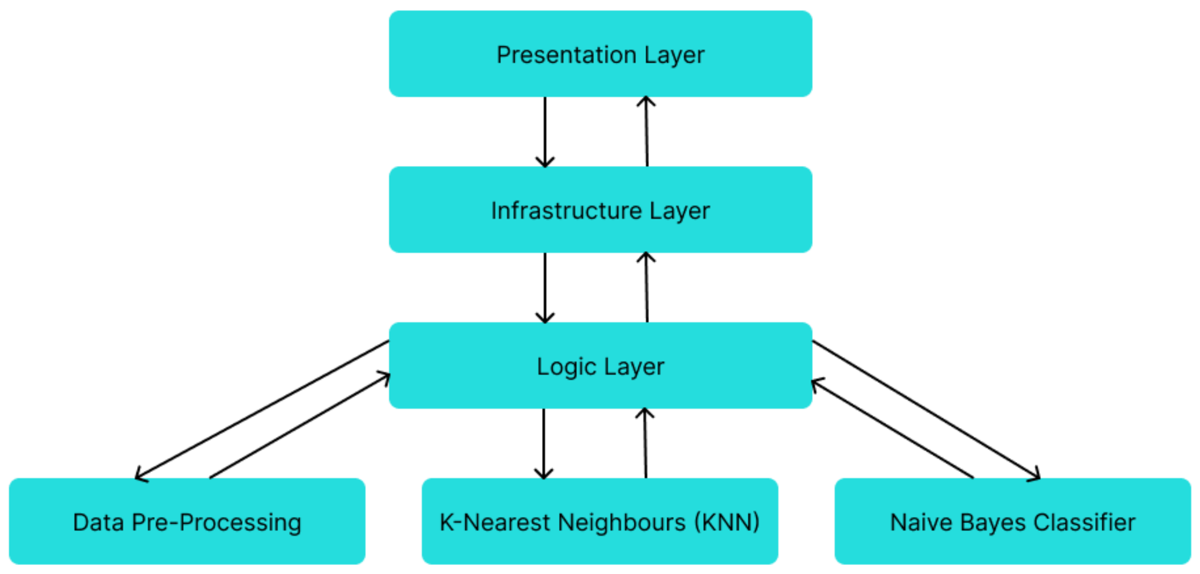
Software Design Architecture:

1. The proposed software design architecture for the Machine Learning Library is based on a modular and layered approach, ensuring flexibility, scalability, and maintainability. The architecture consists of several key components, each responsible for specific functionalities, and interfaces between these components are designed to be clear and well-defined.

By: Madhav Sood(IMT2021009), Kalyan Ram (IMT2021023), Nilay Kamat (IMT2021096), Shlok Agrawal (IMT2021103)

Architecture Diagram:



Layered Architecture:

1. **Components and Connectors:**

a. Presentation Layer:
   - User-Friendly API Component: This layer provides the Java API for user interaction, including functions for data preprocessing, model creation, training, and evaluation.

b. Infrastructure Layer:
   - This is the infrastructure that deals with communication between the presentation layer components and the logic layer components.
   - This layer ensures that the ML library can interface with the different logic layer components seamlessly and connect with the presentation layer.

c. Logic Layer:
    - Data Preprocessing Component: Responsible for logical operations related to data preprocessing, including loading, cleaning.
    - Naive-Bayes Classifier Component: Implements the Naive-Bayes classification algorithm, classification, evaluation, and customization.
    - K-Nearest Neighbors (KNN) Component: Implements the KNN classification algorithm, including training, classification, evaluation, model persistence.

2. Mapping to FRs and NFRs:

a. Presentation Layer:
    - FRs: Provides a user-friendly API for data scientists and developers to interact with the library's functionalities.
    - NFRs: Contributes to usability by offering a well-documented interface.

b. Infrastructure Layer:
    - FRs: Ensures seamless communication between logic layer components, supporting the overall functionality of the library.
    - NFRs: Contributes to scalability by enabling efficient interactions between components, especially in handling large datasets.

c. Logic Layer:
    - FRs: Addresses high-priority functional requirements, including data preprocessing, Naive-Bayes classification, and KNN classification.
    - NFRs: Aligns with performance requirements, ensuring efficient data preprocessing and model training.

3. Most Important Design Decisions:

a. Layered Structure: The decision to organise the architecture into layers ensures a separation of concerns, making the system modular and maintainable.
b. Logical Flow: The layered architecture facilitates a logical flow of operations from the presentation layer to the infrastructure layer and finally the logic layer ,promoting clarity and maintainability.

The layered architecture supports the development of a comprehensive machine learning library by providing a structured and organised framework. Each layer has a clear set of responsibilities, and connectors between layers ensure effective communication, contributing to the achievement of functional and nonfunctional requirements.

Detailed Design:

Detailed Design of Data Structures:

1. **Data Preprocessing Module:**
   a. **Data Loader:**
      - **Data Container:** Utilises a structured data container (e.g., Pandas DataFrame) to hold loaded data.
      - **Error Handling:** Incorporates data structures for handling errors during loading.

   b. **Data Cleaner:**

      - **Missing Values Handling:** Uses appropriate data structures (e.g., data masks) to represent and manage missing values.
      - **Error Log:** Maintains a log of errors encountered during cleaning.
2. **Naive-Bayes Classifier Module:**
   a. **Training Data:**
      - **Probability Distributions:** Uses data structures (e.g., dictionaries) to store probability distributions.
      - **Conditional Probabilities:** Maintains structured data for conditional probabilities.

   b. **Classification Data:**

      - **Prediction Storage:** Utilises data structures to store classification information for efficient retrieval.
3. **K-Nearest Neighbors (KNN) Module:**
   a. **Training Data:**
      - **Labelled Data Storage:** Organises labelled training data efficiently for retrieval.
      - **Distance Matrix:** Uses a data structure (e.g., matrix) to store distances between data points.

   b. **Neighbour Information:**

      - **Nearest Neighbors List:** Maintains a list of nearest neighbours for each data point.
      - **Distance Storage:** Utilises data structures for storing distances between data points.

4. **Model Training and Prediction Module:**
   a. **Model Storage:**
      - **Naive-Bayes Model:** Utilises data structures specific to the Naive-Bayes model representation.
      - **KNN Model:** Uses data structures specific to the KNN model representation.

   b. **Prediction Data:**

      - **Input Data Representation:** Represents input data in a structured format suitable for model prediction.

- **Output Data Format:** Defines the structure for storing and presenting prediction results.

These data structures provide a foundation for implementing the detailed design of the Machine Learning Library, ensuring efficient and organised handling of data throughout the various modules. Each module's data structures are designed to meet specific functional requirements and enhance the overall performance and usability of the library.

Detailed Design of Algorithms:

1. **Naive-Bayes Classifier Module:**
   a. **Naive-Bayes Training Algorithm:**
      a. **Overview:** The algorithm trains a Naive-Bayes classifier on labelled data.
      b. **Steps:**
         i. Calculate class priors and likelihoods.
         ii. Use Laplace smoothing for handling zero probabilities.
         iii. Store probability distributions.
         iv. Return the trained Naive-Bayes model.

   b. **Naive-Bayes Classification Algorithm:**

      c. **Overview:** The algorithm classifies new data points using the trained model.
      d. **Steps:**
         i. Calculate class probabilities for each class.
         ii. Multiply likelihoods with class priors.
         iii. Assign the class with the highest probability.
         iv. Return the predicted class.
2. **K-Nearest Neighbors (KNN) Module:**
   a. **KNN Training Algorithm:**
      a. **Overview:** The algorithm organises labelled data for efficient retrieval during prediction.
      b. **Steps:**
         i. Store labelled training data.
         ii. Optionally, use a distance matrix (e.g., KD-tree) for faster neighbour searches.
         iii. Return the prepared KNN model.

   b. **KNN Classification Algorithm:**

      c. **Overview:** The algorithm classifies new data points using the KNN model.
      d. **Steps:**
         i. Calculate distances to all training data points.
         ii. Identify the k-nearest neighbours.
         iii. Assign the class based on majority voting.
         iv. Return the predicted class.
3. **Model Training and Prediction Module:**
   a. **Unified Model Training Algorithm:**
      a. **Overview:** The algorithm trains either the Naive-Bayes or KNN model based on user selection.

b. **Steps:**
   i. Check the selected model type.
   ii. Call the respective training algorithm.
   iii. Return the trained model.

b. **Unified Prediction Algorithm:**

c. **Overview:** The algorithm makes predictions using either the Naive-Bayes or KNN model.
d. **Steps:**
   i. Check the selected model type.
   ii. Call the respective classification algorithm.
   iii. Return the predicted class.

Detailed Design of APIs:

1. **Data Preprocessing API:**
   a. **Functions:**
      - `load_data(source_type, source_path)`: Loads data from the specified source.
      - `clean_data(data, cleaning_strategy)`: Handles missing values and cleans the data.
2. **Naive-Bayes Classifier API:**
   a. **Functions:**
      - `train_naive_bayes_model(training_data)`: Trains a Naive-Bayes model.
      - `classify_naive_bayes(new_data)`: Classifies new data using the trained Naive-Bayes model.
      - `evaluate_naive_bayes_model(test_data)`: Evaluates the Naive-Bayes model on test data.
3. **K-Nearest Neighbors (KNN) API:**
   a. **Functions:**
      - `train_knn_model(training_data)`: Trains a KNN model.
      - `classify_knn(new_data)`: Classifies new data using the trained KNN model.
      - `evaluate_knn_model(test_data)`: Evaluates the KNN model on test data.
4. **Model Training and Prediction API:**
   a. **Functions:**
      - `train_model(model_type, training_data)`: Trains either the Naive-Bayes or KNN model based on the specified type.
      - `predict(model_type, new_data)`: Makes predictions using the specified model type.

UI Design:

The application we are building only needs a terminal-based user interface as the target audience is developers who will use the ML library in their applications.
Given that the target audience is developers, the terminal-based user interface should prioritise simplicity, clarity, and efficiency. Developers often appreciate concise and informative outputs that can be easily integrated into their workflow

The main objective is to format the output that is aesthetically pleasing and easy to infer.