or applications in real time.

3.7. Prediction Service

For the flight fare prediction project, setting up a prediction service involves creating a robust system capable of receiving input data, processing it, and providing accurate fare predictions in real-time. This service typically utilizes the deployed machine learning models, such as the Random Forest Regressor and XGBoost Regressor, integrated into an application programming interface (API) or web service. The prediction service exposes endpoints that accept input parameters related to flight details, such as departure and arrival locations, travel dates, and other relevant information. Upon receiving a prediction request, the service preprocesses the input data to ensure it aligns with the format expected by the models. This may Involve feature engineering, data normalization, or encoding categorical variables, similar to the preprocessing steps performed during model training. Once the input data is prepared, it is passed to the loaded models for prediction. The models generate fare predictions based on the provided input, which are then returned as responses to the user or application requesting the prediction. The prediction service is designed to handle multiple concurrent requests efficiently, ensuring high availability and low latency. Additionally, error handling mechanisms are implemented to address any issues that may arise during prediction, such as invalid input data or model failures. Overall, the prediction service provides a seamless and reliable interface for users or applications to obtain accurate flight fare predictions on-demand, facilitating informed decision-making and enhancing user experience.

3.7 Cassandra Database

In the flight fare prediction project, Cassandra database plays a crucial role in storing and managing the training data used for model development and testing. Cassandra is a distributed NoSQL database known for its scalability and high availability, making it suitable for handling large volumes of data with high throughput requirements. To interact with Cassandra from Python, the cassandra-driver library is utilized.The provided code demonstrates the setup and utilization of Cassandra within the project environment. Initially, the necessary libraries are imported, including pandas for data manipulation and Cluster from cassandra.cluster for connecting to the Cassandra database. Additionally, authentication details and environment variables are configured to establish a secure connection to the Cassandra cluster.Once the connection is established, the script creates a keyspace named 'flightprice' and defines a table named 'trainingdata' within that keyspace. This table is structured to accommodate various attributes related to flight data, such as the date of journey, airline, destination, duration, price, and other relevant information. Sample data is then inserted into the 'trainingdata' table for testing purposes.Following data insertion, a query is executed to retrieve data from the 'trainingdata' table, and the resulting dataset is converted into a pandas DataFrame for further analysis or preprocessing. This allows for seamless integration between Cassandra and Python, enabling data scientists to leverage

Cassandra's distributed architecture while working with familiar data manipulation tools.Overall, Cassandra serves as a reliable and scalable storage solution for storing training data, facilitating efficient data retrieval and analysis for machine learning model development within the flight fare prediction project.

3.8 Monitoring and Logging

In the flight fare prediction project, monitoring and logging are essential components to ensure the reliability, performance, and security of the deployed system. Monitoring involves the continuous tracking of various metrics and system components to detect anomalies, performance issues, or potential failures in real-time. This includes monitoring the health and availability of servers hosting the prediction service, as well as the responsiveness of the prediction API endpoints. Additionally, monitoring may involve tracking model performance metrics, such as prediction accuracy and latency, to identify any degradation in model performance over time. Logging, on the other hand, involves recording relevant events, errors, or actions occurring within the system. This includes logging incoming prediction requests, along with metadata such as timestamps and user identifiers, to facilitate troubleshooting and auditing. Furthermore, logging can capture errors, exceptions, or warnings encountered during model prediction or data processing, providing valuable insights into system behavior and potential areas for improvement. By implementing robust monitoring and logging mechanisms, the flight fare prediction project can ensure operational efficiency, proactive issue detection, and effective troubleshooting, ultimately enhancing the reliability and performance of the deployed system.

4.Unit Test Cases

Test Case Description	Pre-Requisite	Expected Result
Verify if the prediction service URL is accessible to users	1. Prediction service URL should be defined	Prediction service URL should be accessible to users
Verify if the prediction service loads completely upon accessing the URL	1. Prediction service URL is accessible	Prediction service should load completely upon accessing the URL
Verify if users can submit valid data through the prediction service	1. Prediction service is accessible	Users should be able to submit valid data through the prediction service
Verify if users receive accurate fare predictions upon submitting valid input data	1. Prediction service is accessible	Users should receive accurate fare predictions upon submitting valid input data
Verify if users receive an error message for invalid input data	1. Prediction service is accessible	Users should receive an error message for invalid input data
Verify if the prediction service returns results according to user inputs	1. Prediction service is accessible	The prediction service should return results according to user inputs

Test Case Description	Pre-Requisite	Expected Result
Verify if users have options to filter the recommended results	1. Prediction service is accessible	Users should have options to filter the recommended results
Verify if key performance indicators (KPIs) update based on user inputs	1. Prediction service is accessible	Key performance indicators (KPIs) should update based on user inputs
Verify if the KPIs display details of the predicted flight fares	1. Prediction service is accessible	The KPIs should display details of the predicted flight fares
Verify if the prediction service handles concurrent requests gracefully	1. Prediction service is accessible	The prediction service should handle concurrent requests gracefully
Verify if the prediction service logs incoming requests and responses	1. Prediction service is accessible	The prediction service should log incoming requests and responses
Verify if the prediction service maintains response time within acceptable limits	1. Prediction service is accessible	The prediction service should maintain response time within acceptable limits