**Project Planning:**

* Purpose:

This Project describes a software application that uses the user's tumor description to forecast whether a patient has cancer or not. The program calculates the user's cancer stage using Random Forest model of machine learning.

* System description:

The application uses an API to take the tumors characteristic features and then run it through the model to predict the output. The inputs taken are mean\_radius , mean\_texture and mean\_perimeter.

Based on the input, the programme uses a predefined model mapped to the database to categorise the user's condition. For example, if the user has the tumour conditions which maps that the user has cancer that is the dependent variable is 1 then the output is given that the user has cancer and should consult a doctor.

## **External Interface Requirements**

**1. User Interfaces (UI)**

* **Data Input:**
  + Users will be able to enter patient data that is pertinent to the prediction of breast cancer using the system's user interface. This have attributes like:
    - * The quantity of breast biopsies
      * kind of breast cell
      * Results of mammography (text entry or picture uploading for future improvements)
      * characteristics of tumors, such as texture and size.
  + The system offers an interface where the developer has already chosen a pre-trained machine learning model for the purpose of predicting breast cancer.
  + Users with different levels of technical competence is be able to enter data with ease because to the interface's ease of use and intuitiveness.
  + To find and alert users to inaccurate or missing data entries, error handling is put into place.
  + Clear explanations of the available models, together with information on their accuracy and performance metrics, is available in the EDA of the model.
* **Prediction Output:**
  + The estimated likelihood that a patient would get breast cancer is shown by the system.
  + The interface clearly and succinctly displays the results, including confidence scores and explanations.

**2. Hardware Interfaces**

* There are very few external hardware requirements on the system.
* But it can support integration with future enhancements like:
  + Specialized medical imaging devices (for directly scanning the cancer area and taking the required featural characteristic input of the area.

**3. Software Interfaces (SI)**

* **Machine Learning Libraries:**
  + For cancer prediction tasks, the system leverages machine learning packages such as scikit-learn, PyTorch, and TensorFlow.
* **Database Interface:**
  + The system interfaces with a database that contains historical patient data.
* **External APIs:**
  + The Web application that connects the program and device is created by the system using PyScript.

**4. Communications Interface (CI)**

* The system's primary mode of communication shall be through its user interface.
* Secure communication protocols may be implemented by the system to facilitate interfaces with external APIs.

**Future Enhancements:**

* Integration for smooth data sharing with hospital information systems.
* Support for multiple languages.

**Functional Requirements:**

* + - Breast Cancer prediction using Naive Bayes Model:

**1. Library Imports:**

* + - * Imported the NumPy library for numerical computations.
      * Imported the Pandas library for data manipulation and analysis.
      * Imported the Matplotlib library for creating visualizations
      * Imports the Seaborn library for statistical data visualization

**2. Data Loading:**

* + - * Reading the breast cancer data from a CSV file named "Breast\_cancer\_data.csv" and storing it in a Pandas Data-Frame.

**3. Data Splitting:**

* + - * Importing the “train\_test\_split” function from scikit-learn model selection module.
      * Creating a new Data-Frame “x” containing all features (columns) except the target variable "diagnosis".
      * Separating the target variable "diagnosis" into a Series named “y”.
      * Splitting the data into training and testing sets using “train\_test\_split”.
        + Allocating 20% of the data for testing.
        + Ensuring the class distribution is preserved in both training and testing sets. Crucial for imbalanced datasets like breast cancer data.
        + Randomly shuffling the data before splitting.
        + Setting a random seed for reproducibility i.e. ensures the same split if the code is run multiple times).
      * Printing the shapes (number of rows and columns) of the training and testing sets.

**4. Data Preprocessing (Normalization):**

* + - * Importing the “StandardScaler” class from scikit-learn preprocessing module.
      * Creating a standardizer object “sc” to perform standard scaling.
      * Fitting the standardizer on the training data (X\_train) calculating the mean and standard deviation of each feature.
      * Transforms the testing data (X\_test) using the parameters learned from the fitted standardizer. This ensures both training and testing data have the same scaling, improving model performance.

**5. Model Implementation (Naive Bayes)**

* + - * Importing the Gaussian Naive Bayes classifier from scikit-learn.
      * Creating a Gaussian Naive Bayes classifier object named “clf”.
      * Training the classifier on the training data (X\_train and y\_train). This involves learning the model parameters based on the features and corresponding diagnoses.

**6. Prediction:**

* + - * Using the trained classifier (clf) to predict the diagnosis labels for the testing data (X\_test). The predicted labels are stored in y\_pred.

**7. Evaluation (Confusion Matrix):**

* + - * Importing the confusion\_matrix function from scikit-learn metrics module.
      * Creating a confusion matrix cm that compares the actual diagnoses (y\_test) with the predicted diagnoses (y\_pred).
        + Rows represent the actual diagnoses.
        + Columns represent the predicted diagnoses.
        + Each cell (i, j) shows the number of samples where the actual diagnosis was class “I” but the model predicted is class “j”.
      * The confusion matrix provides insights into the model's performance, such as the number of correct and incorrect predictions for each class.
      * Printing the confusion matrix allows you to visualize these.
    - Breast Cancer Prediction with Decision Tree Classifier:

**1. Library Imports:**

* + - * Imported the NumPy library for numerical computations.
      * Imported the Pandas library for data manipulation and analysis.
      * Imported the Matplotlib library for creating visualizations
      * Imports the Seaborn library for statistical data visualization

**2. Data Loading:**

* Reading the breast cancer data from a CSV file named "Breast\_cancer\_data.csv" and stores it in a Pandas DataFrame named “df”.

**3. Data Splitting:**

* + - * Separating features (X) from the target variable (y).
      * Spliting the data into training and testing sets (X\_train, X\_test, y\_train, y\_test) using “train\_test\_split” while ensuring that class distribution is preserved.
      * Maintaining randomness and setting a random seed) for reproducibility.
* Printing the shapes (number of rows and columns) of the training and testing sets.

**4. Data Preprocessing (Normalization):**

* Using StandardScaler (sc) to normalize the training data (X\_train).
* The testing data (X\_test) is then transformed using the parameters learned from the fitted standardizer (sc.transform(X\_test)) to ensure consistent scaling across both sets.

**5. Model Implementation (Decision Tree):**

* Importing the Decision Tree Classifier class from scikit-learn.
* Creating a Decision Tree Classifier object named “dt”. The “max\_depth” parameter controls the complexity of the tree, with higher values leading to more complex and potentially overfitting trees.
* Training the decision tree classifier on the training data (X\_train and y\_train) where the decision rules are based on the features and corresponding diagnoses.

**6. Prediction:**

* Using the trained decision tree classifier (dt) to predict the diagnosis labels for the testing data (X\_test). The predicted labels are stored in “y\_pred”.

**7. Evaluation (Confusion Matrix):**

* Confusion matrix (cm) shows the distribution of actual vs. predicted diagnoses.
* Interpreting the confusion matrix and the visualization helps identify potential biases or areas for improvement in the model.
* The performance of the model is showed then by calculating the Accuracy, Precision, Recall and F1-Score.

## Breast Cancer Prediction with K-Nearest Neighbors (KNN):

**1. Library Imports:**

* + - * Imported the NumPy library for numerical computations.
      * Imported the Pandas library for data manipulation and analysis.
      * Imported the Matplotlib library for creating visualizations
      * Imports the Seaborn library for statistical data visualization

**2. Data Loading:**

* Reading breast cancer data from a CSV file named "Breast\_cancer\_data.csv" and stores it in a Pandas DataFrame named “df”.

**3. Data Splitting:**

* Splitting the data into training and testing sets (X\_train, X\_test, y\_train, y\_test) using “train\_test\_split”. Here, “stratify=y” ensures the class distribution is preserved, and random\_state=21 sets a random seed for reproducibility.

**4. KNN Hyperparameter Tuning (Number of Neighbours):**

* Finding the optimal number of neighbours for the KNN model.
* Creating a list of neighbours from 1 to 30 (inclusive).
* A loop iterates through each value in neighbours:
  + Creating a KNN classifier object (knn) with the current number of neighbours’ (neighbor).
  + Fitting the KNN classifier on the training data.
  + Appending the training accuracy for the current number of neighbors to the “train\_accuracies” list.
* After the loop completes, we have the lists containing training and testing accuracies for different numbers of neighbors.

**5. Visualization of KNN Performance:**

* Creating a Matplotlib figure for plotting.
* Plotting the training accuracies vs. the number of neighbors.
* Plotting the testing accuracies vs. the number of neighbors.
* The plot visualizes how the model's performance (accuracy) changes with the number of neighbors considered for classification.

**6. KNN Model with StandardScaler:**

* Importing the “make\_pipeline” function from scikit-learn to create a pipeline.
* Creating a pipeline named “clf” that consists of two steps:
  + StandardScaler(): Standardizes the features in the training data.
  + KNeighborsClassifier(): KNN classifier with a chosen number of neighbors (here, 3 based on potential insights from the previous visualization).
* Fitting the pipeline (including both the scaler and the KNN classifier) on the training data.

**7. Evaluation and Prediction:**

* Evaluating the model's performance on the training data.
* Evaluating the model's performance on the testing data.
* Visualizing the confusion matrix for the model's predictions on the testing data.
* Generatiing predictions for the testing data using the trained pipeline (clf).
* The confusion matrix and predicted labels help assess the model's ability to correctly classify new data points.
* The performance of the model is showed then by calculating the Accuracy, Precision, Recall and F1-Score.

## Breast Cancer Prediction with Logistic Regression - Code Breakdown

**1. Library Imports:**

* + - * Imported the NumPy library for numerical computations.
      * Imported the Pandas library for data manipulation and analysis.
      * Imported the Matplotlib library for creating visualizations
      * Imports the Seaborn library for statistical data visualization

**2. Data Loading:**

* Reading breast cancer data from a CSV file named "Breast\_cancer\_data.csv" and storing it in a Pandas DataFrame named “df”.

**3. Data Splitting:**

* Splitting of data into training and testing sets (X\_train, X\_test, y\_train, y\_test) using “train\_test\_split”. Ensuring that the class distribution is preserved, and random\_state=21 sets a random seed for reproducibility.

**4. Data Preprocessing (Normalization):**

* Importing the “StandardScaler” class for data normalization.
* Creating a standardizer object (sc).
* Fitting the standardizer on the training data (calculating mean and standard deviation for each feature).
* Transforing the testing data using the parameters learned from the fitted standardizer. This ensures that there is consistent scaling across both sets, improving model performance for Logistic Regression.

**5. Model Implementation (Logistic Regression):**

* Importing the Logistic Regression class from scikit-learn.
* Createinhg a Logistic Regression classifier object (classifier) with a set random seed (random\_state=0) for reproducibility.
* Training the Logistic Regression classifier on the training data (X\_train and y\_train) which involves learning the model parameters based on the features and corresponding diagnoses.

**6. Prediction:**

* Generating predictions for the testing data using the trained Logistic Regression model (classifier). The predicted labels are stored in y\_pred.

**7. Evaluation (Confusion Matrix):**

* The confusion matrix (cm) and Confusion Matrix Display functions are used to evaluate the model's performance on the testing data.
* Analysing the confusion matrix helps identify potential biases or areas for improvement in the model.
* The performance of the model is showed then by calculating the Accuracy, Precision, Recall and F1-Score.
  + - Breast Cancer Prediction Using SVM (Support Vector Machine) Model:

**1. Importing Libraries:**

* + - * Imported the NumPy library for numerical computations.
      * Imported the Pandas library for data manipulation and analysis.
      * Imported the Matplotlib library for creating visualizations
      * Imports the Seaborn library for statistical data visualization

**2. Loading the Data:**

* Reading a CSV file named "Breast\_cancer\_data.csv" and saving it into a pandas DataFrame named “df”.

**3. Exploratory Data Analysis (EDA):**

* Showing information about the data, including data types and number of non-null entries in each column.
* Providing summary statistics for numerical columns (mean, standard deviation, etc.).
* Checking for missing values in each column.
* Checking for duplicate rows in the data.

**4. Feature Exploration:**

* Selecting columns containing numerical data.
* Selecting columns containing categorical data (objects).
* Calculating the correlation matrix between numerical features.
  + Visualizes the correlation matrix using a heatmap with annotations (sns.heatmap).
* Showing the number of unique values in each categorical column.
* Creating histograms of all features (numerical and categorical).

**5. Preprocessing Categorical Feature:**

* Converting the "diagnosis" column (likely containing text labels like "malignant" or "benign") into a numerical format suitable for the model.

**6. Splitting Data into Training and Testing Sets:**

* Importing the train-test split function.
* Splits the data into training and testing sets using “train\_test\_split”.
  + X: Features (all columns except "diagnosis").
  + y: Target variable ("diagnosis").
  + Selecting 20% of the data for the the testing set.
  + Ensuring the class distribution is similar in both sets (important for classification) and randomly shuffling the data before splitting.
  + Setting a seed for reproducibility (splitting will be the same every time you run the code).
* Printing the shapes of training and testing sets to confirm the split.

**7. Building and Training the Model:**

* Importing the pipeline creation function.
* Importing the standard scaler for normalization.
* Importing the Support Vector Machine classifier.
* Creates a pipeline using “make\_pipeline” :
  + StandardScaler() - Standardizes numerical features (often beneficial for SVM).
  + SVC() - The SVM classifier itself.
* Fitting the model (clf.fit()) on the training data (X\_train, y\_train).

**8. Model Evaluation:**

* Evaluates the model's performance on both training and testing data using clf.score(). This gives the accuracy (percentage of correct predictions). Printing the Accuracy scores for Training and Testing dataset.
* Creating a confusion matrix visualization using “ConfusionMatrixDisplay” to see how many data points were classified correctly or incorrectly (categorized by actual and predicted classes).

**9. Making Predictions:**

* Predicting class labels for the testing data using “clf.predict” and storing it.
* Also calculating other performance features like F1-score, Recall And Precision for providing a comprehensive evaluation.

## Breast Cancer Prediction with Random Forest - Code Breakdown

**1. Library Imports:**

* Importing necessary libraries for numerical computations (numpy), data manipulation (pandas), visualization (matplotlib.pyplot), model selection (sklearn.model\_selection), random number generation (datetime), and the Random Forest Classifier (RandomForestClassifier) from “sklearn.ensemble”. Also suppressing warnings using warnings.filterwarnings('ignore').

**2. Data Loading and Exploration:**

* Reading breast cancer data from a CSV file named "Breast\_cancer\_data.csv" into a Pandas DataFrame named “df”.
* Exploring the data and checking the data dimensions (number of rows and columns) , displaying the first few rows of the data. Also, providing summary statistics for numerical features.

**3. Feature and Target Variable Selection:**

* Selecting the first three columns as features (X) and the last column as the target variable (Y). This assumes that the first three columns are features and the last column contains the diagnosis labels.
* Printing the shapes of X and Y to confirm their dimensions.

**4. Data Splitting:**

* Splitting the data into training and testing sets using “train\_test\_split”. Here, 17.5% of the data is allocated for testing and a random seed ensures reproducibility.
* Printing the shapes of the training and testing splits to visualize the distribution.

**5. Baseline Dummy Classifier:**

* Createing a baseline model using DummyClassifier from sklearn.dummy. This classifier simply predicts the most frequent class (majority vote) and serves as a reference point for the Random Forest model's performance.
* Training the dummy classifier on the training data using “clf.fit”.
* Predicting the labels for the testing data.
* Evaluating the baseline model using the confusion matrix and accuracy score. The confusion matrix shows how many predictions fall into each category (correct/incorrect) and accuracy indicates the proportion of correct predictions.

**6. Random Forest Classifier:**

* Creating a Random Forest Classifier object with 100 decision trees and a set random seed (random\_state=0) for reproducibility.
* Training the Random Forest model on the training data (
* Predicting labels for the testing data.
* Evaluating the Random Forest model using various performance metrics like Accuracy, Recall, F1 Score.

**7. Model Persistence:**

* Saving the trained Random Forest model using Python's pickle library (pickle.dump) to a file named "Breast\_Cancer.sav". This allows you to load the model later for predictions without retraining.

I'd be glad to provide a detailed description of the code, incorporating the strengths of previous responses and addressing potential shortcomings:

**App Breast-cancer:**

**1. Library Imports:**

* Importing NumPy library which provides numerical computation capabilities for array manipulation and mathematical operations.
* Importing Pandas library which enables data manipulation through DataFrames (similar to spreadsheets) for efficient data organization and analysis.
* Importing Flask library which creates the foundation for the web application, handling routing, request processing, and response generation.
* Importing Request from Flask library which facilitates access to user input submitted through forms on the web interface.
* Importing Render\_template from Flask library which renders HTML templates, dynamically inserting content based on variables passed from the Python code.
* Importing Pickle library for loading a pre-trained machine learning model saved in a pickle file format.

**2. Model Loading:**

* Loading the pre-trained Random Forest model named "Breast\_Cancer.sav" using pickle.

**3. Flask App Initialization:**

* Creating a Flask application instance named “app” which serves as the core of the web app, managing routes, handling requests, and coordinating responses.

**4. Webpage Routes:**

The code defines two routes (functions) that handle user interaction within the web app:

* @app.route('/'):
  + Decorator that maps the root path (/) of the web app to the home function.
  + home function: Likely renders an HTML template named "index1.html" using render\_template. This template would be the initial webpage that the user sees when they visit the web app.
* @app.route('/predict', methods=['POST']):
  + Decorator that maps the /predict path to the predict function, handling HTTP POST requests (typically used for form submissions).
  + predict function: Performs the core prediction task:
    - Extracts user input values from the submitted form using request.form.values().
    - Converts form values (strings) to float numbers using a list comprehension, assuming the form fields correspond to numerical features.
    - Creates a NumPy array named final\_features containing the user-provided features in the expected order.
    - Constructs a Pandas DataFrame named df with the features and their corresponding names (assumed to be in a specific order: "mean\_radius", "mean\_texture", "mean\_perimeter"). These feature names are likely defined elsewhere in your code (not shown here).
    - Uses the loaded model (model.predict) to generate a prediction on the df DataFrame (containing the user's features).
    - Based on the prediction (0 or 1), assigns a descriptive text to the variable res\_val indicating "breast cancer" or "no breast cancer".
    - Renders the "index1.html" template again, but this time with the predicted result (prediction\_text) passed as a variable. This variable will likely be displayed on the webpage to show the user the model's prediction.

**5. Running the Web App:**

* if \_\_name\_\_ == "\_\_main\_\_"::
  + Ensures the code within this block executes only when the script is run directly (not imported as a module).
* app.run(debug=True):
  + Starts the Flask development server. Setting debug=True enables automatic code reloading during development, allowing changes to be reflected without restarting the server. You'll likely want to set this to False for a deployed web application.

**Overall Functionality:**

This app implements a basic Flask web application that allows users to interact with a pre-trained Random Forest model for breast cancer prediction. Users can access the web app through a web browser and submit their features through a form. The application then processes the input, makes a prediction using the loaded model, and displays the result on the webpage.

**Non-Functional Requirements:**

**Usability:**

* **User Interface (UI):** The web application is a user-friendly and intuitive interface that is easy for users with varying levels of technical expertise to navigate.
* **Input Validation:** The application validates user input to ensure it's in the correct format and within expected ranges. This helps prevent errors and improves data quality.

**Reliability:**

* **Availability:** The web application is available to the users for use most of the time, with minimal downtime for maintenance or upgrades.
* **Accuracy**
* **Error Handling:** The application handles unexpected errors and provide informative messages to users. **Performance:**
* **Response Time:** The application responses to user actions (e.g., form submissions) within a reasonable time.
* **Scalability:** The application is be able to handle a large number of users and data volume without performance degradation.
* **Resource Utilization:** The application efficiently utilize system resources (CPU, memory) to avoid impacting other applications or the overall system performance.

**Security:**

* **Data Security:** User data (both input features and potentially model outputs) is treated with confidentiality and protected from unauthorized access or modification by implementing encryption techniques.
* **Authentication and Authorization:** The application is having mechanisms for user authentication and authorization to restrict access to sensitive data or functionalities.
* **Error Handling:** Security-related errors or vulnerabilities is handled appropriately, with logging and potential security measures like input sanitization to prevent attacks.

**Maintainability:**

* **Modular Design:** The application code is and well-organized, using clear naming conventions and proper code comments which facilitates code understanding, maintenance, and future modifications.
* **Logging:** Implement logging mechanisms to track application activity, errors, and user interactions for debugging, troubleshooting, and performance analysis.