Low Level Design LLD Documentation

**LOW LEVEL DESIGN(LLD) DOCUMENT**

**FLIGHT FARE PREDICTION PROJECT**

**Revision Number: 1.0**

**Last Date of Revision: 09/08/2022Document Version Control**

| **Date Issued** | **Version** | **Description** | **Author** |
| --- | --- | --- | --- |
| **09/08/2022** | **1.0** | **Initial HLD** | **Mobo Somto** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Table of contents**

[**Introduction**](#_540h7kkhk5pp) **4**

[1.1 What is a Low Level Design(LLD) Document?](#_9vu7fxdk02o7) 4

[1.2 Scope](#_m0nf4iwuqiz0) 4

[**Nomenclature And Procedures To Be Followed**](#_1h5hjn16k5r7) **5**

[2.1 Nomenclature To Be Adhered To During The Project Work](#_7930plruzs) 5

[2.2 Procedures To Be Followed During The Project Work](#_m953aa80obus) 5

[**Architecture**](#_ka0vxqqg69ns) **7**

[**Architecture Description**](#_1y8xuszgf9kv) **8**

[4.1 Dataset Description](#_8g1wnn59e5e6) 8

[4.2 Data Validation](#_7kyzhqws71on) 8

[4.3 Data Transformation](#_1f4fidhoplth) 9

[4.4 Model Training and Model Evaluation](#_wnr9lplmxqs9) 9

[4.5 Cloud Setup and Deployment](#_odv0jm8o4clg) 9

[4.6 Taking Input from the User](#_gjnokzzdz4b1) 9

[4.7 Viewing logs](#_fhdtdty121zd) 10

[4.8 Viewing Artifacts](#_aouc2spsl0gg) 10

[4.9 Training a model and Updating Model Config](#_8oxhnefkl99b) 10

[4.10 Viewing Experiment History](#_hbxcxukf3i9i) 10

# Introduction

## 1.1 What is a Low Level Design(LLD) Document?

The goal of this low level design document is to describe the internal logical design of the actual code used to implement the Flight Fare system project. The document describes the class diagram with the methods and relationships that exist among other classes and program specifications. It describes the modules so that any 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 organisation may be defined during requirement analysis and then refined during data design work.

# Nomenclature And Procedures To Be Followed

## 2.1 Nomenclature To Be Adhered To During The Project Work

The Flight Fare project was built as a Python package using Python’s built in module setuptools as such:

* All packages were created in lower case letters.
* All directories and subdirectories follow a lower case naming convention separated by “\_”.
* All class names follow a Camel Case format.
* All functions and method names follow a lower case naming convention separated by “\_”.
* All Machine learning models follow the convention:
  + “**{yyyymmddhms}**”
* File names contained in “flight\_logs” directory follow the convention:
  + “**{yyyy-mm-dd\_H:M:S}.log**”
* Directory names contained within an artifact follow the convention:
  + “**{yyyy-mm-dd-H-M-S}**”
* Imports follow any of the following formats:
  + **from “{package\_name}.{directory\_name}.{module\_name} import class\_name**
  + **from “{package\_name}.{directory\_name}.{function\_name} import func\_name**

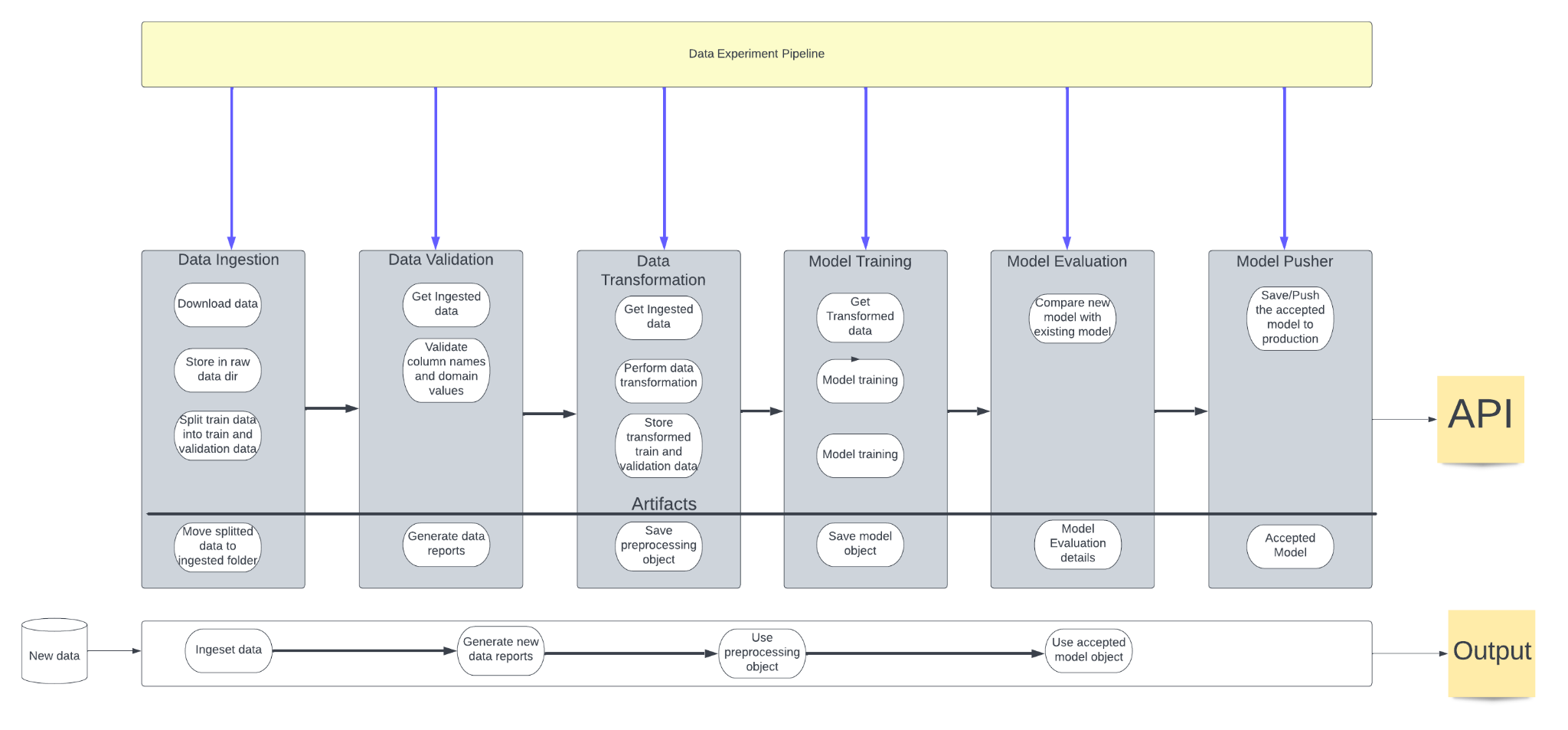
## 2.2 Procedures To Be Followed During The Project Work

The project was built in the form of a Python package called “flight” whereby all the processes/stages required for data science project life cycle have been created. It includes data ingestion, data validation, model training, model evaluation, model pusher and overall a pipeline to carry out all the above processes.

The entire life cycle is run by a pipeline and each process/stage accepts a set of inputs gotten from config/config.yaml, which would be used by “flight/config/configuration.py” to set the configuration details of a component. Each component generates an artifact which can/or will be utilized by other components.

* All python files relating to error handling of the entire system should be stored in the **“flight/exception/”** folder.
* All python related to logging the components of the system should be stored in the **“flight/logger/”** folder
* All python files relating to structure/entities should be stored in the **“flight/entity/”** folder.
* All python files relating to input data needed for the configuration of any component should be stored in the **“.config/”** folder.
* All python files relating to component configuration should be stored in the **“flight/config/”** folder.
* All python files relating to core processes (i.e: data ingestion to model deployment) involved in the data science lifecycle should be stored in the **“flight/component/”** folder.
* All variables needed by the system should be stored in the **“flight/constants/”** folder.
* All artifacts produced by any component should be stored in the **“flight/artifact/”** folder. NB: “artifacts are the output after a component has executed successfully.
* All python files relating to running the entire pipeline should be stored in the **“flight/pipeline/”** folder.
* All python classes/functions not related to any of the core components of a given module should be stored in the **“flight/utils/”** folder
* Machine learning models would be stored in a folder “.**saved\_models/**”

# Architecture



# Architecture Description

## 4.1 Dataset Description

Based on the format of the given dataset any new or incoming dataset should follow the following format:

1. The data must be provided in excel format (.xlsx).
2. The file must contain 11 columns for training and 10 columns (except target column) for predictions. Below are the data types and description of the dataset columns:
3. “X1”(Object): Airline

Name of various airlines.

1. “X2”(Object): Date\_of\_Journey

Scheduled date for the flight.

1. “X3”(Object): Source

Location of flight before take off.

1. “X4”(Object): Destination

Location of flight after landing.

1. “X5”(Object): Route

Airline path followed

1. “X6”(Object): Dep\_Time

Time that the flight took off.

1. “X7”(Object): Arrival\_Time

Time that the flight landed.

1. “X8”(Object): Duration

Flight time taken from source to destination.

1. “X9”(Object): Total\_Stops

No of stops made by the flight before reaching its destination.

1. “X10”(Object): Additional\_Info

Information about the flight. (e.g: In flight meal)

1. “X11”(Numerical): Price

Cost of boarding the flight.

## 4.2 Data Validation

After the data has been ingested, the validations that will be performed on the dataset include column validation and domain value validation.

Column validation involves comparing the column names of the new dataset with the column names stored in the config/schema.yaml file to ensure they are correct. While domain value ensures that unexpected values are not imputed in a column.

## 4.3 Data Transformation

The following transformations were performed on the data. Missing value imputation, scaling, extraction of individual date values from columns which contain date values, encoding categorical variables and conversion of time related values to a constant format (minutes). All these transformations were done using Column Transformers, after which we save the preprocessing object which will be utilized for a new dataset, and the preprocessed data in numpy array format.

## 4.4 Model Training and Model Evaluation

After the data has been processed, we begin testing models on our dataset and based on the specified model accuracy a model is chosen as the accepted model.

Now in a situation where there is no model already in production, we deploy the newly trained model to production. However, if a model already exists in production, we compare the newly trained model with the already existing model and based on the better model we choose one of the two of which we can deploy to production.

## 4.5 Cloud Setup and Deployment

For deployment, Github, Github Actions, Docker and Heroku were used to deploy the Web Application to the web. This ensures that an MLOp’s approach was used in creating the application so as to keep the application up to date based on any modifications made to the code.

## 4.6 Taking Input from the User

Through the web application we take input from the user in the form of a form which is made available on the web application. Now based on the model received from training we use the model to make flight fare predictions which would be outputted to the user.

## 4.7 Viewing logs

The Web Application provides a means for the system admin to observe and monitor the processes performed on the system using logs hyperlink provided on the application.

## 4.8 Viewing Artifacts

The application also generates artifacts as a result of running an experiment. You can view the artifacts produced by each process by clicking on the artifact hyperlink.

## 4.9 Training a model and Updating Model Config

The application provides a direct means of training a model by selecting your model and editing the parameters from the application. In order to tune your model simply navigate to update model config and tweak the parameters which you want your model to have, copy the json and insert in the update model config section to update your model configurations.

Finally, simply click on train a new flight estimator to generate a new model.

**NB**: Training a model takes time.

## 4.10 Viewing Experiment History

This section of the application provides a history of all the experiments performed on the application to train a new model.