Low-Level Design (LLD)

Investment Prediction

|  |  |
| --- | --- |
| Written By | Divit |
| Document Version | 2.0 |
| Last Revised Date | 19 -october - 2024 |

# Document Control

Document Version Control

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Description |
| 1.0 | 2024-08-10 | Divit | Initial version of the document. |
| 2.0 | 2024-09-  12 | Divit | Updated Unit Test Cases and document structure |

Approval Status:

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

# Table of Contents

1. Introduction
   * 1.1 What is a Low-Level Design Document?
   * 1.2 Scope
2. Architecture
3. Architecture Description
   * 3.1 Data Description
   * 3.2 Data Preprocessing
   * 3.3 Feature Engineering
   * 3.4 Model Training
4. Unit Test Cases

# Introduction

## What is a Low-Level Design Document?

## A **Low-Level Document (LLD)** for an **investment prediction project** is a comprehensive technical blueprint that provides detailed specifications and implementation steps for building the project. It translates the broader objectives and architectural overview from the High-Level Design (HLD) into actionable tasks and system components for developers, data engineers, and data scientists. The document meticulously outlines how each module of the system—such as data ingestion, preprocessing, feature engineering, machine learning model development, and visualization—will be implemented. It includes precise details about the tools, frameworks, programming languages, and databases to be used.

## Scope

The LLD focuses on the design of key components, such as data preprocessing, model training, feature engineering, API services, database interactions, and logging. This document provides the necessary technical details to implement the Investment Prediction system. It covers:

* + - Data structure and storage
    - Data preprocessing and feature engineering techniques
    - Machine learning model architecture and training process
    - API design and implementation
    - Database interactions
    - Testing strategies

The document does not cover deployment strategies, user interface design, or system scalability considerations, as these are typically addressed in separate documents.

# Architecture

Here's a simplified architecture diagram showing the data flow through the Investment Prediction system:

+

|

|

|

|

+

+

|

+

|

+

|

+

|

Raw Data

(CSV)

+---->+ Data Preprocessing+---->+ Feature

|

|

+

|

|

+

|

|

+

| Engineering

|

+

+

|

| v

+

|

|

|

|

+

+ + + + +

| | | | |

| Cassandra DB|< +

Model Training

(lstm)

<-----+ Processed Data

|

|

+

+

|

|

+

|

|

+

|

|

+

|

|

+

+

|

|

|

|

+

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | ^  |  | | |  | | | | |
| +  | | + | | +  | | +  | | +  | | | |
| |  |  + | Flask | API | +---->+  | |  + + | |  | Prediction | |  |  + |

# Architecture Description

## Data Description

The Flight Fare Prediction system uses a dataset containing historical flight fares and relevant parameters. The data is stored in Cassandra DB, a NoSQL database known for its scalability and performance with large datasets.

Key data fields include:

* + - Date of journey
    - Source and Destination
    - Airline
    - Number of stops
    - Duration
    - Price (target variable) Data schema in Cassandra:

CREATE TABLE stock\_prices (

stock\_symbol TEXT,

trading\_date DATE,

open\_price DECIMAL,

close\_price DECIMAL,

high\_price DECIMAL,

low\_price DECIMAL,

volume BIGINT,

PRIMARY KEY (stock\_symbol, trading\_date)

) WITH CLUSTERING ORDER BY (trading\_date ASC);

## Data Preprocessing

Data preprocessing is a crucial step to ensure the quality and consistency of the input data. The preprocessing module performs the following operations:

### Handling missing values:

* + For numerical columns: Impute with median or mean
  + For categorical columns: Impute with mode or create a new category for missing values

1. **Date parsing**: Convert the 'Date of Journey' into multiple features like day of week, month, and year.

### Encoding categorical variables:

* + Use one-hot encoding for stock prices
  + If there are many categories, consider using frequency encoding or target encoding

### Handling outliers:

* + Use Interquartile Range (IQR) method to detect and handle outliers in numerical features

Python code snippet for handling missing values:

import pandas as pd import numpy as np

def handle\_missing\_values(df): for column in df.columns:

if df[column].dtype == 'object': df[column].fillna(df[column].mode()[0], inplace=True)

else:

df[column].fillna(df[column].median(), inplace=True) return df

## Feature Engineering

Feature engineering involves creating new features or transforming existing ones to improve the model's predictive power. Key feature engineering steps include:

* **Generate Features:**
  + **Economic Indicators:** Combine macroeconomic indicators like GDP, inflation rates, or interest rates.
  + **Market Sentiment:** Aggregate sentiment scores from social media or news for specific stocks.
  + **Technical Indicators:** Add features like Relative Strength Index (RSI) or Bollinger Bands.
* **Feature Selection:**
  + Use methods like correlation analysis or feature importance from models to eliminate irrelevant features.

## Model Training

The FInvestment Prediction system uses a LSTM model for predicting stock prices.

Key steps in model training:

### Data splitting:

* + Split the preprocessed data into training (70%), validation (15%), and test (15%) sets

### Hyperparameter tuning:

* + Use LSTM to find the optimal hyperparameters
  + Key hyperparameters to tune: n\_estimators, max\_depth, min\_samples\_split, min\_samples\_leaf

### Model training:

* + Train the LSTMmodel with the best hyperparameters on the training data

### Model evaluation:

* + Evaluate the model on the validation set using metrics like Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE)

### Feature importance analysis:

* + Analyze feature importances to understand which factors most influence fare predictions

Python code snippet for model training:

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import LSTM

model=Sequential()

model.add(LSTM(50,return\_sequences=True,input\_shape=(100,1)))

model.add(LSTM(50,return\_sequences=True))

model.add(LSTM(50))

model.add(Dense(1))

model.compile(loss='mean\_squared\_error',optimizer='adam')

# Unit Test Cases

Here we define unit test cases for each core component of the Investment Prediction system:

|  |  |  |
| --- | --- | --- |
| Test Case Description | Pre-Requisite | Expected Result |
| Validate successful API call | Valid API key and request params | 200 HTTP status with JSON response |
| Handle missing value | Dataset with missing rows | Missing values filled/dropped |
| |  | | --- | |  |  |  | | --- | | Validate outlier detection | | Dataset with known outliers | Outliers flagged or removed |
| Ensure normalization works | Dataset with varied scales | All features normalized to [0, 1] |
| Verify feature encoding | Categorical data | One-hot or label-encoded features |
| |  | | --- | |  |  |  | | --- | | Validate feature generation | | Raw time-series data | Derived features (e.g., moving avg |
| Validate model training | |  | | --- | |  |  |  | | --- | | Processed dataset | | Successfully trained model |
| |  | | --- | |  |  |  | | --- | | Handle insufficient data | | Minimal dataset | Appropriate error or warning |
| |  | | --- | |  |  |  | | --- | | Test hyperparameter tuning | | Dataset and parameter grid | Model trained with best param |
| Check training time | Large dataset | Training completes within SLA |

|  |  |  |
| --- | --- | --- |
|  |  |  |
| |  | | --- | |  |  |  | | --- | | Validate single prediction | | Single feature set | Correct predicted value |
| Ensure batch prediction | Batch of feature sets | Predictions for all inputs |
| Validate data rendering on charts | Sample predictions | Correct display on graph |
| |  | | --- | |  |  |  | | --- | | Test data filtering functionality | | |  | | --- | | functionality |  |  | | --- | | Filter parameters | | Filtered view of data |
| Test prediction function with sample input | Trained model is saved and loaded | Prediction function should return a reasonable fare estimate for given input |
| Validate model serialization and deserialization | Trained model is available | Model should be successfully saved to and loaded from disk |
| Ensure responsivenes | Various screen sizes | UI adapts to screen size |
| Test user interactions | Button clicks, dropdown inputs | Corresponding actions performed |
| Test logging for errors | Error in data preprocessing | Error logged with detail |
| Validate performance metrics logging | Model training process | Metrics logged (e.g., time, accuracy) |

These comprehensive unit tests cover the core components of the Investment Prediction system, focusing on data processing, feature engineering, and model performance. They ensure that each part of the data pipeline and modeling process functions as expected, helping to catch potential issues early in the development process.