**Gemstone Price Prediction – High Level Design Document**

**Objective of the Project:**

The objective of this project is to predict the price of gemstones, given certain attributes of the gemstone. A company would use this prediction to distinguish between higher profitable stones from lower profitable stones.

**About the Data used in the Project:**

The dataset contains the following attributes:

* Carat
* Cut
* Color
* Clarity
* Depth
* Table
* X
* Y
* Z
* Price

“Price” is the target feature.

**Project Components:**

The project component section contains all the modules for the project.

**Config Entity:**

The config entity module contains the following classes:

1. DataIngestionConfig():
   1. Path in which the train and validation split will be stored.
2. DataTransformationConfig():
   1. The path to store the preprocessing object.
3. StoreFeatureConfig():
   1. The path that will store the transformed datasets
4. ModelTrainerConfig():
   1. The path to store the trained model.

**Data Ingestion:**

The data ingestion module reads the train data from the source and then splits the data into the train and validation set. Once the train data has been split into the train and validation set, the data is stored in the artifacts folder. All data is stored as parquet files.

To split the data, a class – DataIngestion() – is created to read the data from source and split the data into a train and test dataset. This class contains two methods – the constructor and initiate\_data\_ingestion().

**Data Transformation:**

This module contains the class to transform the train and test datasets. The class contains two methods (other than the constructor) – create\_data\_transformation\_object() and initiate\_data\_transformation().

The create\_data\_transformation\_object() method creates the preprocessor object. The preprocessor object itself is a scikit-learn column transformer object, which is made up of several scikit-learn pipelines.

The initiate\_data\_transformation() method takes the train path and test path as arguments, transforms the datasets using the preprocessor object and returns the transformed datasets.

As a part of this transformation, the following additional features will be created (ALL of these features are calculated features):

* Volume
* Surface Area
* Aspect ratio of x and y
* Diagonal distance
* Relative height
* Relative position
* Volume ratio
* Length ratio
* Width ratio
* Height ratio

This feature creation is built into the data transformation pipeline.

**Store Features:**

This module contains the FeatureStoreCreation() class to store the transformed datasets in the path as defined by the StoreFeatureConfig class (from the config\_entity module).

The FeatureStoreCreation class contains the create\_feature\_store() method, which creates the feature store folder (if the folder does not already exist) and stores the transformed datasets into the newly created folder.

**Model Trainer:**

This module contains the ModelTrainer() class, which uses the training data and test data to create a trained model. The class contains two methods – create\_feature\_target\_datasets() and initiate\_model\_training().

The create\_feature\_target\_datasets() method creates the feature and target datasets from the transformed data.

The initiate\_model\_training() method trains the model and then saves the trained model in the path as defined by the ModelTrainerConfig class from the config\_entity module.

**Project Pipelines:**

The project uses three pipelines:

* Feature Pipeline:
  + The feature pipeline ingests the data, transforms the data and then stores the transformed data in the data store.
* Training Pipeline:
  + The training pipeline uses the ModelTrainer to train the model. The ModelTrainer uses the transformed data from the Feature Store.
* Predict Pipeline:
  + The predict pipeline uses data entered by the end user to make predictions on the price of the gemstone.

**Project Utilities File:**

The utilities file consists of functions that are used throughout the project.

Fetch\_error\_message() – This function fetches the error message from the python sys package and returns the information to the CustomException class (see – Custom Exception File section below).

Save\_Object() – This function takes a file path and an object and saves the object as a pickle file in the given file path.

Create\_aspect\_features() – This function builds additional features for the aspects features.

**Custom Exception File:**

For the project, a custom exception file was created to provide a more readable error message. This file comprises of a single class – CustomException – that inherits from the overall Python Exception class. This class takes the error message, fetched from the fetch\_error\_message() function, and then feeds the error message to the Parent Class (Exception class). Once the error message is obtained, it is displayed as a readable string via the python built-in \_\_str\_\_ method.