HIGH LEVEL DESIGN (HLD)
FLIGHT FARE PREDICTION

Document Version control

Date issued	Version	Description	Authors
05/13/2024	1	Initial HLD	GOBI KRISHNAN
			SRIRAM
			SHINY

Contents

ABSTRACT

The burgeoning demand for air travel necessitates accurate and reliable flight fare prediction models to aid passengers, travel agencies, and airlines in making informed decisions. This study explores the development of a robust flight fare prediction system utilizing machine learning techniques. By leveraging historical flight data, including features such as departure date, booking date, airline, route, and seasonal trends, we construct a model that predicts future flight prices with a high degree of accuracy.

The dataset comprises extensive flight fare records collected from various online travel platforms. Data preprocessing steps involve handling missing values, encoding categorical variables, and feature scaling. Several machine learning algorithms, including Linear Regression, Decision Trees, Random Forests, and Gradient Boosting, are employed to build predictive models. These models are evaluated based on their performance metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared (R^2) scores.

Our results indicate that ensemble methods, particularly Gradient Boosting, outperform other algorithms in terms of prediction accuracy and robustness. The model demonstrates significant potential in capturing complex patterns and interactions among features that influence flight fares. Additionally, feature importance analysis reveals that factors such as the booking lead time, day of the week, and airline have substantial impacts on fare prices.

This research contributes to the domain of airfare prediction by providing a scalable and efficient solution that can be integrated into travel recommendation systems and dynamic pricing engines. Future work will focus on enhancing model performance by incorporating real-time data and exploring advanced techniques such as deep learning and hybrid models. The implications of this study extend to better fare management, optimized pricing strategies for airlines, and cost savings for travelers.

1. Introduction

1.1 Why this High-Level Document?

The purpose of this High-Level Design (HLD) Document is to add necessary detail to the current project description to represent a suitable
model for coding. This document is also intended to help detect contradictions prior to coding, and can be used as a reference manual for how the modules interact at a high level.
The HLD will:

1. Present all of design aspects and define them in detail
2. Describe all user interface being implemented
3. Describe the hardware and software interfaces
4. Describe the performance requirements
5. Include design features and architecture of the project

List and describe the non-functional attributes like:
1. Security
2. Reliability
3. Maintainability
4. Portability
5. Reusability
6. Application compatibility
7. Resource utilization
8. Serviceability

1.2 Scope

The HLD documentation presents the structure of the system, such as database architecture, application architecture (layers), application flow (Navigation), and technology architecture. The HLD uses non-technical to mildly-technical terms which should be understandable to the administrators of the system.

2. General Description

2.1 Definitions

TERM	DESCRIPTION
FFP	Flight fare Prediction
IDE	Integrated Development Environment
API	Application Programming Interface

2.2 Product Perspective

The Flight Fare Prediction (FFP) Model is a critical component of a larger travel recommendation system. It leverages historical flight data to forecast future ticket prices, incorporating multiple factors such as travel date, departure and arrival times, route, duration, stops, and additional information. The model is designed to provide real-time price predictions to enhance the user experience on travel booking platforms.

Problem Statement

Travelling through flights has become an integral part of today's lifestyle as more and more people are opting for faster travelling options. The flight ticket prices increase or decrease every now and then depending on various factors like timing of the flights, destination, and duration of flights various occasions such as vacations or festive season. Therefore, having some basic idea of the flight fares before planning the trip will surely help many people save money and time. The main goal is to predict the fares of the flights based on different factors available in the provided dataset.

2.4 Proposed Solution

The proposed solution involves developing a machine learning model to predict flight prices based on a variety of input features. The goal is to provide accurate and real-time fare forecasts to users, travel agencies, and airlines. The solution encompasses data collection, preprocessing, model development, deployment, and continuous improvement.

2.5 Further Improvements

1.Advanced Feature Engineering

Interaction Features: Create interaction terms between features that could have combined effects on the flight prices.

Temporal Features: Incorporate features like day_of_week, is_holiday, and season to capture temporal patterns.

Lagged Features: Use historical prices or trends as lagged features to help the model understand past price behaviors.

Weather Data: Integrate weather conditions at the source and destination as features, as they can impact flight prices.

2. Model Optimization

Hyperparameter Tuning: Perform more exhaustive hyperparameter tuning using techniques like Bayesian Optimization or Hyperopt to find the best parameters for the XGBoost model.

Ensemble Methods: Combine predictions from multiple models (e.g., XGBoost, Random Forest,

and Neural Networks) using ensemble techniques like stacking, bagging, or blending to improve accuracy.

Neural Networks: Explore advanced neural network architectures, such as Long Short-Term Memory (LSTM) networks, which can capture temporal dependencies in the data.

3. Data Augmentation and Enrichment

Additional Data Sources: Incorporate additional data sources such as airline promotions, fuel prices, geopolitical events, and economic indicators.

Real-time Data: Integrate real-time data streams to ensure the model is updated with the latest information, potentially using APIs from flight tracking services.

4. Model Deployment Enhancements

Auto-scaling: Implement auto-scaling features in the cloud deployment to handle varying loads efficiently.

Caching Predictions: Use caching mechanisms to store and quickly retrieve frequently requested predictions, reducing response times.

Batch Predictions: For users requiring predictions for multiple routes or dates, implement batch processing to handle large requests efficiently.

5. Improving User Experience

Personalization: Customize fare predictions based on user profiles, travel history, and preferences.

Price Alerts: Implement a feature that allows users to set alerts for price drops on specific routes or dates.

Visualization: Provide users with visual insights, such as price trends over time, to help them make informed decisions.

6. Model Interpretability

Explainable AI: Use techniques like SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-agnostic Explanations) to provide interpretability for the model's predictions. This can help build trust and provide insights into what factors are driving fare changes.

Feature Importance Analysis: Regularly update and display feature importance to understand which factors are most influencing the predictions.

7. Performance Monitoring and Maintenance

Drift Detection: Implement mechanisms to detect data drift and model performance degradation over time, ensuring timely updates and retraining of the model.

Regular Updates: Set up a schedule for regular retraining of the model using the latest data to maintain accuracy.

A/B Testing: Conduct A/B testing when deploying new model versions to compare their performance against the current model.

8. Scalability and Robustness

Distributed Computing: Utilize distributed computing frameworks like Apache Spark for handling large-scale data processing and model training.

High Availability: Ensure the system is highly available and fault-tolerant by using redundant systems and failover mechanisms.

9. Security and Compliance

Enhanced Security: Implement advanced security measures such as multi-factor authentication, end-to-end encryption, and regular security audits.

Compliance Updates: Continuously monitor and update the system to comply with new regulations and standards in data protection and privacy.

10. Feedback Loop and User Engagement

User Feedback Integration: Create a feedback loop where users can provide input on the accuracy and usefulness of the predictions, helping to continuously improve the model.

Community Engagement: Foster a community of users who can share tips, insights, and travel hacks, enriching the data ecosystem and enhancing the model's effectiveness.

2.6 Technical Requirements:

1. Hardware Requirements

Development and Testing

Processor (CPU):Intel Core i7 or AMD Ryzen 7 (Quad-core or better)

Memory (RAM):16-32 GB

Storage:512 GB - 1 TB SSD for fast read/write operations

Graphics (GPU):NVIDIA GTX 1060/1660 or equivalent (optional for model training)

Network:Stable broadband internet connection

Production and Deployment

Server Specifications:

Processor (CPU): Dual Intel Xeon or AMD EPYC processors

Memory (RAM): 64-128 GB

Storage:1 TB NVMe SSD for the operating system and frequently accessed data

Additional 2-4 TB SSD/HDD for storing large datasets

Graphics (GPU): NVIDIA Tesla V100/A100 for high-performance computing

Network: High-speed, redundant internet connections

2. Software Requirements

Development Tools

Operating System:Windows 10/11, macOS, or a Linux distribution (e.g., Ubuntu, CentOS)

Programming Languages:Python 3.8 or higher

Development Environment:Jupyter Notebook/JupyterLab for exploratory data analysis and prototyping

Integrated Development Environment (IDE) such as PyCharm, VS Code

Version Control:Git for version control and GitHub/GitLab for repository management Libraries and Frameworks.

Data Processing: Pandas, NumPy

Data Visualization:Matplotlib, Seaborn

Machine Learning:Scikit-learn, XGBoost, LightGBM, TensorFlow/Keras for advanced models

Model Deployment:Flask for building APIs

Database Management:Cassandra database

Cloud Services: AWS (e.g., EC2, S3, RDS), Google Cloud Platform for scalable infrastructure

2.7 Data Requirements:

Airline: Name of the airline operating the flight.

Date_of_Journey: The date of the journey.

Source: The departure city.

Destination: The arrival city.

Route: The route taken by the flight.

Dep_Time: The departure time.

Arrival_Time: The arrival time.
Duration: The total duration of the flight.
Total_Stops: The number of stops between the source and destination.
Additional_Info: Additional information about the flight.
Price: The ticket price (target variable).

2.8 Tools used

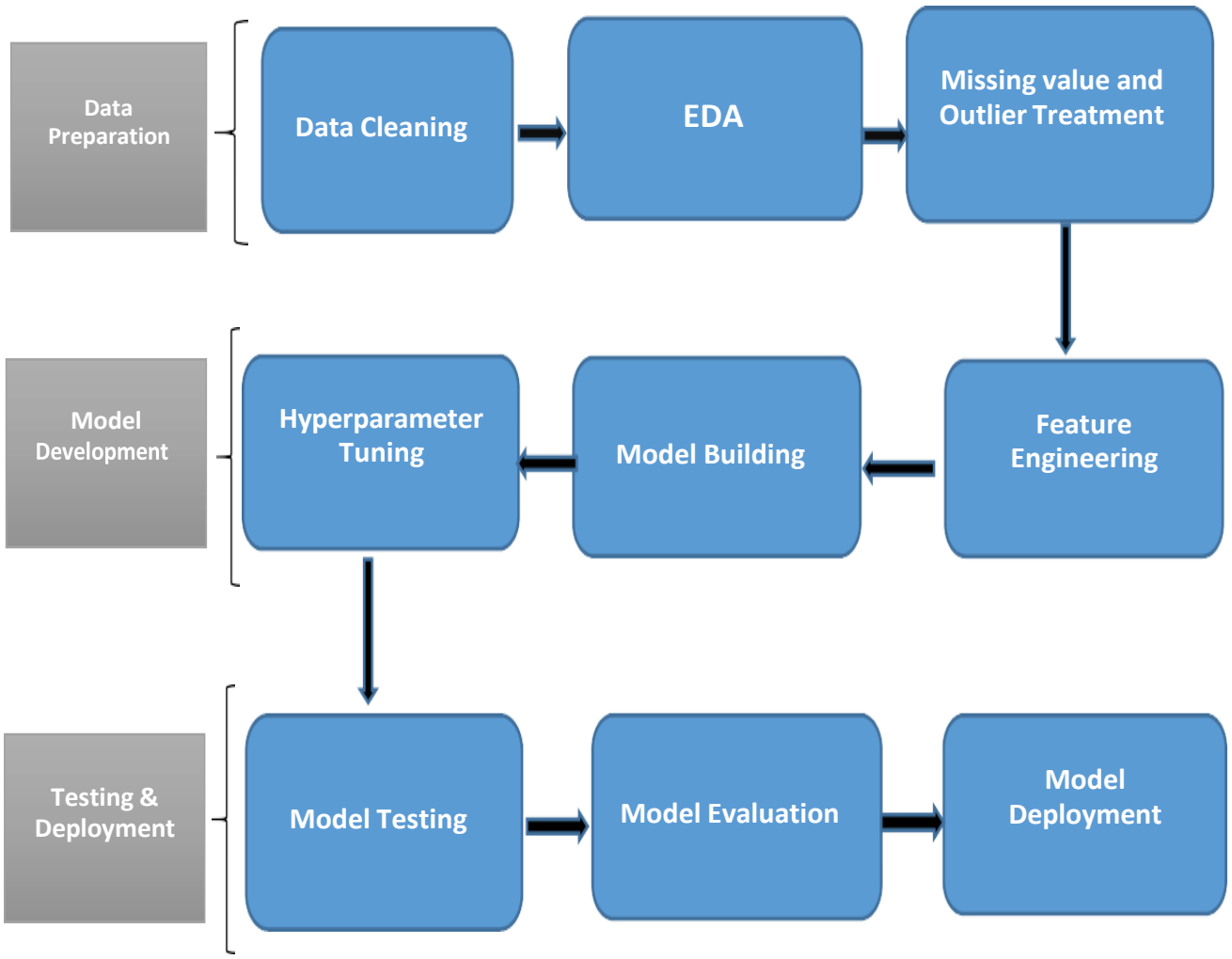
Python programming language and frameworks such as NumPy, Pandas, Scikit-learn, Matplotlib, Seaborn, Flask ,Jupyter Notebook, Visual Studio Code and a few other libraries were used to build the whole model.

1. Juyter notebook and Visual Studio code were used as IDE.
2. For visualization tasks, Matplotlib, Seaborn and plotly were used.
3. Flask were used for building the web application and server to run the code.
4. GitHub is used as version control system.
5. NumPy and Pandas were used to clean and interpret data.
6. Scikit learn was used to cross validate and compare different models.

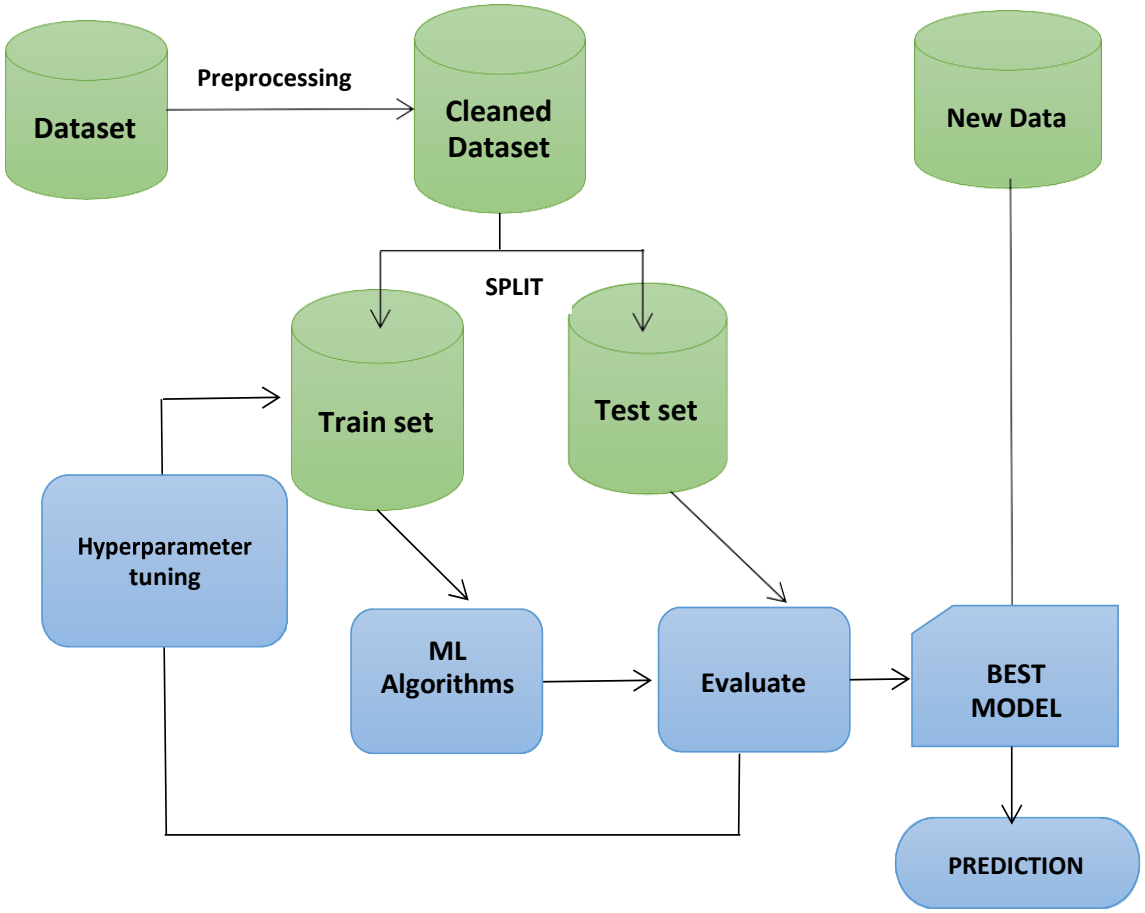
3. Design Details

3.1 Process Flow

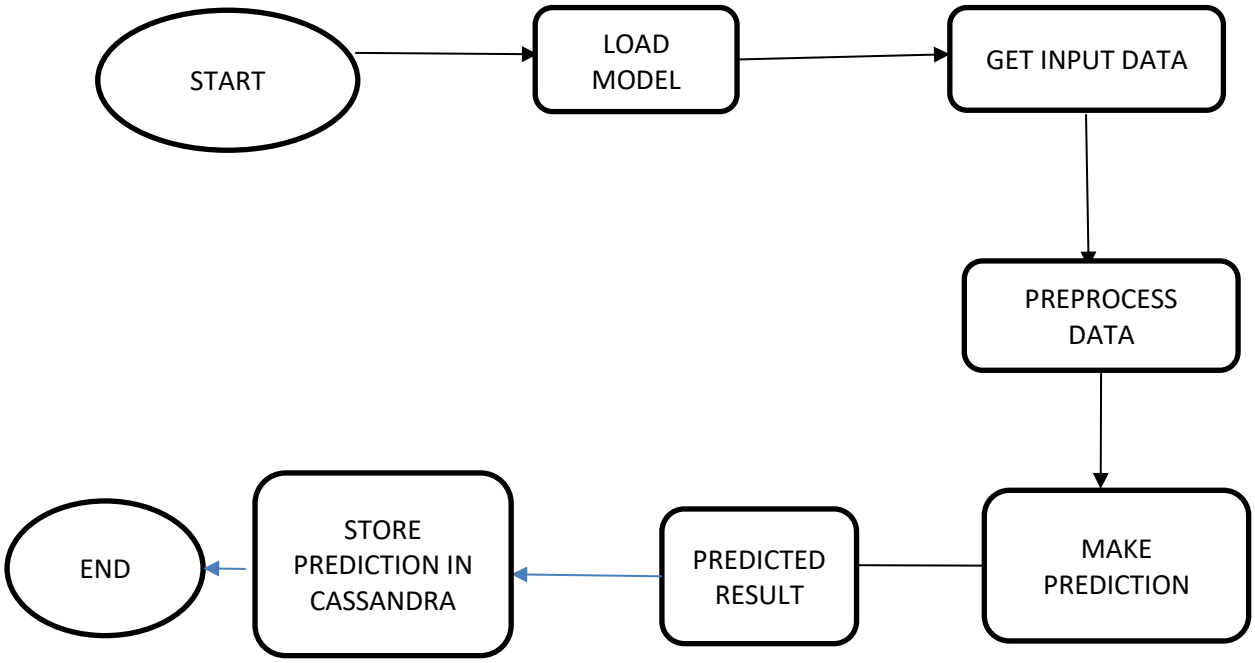
For accomplishment of the task, we will use a trained Machine Learning model. The process flow diagram is shown below :



3.1.1 Model Training and Evaluation



3.1.2 Deployment Process



3.2. EVENT LOG

An event log for a flight fare prediction project using an XGBoost model can help track key events and actions during the deployment and execution process.

1. The deployment process for the flight fare prediction project is initiated.
2. The trained XGBoost model is loaded from storage.
3. Input data for prediction, such as flight details, is received.
4. Data preprocessing begins, which includes handling missing values, encoding categorical variables, and scaling features.
5. Data preprocessing is completed.
6. The model prediction process starts using the preprocessed data.
7. The model prediction process is completed.
8. The predicted flight fare is outputted.
9. The system connects to the Cassandra database.
10. The process of storing the prediction results in the Cassandra database begins.
11. The prediction results are stored in the Cassandra database.
12. The deployment process is completed.
13. System monitoring is performed to ensure all systems are operational.
14. New data is received for model retraining.
15. Model retraining is initiated with the new data.
16. Model retraining is completed.
17. The updated model is deployed.

3.2.1ERROR HANDLING

For the flight fare prediction project using the XGBoost model, error handling encompasses several key areas to ensure robust operation. When loading the model, potential errors such as file not found or corruption are handled by verifying paths, using backups, and logging details. During data collection, invalid input data and network issues are managed by validating data formats, retrying retrievals, prompting users for correct data, and logging errors. In data preprocessing, handling missing values and incorrect data types involves imputation, validation, and conversion, with fallback notifications and logging. Prediction errors, like model failures and resource limitations, are addressed by catching errors, providing defaults, optimizing resources, and logging. For database operations with Cassandra, connection and read/write failures are mitigated by retrying operations, notifying users, and detailed logging. Finally, performance degradation and system failures are monitored and managed through regular model retraining, robust monitoring, alert systems, disaster recovery plans, and comprehensive logging of performance metrics and system issues. This integrated error handling strategy ensures system resilience and continuous operation.

4.PERFORMANCE

The performance of the flight fare prediction system hinges on several key factors. Firstly, the accuracy of the prediction model, gauged by metrics like Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE), ensures precise fare estimations. Secondly, optimizing latency, measured by the time taken for predictions, guarantees a swift user experience, while throughput, denoting the system's capacity for concurrent predictions, ensures smooth operation during peak loads. Efficient data preprocessing streamlined data storage, and retrieval from the Cassandra database contribute to swift processing. System-wide resource utilization, scalability, and reliability, monitored through CPU, memory, and disk usage, enable continuous, high-performance operation. Lastly, continuous monitoring, periodic model retraining, and comprehensive logging and auditing maintain system efficiency and reliability over time. This integrated approach ensures the flight fare prediction system delivers accurate predictions swiftly and reliably to users.

4.1 DEPLOYMENT

5.CONCLUSION

In conclusion, the flight fare prediction project presents a robust and efficient system for estimating flight fares accurately and swiftly. By focusing on key performance metrics such as model accuracy, latency, throughput, and system-wide resource utilization, the system ensures reliable operation and a seamless user experience. Through efficient data preprocessing, optimized database operations, and scalable system architecture, the project delivers swift and accurate predictions even under high demand. Continuous monitoring, periodic model retraining, and comprehensive logging further ensure the system's reliability and effectiveness over time. Overall, the flight fare prediction system stands as a reliable tool for users, providing accurate fare estimates efficiently and consistently.