Breast Cancer Detection using Machine Learning

by

Chellaboina Venkata Ishwarya

Eslavath Naresh

This project is submitted to the Gannon University graduate faculty in

partial fulfillment for the degree Master of Science in Computer and Information Science.

Major: M.S. in Computer & Information Science

Approved:

|  |
| --- |
|  |
| *Dr. Sheheeda Manakkadu* |
| Advising Professor in Charge of Research |

Gannon University

Erie, Pennsylvania 16541

December 2023

# Acknowledgement

I am immensely grateful to numerous individuals who have contributed to the successful completion of this project. Foremost, I extend my heartfelt gratitude to my advisor. Dr. Sheheeda Manakkadu. Her role as an educator, mentor, and supervisor has been fundamental to my journey. Dr. Sheheeda Manakkadu invaluable advice, continuous encouragement, and profound insights have greatly influenced my work. Her unwavering support, patience, and extensive knowledge have been pivotal in my pursuit of a master’s degree. I am especially thankful for her availability during crucial times and her guidance, which was instrumental in the timely completion of this project. Learning under Dr.Sheheeda Manakkadu's tutelage has been a true honor.

Additionally, I extend my sincere thanks to the members of my project committee. Their encouragement, perceptive comments, and probing questions have not only helped me to sidestep potential errors but have also significantly enhanced my presentation skills. Working alongside them has enriched my research experience, deepening my understanding of the subject matter. I am proud and grateful for the opportunity to work and learn with such esteemed professionals.

## ABSTRACT

Breast cancer diagnosis plays a pivotal role in determining treatment strategies and patient outcomes. This study employs machine learning techniques, specifically Random Forest and Logistic Regression models, to predict the malignancy of breast tumors utilizing the

Wisconsin Diagnostic Breast Cancer (WDBC) dataset. The dataset encompasses diverse features extracted from fine-needle aspirate biopsies, covering critical aspects like texture, radius, and concavity.

The central goal of this research is to assess the effectiveness of Random Forest and Logistic Regression models in classifying breast tumors as malignant or benign based on the distinctive features present in the WDBC dataset. Feature selection methods are implemented to enhance model interpretability and performance. The study systematically explores the influence of various hyperparameter configurations on the models' ability to accurately predict malignancy, considering metrics such as sensitivity, specificity, and overall classification accuracy.

Preprocessing steps involve meticulous handling of missing values, standardization of features, and addressing potential dataset biases. Cross-validation techniques are applied to gauge the models' generalization capabilities, and the findings are validated on an independent test set. The ultimate objective is to provide actionable insights into the predictive capabilities of Random Forest and Logistic Regression models in determining breast tumor malignancy, offering valuable guidance to healthcare professionals for informed decision-making in breast cancer diagnosis.

Contents

[Acknowledgement 2](#_Toc153571826)

[ABSTRACT 3](#_Toc153571827)

[1. Introduction 7](#_Toc153571828)

[1.1 Problem Definition 7](#_Toc153571829)

[1.2 Current Challenges 7](#_Toc153571830)

[1.2.1 Diagnostic Accuracy 7](#_Toc153571831)

[1.2.2 Resource Intensity 7](#_Toc153571832)

[1.3 Objectives 8](#_Toc153571833)

[1.3.1 Enhanced Accuracy 8](#_Toc153571834)

[1.3.2 Non-Invasive Diagnosis 8](#_Toc153571835)

[1.3.3 Accessibility 8](#_Toc153571836)

[1.4 Significance 8](#_Toc153571837)

[1.5 Success Criteria 8](#_Toc153571838)

[1.5.1 Accuracy Metric 8](#_Toc153571839)

[1.5.2 Precision, Recall, and F1 Score 8](#_Toc153571840)

[1.5.3 Interpretability 9](#_Toc153571841)

[1.5.4 Generalization 9](#_Toc153571842)

[1.5.5 Scalability and Resource Efficiency 9](#_Toc153571843)

[1.5.6 Ethical Considerations 9](#_Toc153571844)

[1.5.7 Deployment Feasibility 9](#_Toc153571845)

[1.6 Project Objectives 10](#_Toc153571846)

[1.7 Project Planning 10](#_Toc153571847)

[1.7.1 Weekly Milestones 11](#_Toc153571848)

[2. Background Information and Related Works 12](#_Toc153571849)

[2.1 Related Work 12](#_Toc153571850)

[3. Data Collection and Preprocessing 13](#_Toc153571851)

[3.1 Cleaning 13](#_Toc153571852)

[Figure 3.1.1: Meta Data 13](#_Toc153571853)

[3.1.2 Descriptive Statistics 13](#_Toc153571854)

[3.1.3 Class Distribution 14](#_Toc153571855)

[3.1.4 Data Splitting 14](#_Toc153571856)

[3.1.5 Dataset Dimensions 14](#_Toc153571857)

[4. Methodology 14](#_Toc153571858)

[4.1 Logistic Regression and Random Forest Algorithms 14](#_Toc153571859)

[4.1.1 Random Forest 15](#_Toc153571860)

[Figure 4.1.2: Linear Regression 16](#_Toc153571861)

[Figure 4.1.3: Random Forest 16](#_Toc153571862)

[4.2 Libraries and Modules 16](#_Toc153571863)

[4.2.1 NumPy (`import numpy as np`): 16](#_Toc153571864)

[4.2.2 Pandas (`import pandas as pd`): 17](#_Toc153571865)

[4.2.3 scikit-learn (`import sklearn.datasets`): 17](#_Toc153571866)

[4.2.4 train\_test\_split` from scikit-learn (`from sklearn.model\_selection import train\_test\_split`): 17](#_Toc153571867)

[4.2.5 Logistic Regression from scikit-learn (`from sklearn.linear\_model import LogisticRegression`): 17](#_Toc153571868)

[4.2.6 accuracy\_score` from scikit-learn (`from sklearn.metrics import accuracy\_score`): 17](#_Toc153571869)

[4.3 Implementation 18](#_Toc153571870)

[4.3.1 Data Frame Structuring 18](#_Toc153571871)

[4.3.2 Encoding the Target Variables 18](#_Toc153571872)

[Figure 4.3.2: Encode 19](#_Toc153571873)

[4.3.3 Distribution of target classes 19](#_Toc153571874)

[Figure 4.3.4: Plot 20](#_Toc153571875)

[4.3.5 Correlation Heat Map 20](#_Toc153571876)

[Figure 4.3.6: Heatmap 21](#_Toc153571877)

[4.3.7 Feature Correlation with target variables 22](#_Toc153571878)

[4.3.8: Bar plot code 22](#_Toc153571879)

[Figure 4.3.9: Bar plot 23](#_Toc153571880)

[4.3.10 Scatter Plot: Malignant vs benign tumor 23](#_Toc153571881)

[Figure 4.3.11: Scatter Plot 24](file:////Users/akash/Downloads/report3_15-12%20(1).docx#_Toc153571882)

[4.3.12 Separating features and target variables 25](#_Toc153571883)

[4.3.13 Data Splitting 25](#_Toc153571884)

[4.3.14 Model Training 26](#_Toc153571885)

[4.3.15 Model Training for Logistic regression & Random forest 27](#_Toc153571886)

[4.3.16 Training KNN & SVM 28](#_Toc153571887)

[28](#_Toc153571888)

[4.3.17 Model Training for Decision tree &Naïve Bayes 29](#_Toc153571889)

[4.4Model Performance 30](#_Toc153571890)

[4.4.1Logistic Regression 30](#_Toc153571891)

[Figure 4.4.2: Model Evaluation 30](#_Toc153571892)

[4.4.3 Model Evaluation on Test Data 31](#_Toc153571893)

[Figure 4.4.4: Prediction using logistic regression. 31](#_Toc153571894)

[4.4.5 Random Forest 32](#_Toc153571895)

[Figure 4.4.6: Prediction using Random Forest 32](#_Toc153571896)

[4.4.7 KNN Classifier 33](#_Toc153571897)

[4.4.8 Prediction using KNN. 33](#_Toc153571898)

[4.4.9 SVM 34](#_Toc153571899)

[4.4.10 Figure: Prediction using SVM. 34](#_Toc153571900)

[4.4.11 Decision Tree 35](#_Toc153571901)

[4.4.12 Figure: Prediction using Decision Tree 35](#_Toc153571902)

[4.4.13 Naïve Bayes Model 36](#_Toc153571903)

[4.4.14 Figure: Prediction using Naïve Bayes Model 36](#_Toc153571904)

[Figure 4.4.15: Average Performance for 100 iterations 37](#_Toc153571905)

[4.4.16 Details of the 100 iterations graph representation explained. 37](#_Toc153571906)

[4.4.17 Detailed Graph Explanation of Graphs 37](#_Toc153571907)

[4.4.18 Box Plot Performance comparison of all the machine learning algorithms used 39](#_Toc153571908)

[4.4.19 Model Serialization 43](#_Toc153571909)

[5. Results 43](#_Toc153571910)

[5.1 Significance of this project 43](#_Toc153571911)

[5.2 Project Demonstration and User Interface 43](#_Toc153571912)

[5.2.1 Flask Application Details 44](#_Toc153571913)

[Figure 5.2.2 : Flow Chart 45](#_Toc153571914)

[5.2.3 Deployment on Amazon Service 45](#_Toc153571915)

[5.2.4 Utilization of Machine Learning Models 45](#_Toc153571916)

[5.2.5 Implementation using Flask. 45](#_Toc153571917)

[Figure 5.2.6 : Login Page 46](#_Toc153571918)

[Figure 5.*2.7* Home page 46](file:////Users/akash/Downloads/report3_15-12%20(1).docx#_Toc153571919)

[Figure 5.*2.8* Detect Cancer Page 47](#_Toc153571920)

[Figure 5.2.9: Result page 47](#_Toc153571921)

[6.Conclusion 53](#_Toc153571922)

[7.Scope and Future Works 53](#_Toc153571923)

[8. References 54](#_Toc153571924)

# 1. Introduction

Breast cancer has become a major concern worldwide, affecting women's health and causing a significant burden. It is crucial to establish accurate diagnoses in a timely manner, allowing for the development of appropriate treatment plans and enhancing patient outcomes. Recent advancements in machine learning (ML) have shown remarkable potential in enhancing breast cancer diagnosis precision and efficiency. This research explores the implementation of influential ML algorithms using the Wisconsin Diagnostic Breast Cancer (WDBC) dataset.

The WDBC dataset is a comprehensive repository of features derived from fine-needle aspirate biopsies, encompassing intricate details such as texture, radius, and concavity. By utilizing algorithms this study aims to contribute to the evolving discourse on improving breast cancer diagnostics by conducting a comparative analysis of these algorithms.

This investigation assesses the predictive capabilities of six models in differentiating between malignant and benign breast tumors. By examining feature selection techniques and conducting a thorough examination of hyperparameter configurations, this research aims to shed light on the strengths and limitations of these models in the context of breast cancer classification. The findings of this study have the potential to expand the diagnostic toolkit available to healthcare professionals, offering valuable insights for informed decision-making and personalized patient care in the field of breast cancer.

In conclusion, this research endeavors to evaluate the predictive capabilities of models in discerning between malignant and benign breast tumors. Through an exploration of feature selection techniques and a systematic examination of hyperparameter configurations, this study seeks to illuminate the strengths and limitations of these models in the context of breast cancer classification. The findings of this study hold the potential to augment the diagnostic toolkit available to healthcare professionals, providing them with valuable insights for informed decision-making and personalized patient care in the realm of breast cancer.

### 1.1 Problem Definition

Breast cancer is a prevalent health concern worldwide, and early and accurate diagnosis plays a pivotal role in effective treatment and improved patient outcomes. Traditional diagnostic methods, while valuable, are subject to certain limitations, often relying on invasive procedures. This project addresses the critical need for an advanced and non-invasive diagnostic tool by leveraging machine learning techniques.

### 1.2 Current Challenges

### 1.2.1 Diagnostic Accuracy

Current diagnostic methods, including mammography and biopsies, may have limitations in terms of sensitivity and specificity. False positives and false negatives can occur, leading to unnecessary treatments or delays in identifying malignant tumors.

### 1.2.2 Resource Intensity

Traditional diagnostic methods can be resource-intensive, requiring specialized equipment and expertise. This can pose challenges, especially in regions with limited access to advanced healthcare facilities.

### 1.3 Objectives

The primary objective of this project is to develop a machine learning model capable of accurately classifying breast tumors as malignant or benign based on features derived from fine-needle aspirate biopsies. By achieving this, the project aims to address the following:

### 1.3.1 Enhanced Accuracy

Improve the accuracy of breast cancer diagnosis beyond current standards, reducing the likelihood of misclassifications and providing a more reliable diagnostic tool.

### 1.3.2 Non-Invasive Diagnosis

Contribute to the development of a non-invasive diagnostic method that can complement existing approaches, reducing the need for invasive procedures and their associated risks.

### 1.3.3 Accessibility

Develop a model that, once trained, can be deployed and utilized in various healthcare settings, including those with limited resources, thereby enhancing accessibility to accurate diagnostic tools.

### 1.4 Significance

The successful implementation of this project holds the potential to significantly impact breast cancer diagnosis by offering a reliable, non-invasive, and accessible tool. This can lead to earlier detection of malignant tumors, allowing for timely interventions and improved patient outcomes.

### 1.5 Success Criteria

The success of the project will be evaluated based on a set of comprehensive criteria that collectively assess the effectiveness, reliability, and practicality of the machine learning model. The success criteria are designed to address key aspects of the project's objectives and to ensure that the developed model meets the desired standards.

### 1.5.1 Accuracy Metric

Definition: Accuracy represents the proportion of correctly classified instances out of the total instances in the testing set.

Success Criterion: Achieve a high overall accuracy, indicating the model's ability to correctly predict both malignant and benign cases.

### 1.5.2 Precision, Recall, and F1 Score

**Precision:**

Definition: Precision measures the accuracy of positive predictions, indicating the ratio of correctly predicted malignant cases to the total predicted malignant cases.

Success Criterion: Attain a high precision score to minimize false positives and ensure reliable identification of malignant cases.

**Recall:**

Definition: Recall (Sensitivity or True Positive Rate) measures the ability of the model to capture all actual positive instances.

Success Criterion: Achieve a high recall score to minimize false negatives, ensuring that the model effectively identifies most malignant cases.

**F1 Score:**

Definition: The F1 score is the harmonic mean of precision and recall, providing a balanced measure of the model's performance.

Success Criterion: Attain a high F1 score, indicating a good balance between precision and recall.

### 1.5.3 Interpretability

Definition: Interpretability refers to the model's ability to provide understandable and transparent insights into the features influencing its predictions.

Success Criterion: Develop a model that offers meaningful interpretations of its predictions, enhancing its utility for healthcare practitioners.

### 1.5.4 Generalization

Definition: Generalization assesses the model's performance on new, unseen data not used during training.

Success Criterion: Demonstrate the model's ability to generalize well to unseen instances, ensuring its reliability in real-world applications.

### 1.5.5 Scalability and Resource Efficiency

Definition: Scalability evaluates the model's performance as the dataset size increases, and Resource Efficiency considers the model's computational requirements.

Success Criterion: Develop a model that scales well to larger datasets and operates efficiently, making it feasible for deployment in various healthcare settings.

### 1.5.6 Ethical Considerations

Definition: Ethical considerations encompass the adherence to privacy and fairness standards in handling health data.

Success Criterion: Ensure that the model development and deployment adhere to ethical guidelines, safeguarding patient privacy and promoting fairness in predictions.

### 1.5.7 Deployment Feasibility

Definition: Deployment feasibility assesses the practicality of implementing the model in real-world healthcare settings.

Success Criterion: Develop a model that is deployable, user-friendly, and suitable for integration into existing healthcare systems.

By achieving success across these criteria, the project aims to contribute to the advancement of breast cancer diagnosis through the development of a reliable, interpretable, and ethical machine learning model. These criteria collectively address the multifaceted nature of the project's objectives, ensuring a comprehensive evaluation of the model's performance.

### 

### 1.6 Project Objectives

**Develop and Train a Machine Learning Model:**

Build a machine learning model specifically tailored for breast cancer classification using the Wisconsin Diagnostic Breast Cancer dataset.

**Evaluate Model Performance Using Key Metrics:**

Employ a comprehensive set of evaluation metrics, including accuracy, precision, recall, and F1 score, to assess the model's effectiveness.

**Explore Hyperparameter Tuning:**

Investigate different hyperparameter configurations to optimize the Logistic Regression model's performance.

**Investigate Additional Feature Engineering Techniques:**

Explore and implement additional feature engineering methods to enhance the interpretability and predictive power of the model.

**Assess Ensemble Methods, Particularly Random Forest:**

Evaluate the performance of ensemble methods, with a specific focus on Random Forest, and compare it to the initial Logistic Regression model.

**Document Findings and Insights:**

Thoroughly document the entire process, including methodologies, results, and key insights gained during the exploration and experimentation phases.

Contribute to Breast Cancer Diagnosis Advancements:

Contribute to the broader field of breast cancer diagnosis by developing a reliable, interpretable, and ethical machine learning model.

These objectives articulate the specific goals that guide the breast cancer classification project, encompassing model development, optimization, and the broader impact on breast cancer diagnosis.

### 1.7 Project Planning

**Weeks 1-2: Project Kickoff and Dataset Exploration**

* Define project goals, objectives, and success criteria.
* Familiarize yourself with the Wisconsin Diagnostic Breast Cancer dataset.
* Begin exploratory data analysis (EDA) to understand the dataset's structure and characteristics.

**Weeks 3-4: Data Preprocessing**

* Handle missing values and outliers.
* Perform feature engineering to enhance the dataset.
* Explore descriptive statistics and visualizations to gain insights.

**Weeks 5-6: Initial Model Development**

* Choose various models as the initial classification algorithm.
* Split the dataset into training and testing sets.
* Train the models on the training set.

**Weeks 7-8: Model Evaluation and Hyperparameter Tuning**

* Evaluate the model using accuracy, precision, recall, and F1 score metrics.
* Explore hyperparameter tuning to optimize the Logistic Regression model.
* Document and compare model performance before and after tuning.

**Weeks 9-10: Feature Engineering and Documentation**

* Investigate additional feature engineering techniques.
* Assess the impact of feature engineering on model interpretability and performance.
* Begin documentation of the project, detailing methodologies, and early findings.

**Weeks 11-12: Ensemble Methods and Finalization**

* Explore the use of ensemble methods, specifically Random Forest.
* Compare the performance of Random Forest with the Logistic Regression model.
* Finalize the project documentation, including a comprehensive report summarizing the entire process, results, and insights.

**Continuous Tasks:**

* Regularly update documentation throughout each phase of the project.
* Regularly check and maintain code quality and consistency.
* Conduct regular team meetings to discuss progress, challenges, and next steps.

### 1.7.1 Weekly Milestones

Week 2: Completed initial EDA and dataset exploration.

Week 4: Concluded data preprocessing with clean and enhanced dataset.

Week 6: Trained and evaluated the initial models.

Week 8: Optimized the Logistic Regression model through hyperparameter tuning.

Week 10: Explored additional feature engineering techniques and documented findings.

Week 12: Completed the exploration of ensemble methods, finalized model selection, and concluded the project documentation.

This weekly breakdown provides a structured plan for the 12-week period, allowing for a phased and organized approach to the breast cancer classification project. Adjustments can be made based on the project's pace and evolving requirements.

## 2. Background Information and Related Works

Breast cancer remains a significant health issue worldwide. It is the most diagnosed cancer among women and ranks among the leading causes of cancer-related deaths. Accurate diagnosis is crucial for effective treatment and improved survival rates. Machine learning (ML) has emerged as a promising approach to enhance the diagnostic process in breast cancer.

The use of ML algorithms in breast cancer diagnosis has gained traction in recent years. Studies have focused on various aspects, including feature extraction techniques, model selection, and dataset considerations. These analyses aim to identify the most suitable ML techniques for accurate and timely breast cancer diagnoses.

This research specifically explores the use of models in breast cancer classification using the WDBC dataset. This comprehensive dataset has been widely employed in ML-based breast cancer diagnosis due to its comprehensive collection of features. By examining these models and their effectiveness in classifying breast cancer cases, this study aims to contribute to the growing body of knowledge on ML applications in breast cancer diagnosis.

The impact of this research extends beyond providing a valuable dataset for ML algorithms. It also emphasizes the importance of understanding and utilizing different feature extraction methods and datasets in achieving accurate breast cancer diagnoses. By applying advanced ML techniques, such as Random Forest ,Logistic Regression, Naive Bayes, Decision Tree, SVM, K Nearest Neighbor this study aims to improve the accuracy and efficiency of breast cancer diagnostics.

### 2.1 Related Work

Numerous studies have delved into the application of machine learning in breast cancer classification, showcasing the potential for improved diagnostic accuracy. Existing research often explores a variety of algorithms, feature engineering techniques, and datasets to optimize model performance. Key findings in related work include:

**Algorithmic Approaches**:

Logistic Regression, Support Vector Machines, Random Forest, and Neural Networks have been frequently explored for breast cancer classification due to their adaptability to binary classification tasks.

**Feature Importance:**

Feature engineering, including the selection and transformation of features, has proven influential in enhancing model interpretability and predictive power. Studies often highlight specific features that contribute significantly to accurate classification.

**Ensemble Methods:**

Ensemble methods, such as Random Forest, have demonstrated success in capturing complex patterns within breast cancer datasets. Their ability to mitigate overfitting and improve generalization is a focus of exploration.

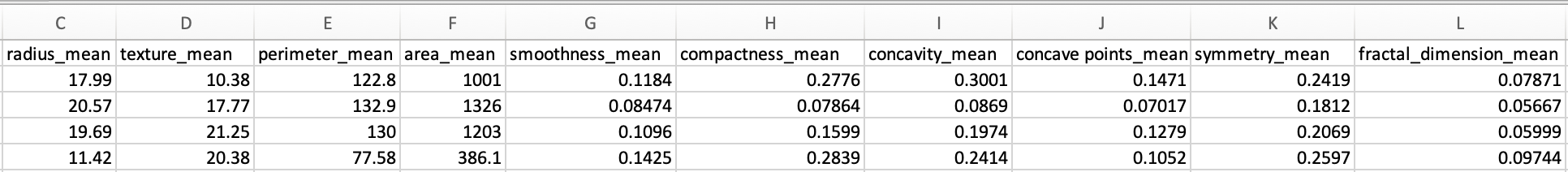
## 3. Data Collection and Preprocessing

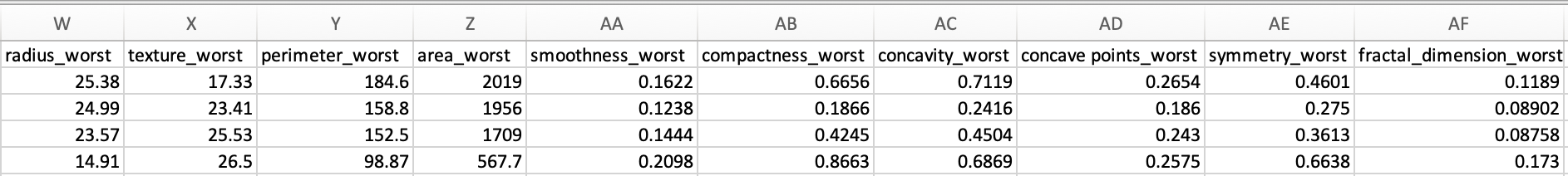
### 3.1 Cleaning

A thorough examination of the dataset was conducted to assess its quality, focusing on the presence of missing values. Utilizing the isnull().sum() method in Pandas, we observed that our dataset, represented by the data\_frame, exhibits a commendable data integrity with no missing values across any of its columns. This absence of null values ensures the robustness of our dataset, providing a solid foundation for subsequent data analysis and modeling processes.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

### Figure 3.1.1: Meta Data

### 3.1.2 Descriptive Statistics

A comprehensive exploration of the dataset was initiated through the use of the describe() method from the Pandas library. This method provides a detailed summary of the numerical features, encompassing key statistical measures such as mean, standard deviation, and quartiles. The resulting descriptive statistics offer valuable insights into the central tendency and variability of our dataset.

### 

### 3.1.3 Class Distribution

To understand the distribution of classes within our target variable, the value\_counts() method was applied to the 'label' column. This step revealed a clear overview of the frequency of each class, contributing essential information for handling potential class imbalances in subsequent modeling.

### 3.1.4 Data Splitting

In preparation for model training, the dataset was divided into features (X) and the target variable (Y). The features matrix, denoted as X, was derived by excluding the 'label' column, while Y represents the target variable itself. Following this, the dataset was further partitioned into training and testing sets using the train\_test\_split function. This pivotal step ensures a robust evaluation of the model's performance on unseen data.

### 3.1.5 Dataset Dimensions

To verify the integrity of the data splitting process, the shapes of the feature matrices (X, X\_train, and X\_test) were examined. The resulting dimensions were confirmed to align as expected, providing confidence in the correctness of the data preparation steps.

## 4. Methodology

### 4.1 Logistic Regression and Random Forest Algorithms

Objective: Logistic Regression is primarily used for binary classification problems, where the goal is to predict the probability that an instance belongs to a particular class (e.g., malignant or benign in the case of breast cancer).

Working Algorithm:

1. Sigmoid Function: Logistic Regression employs the sigmoid function (logistic function) to transform a linear combination of input features into a value between 0 and 1.
2. The formula for the sigmoid function is *σ*(*z*)=1/+*e*−*z*1, where z is the linear combination of input features.

Hypothesis Function:

1. The logistic regression hypothesis function is defined as hθ (x)=σ (θ T x), where θ represents the model parameters (weights) and x is the vector of input features.

Decision Boundary:

1. The decision boundary is set at σ (θ T x)=0.5. if *σ*(*θTx*) is greater than 0.5 the instance is precited to belong to class 1 otherwise it is predicted to belong to class O.

Training:

1. The model is trained by minimizing a cost function, often the logistic loss, through techniques like gradient descent. The goal is to find optimal parameter values (θ) that minimize the difference between predicted and actual outcomes.

### 4.1.1 Random Forest

Objective:

Random Forest is an ensemble learning method used for both classification and regression tasks. It constructs multiple decision trees during training and outputs the mode (classification) or average (regression) prediction of the individual trees.

Working Algorithm: bootstrapped Sampling:

1. Random forest builds multiple decision tress using bootstrapped samples from the original dataset. Reach tree is trained on a different subset of the data.

Feature Randomization:

1. At each node of a decision tree a random subset of feature is considered for splitting this introduces diversity among the trees reducing the risk of overfitting.

Decision Trees:

1. Each decision tree is constructed recursively by choosing the best split at each node the split is determined based on a criterion such as gini impurity for classification tasks.

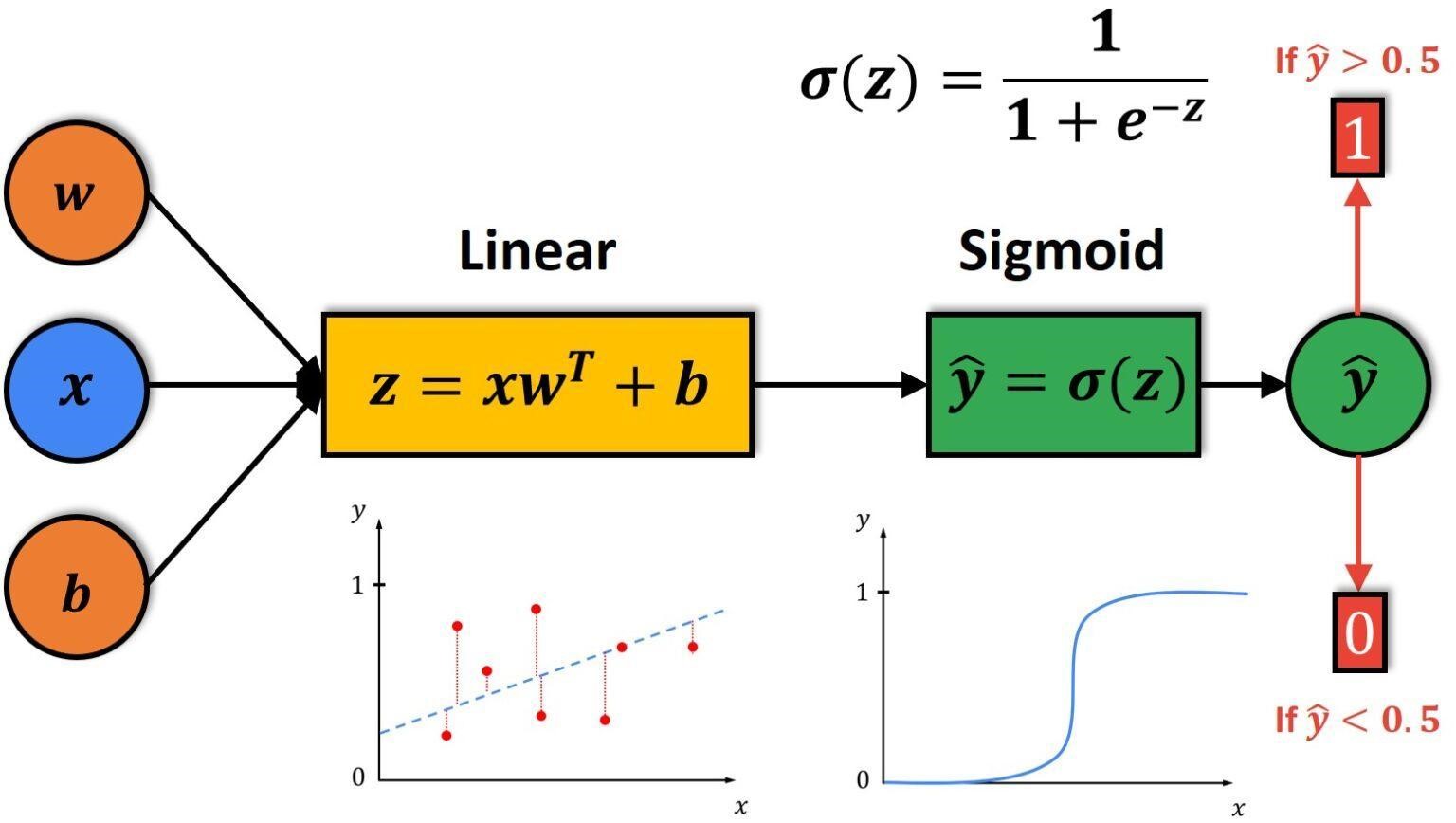
Voting (Classification) or Averaging (Regression):

1. For classification tasks, the final prediction is determined by a majority vote among the individual trees. For regression tasks, the final prediction is the average of the individual tree predictions.

Advantages:

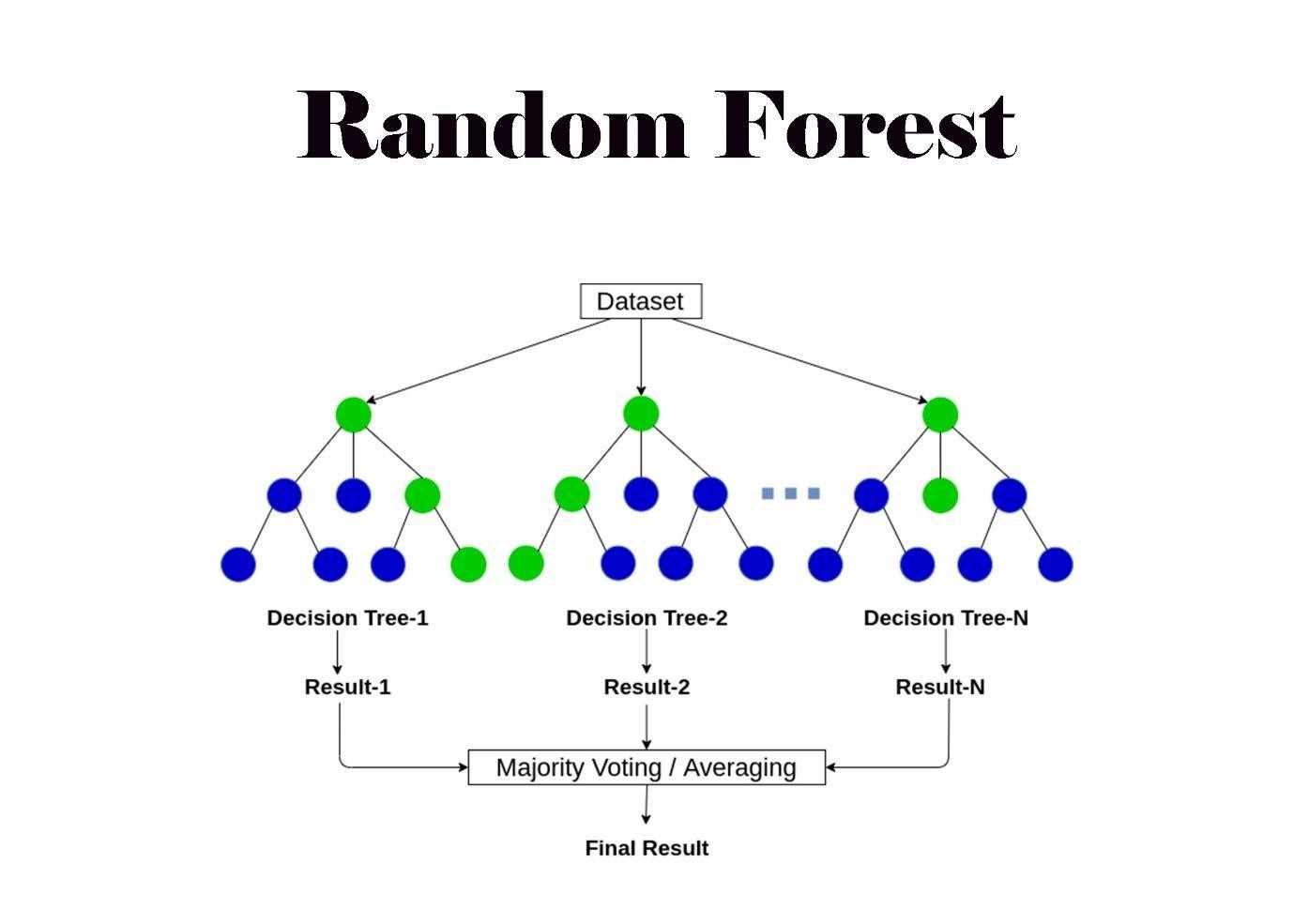
1. Random Forest is known for its ability to handle complex relationships in data, reduce overfitting, and provide robust predictions. The ensemble of diverse trees often results in a more accurate and stable model.

Both Logistic Regression and Random Forest are powerful tools in machine learning, each with its strengths and suitability for different types of problems. Logistic Regression is interpretable and well-suited for linearly separable problems, while Random Forest excels in capturing non-linear relationships and handling high-dimensional data.



### Figure 4.1.2: Linear Regression

Source: https://datahacker.rs/004-machine-learning-logistic-regression-model/



### Figure 4.1.3: Random Forest

Source: https://www.analyticsvidhya.com/blog/2021/05/bagging-25-questions-to-test-your-skills-on-random-forest-algorithm/

### 4.2 Libraries and Modules

The foundation of any data science or machine learning project lies in the strategic selection and implementation of libraries and modules. These powerful tools provide essential functions and structures that streamline the development process. In this project, we leverage the following key libraries and modules:

### 4.2.1 NumPy (`import numpy as np`):

NumPy serves as the cornerstone for numerical computing in Python. Its versatile array operations and mathematical functions facilitate efficient data manipulation and computation.

import numpy as np

import pandas as pd

import sklearn.datasets

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import roc\_auc\_score

data\_frame = pd.read\_csv('/content/dataset.csv')

### 4.2.2 Pandas (`import pandas as pd`):

Pandas is instrumental in handling structured data. The Data Frame structure simplifies data organization and manipulation, making it an invaluable asset in our data preprocessing and analysis.

### 4.2.3 scikit-learn (`import sklearn.datasets`):

The scikit-learn library provides a rich collection of machine learning tools. The `dataset’s module from scikit-learn aids in accessing standard datasets, including the Wisconsin Diagnostic Breast Cancer dataset pivotal to this project.

### 4.2.4 train\_test\_split` from scikit-learn (`from sklearn.model\_selection import train\_test\_split`):

The `train\_test\_split` function is a fundamental tool for partitioning our dataset into training and testing sets, a crucial step in evaluating model performance.

### 4.2.5 Logistic Regression from scikit-learn (`from sklearn.linear\_model import LogisticRegression`):

The `LogisticRegression` class from scikit-learn forms the bedrock of our classification approach. Leveraging logistic regression allows us to model and predict the likelihood of breast tumors being malignant or benign.

### 4.2.6 accuracy\_score` from scikit-learn (`from sklearn.metrics import accuracy\_score`):

The `accuracy\_score` function from scikit-learn's `metrics` module provides a reliable metric for assessing the accuracy of our classification model, comparing predicted labels with true labels.

## 4.3 Implementation

### 4.3.1 Data Frame Structuring

This line of code calls the describe method on the DataFrame (data\_frame). The describe method provides descriptive statistics of the DataFrame, including measures of central tendency, dispersion, and shape of the distribution of a dataset's values. It gives you a quick overview of key statistics for each column in the DataFrame, such as count, mean, standard deviation, minimum, and maximum.

This line of code counts the number of occurrences where the 'diagnosis' column in the DataFrame (data\_frame) is equal to 'M' (indicating malignant). Here's a breakdown:

data\_frame.diagnosis == 'M': This creates a boolean Series where each element is True if the corresponding element in the 'diagnosis' column is 'M' and False otherwise.

.sum(): This sums up the True values in the boolean Series, effectively counting the number of occurrences where the diagnosis is 'M'. these lines of code provide a statistical summary of the DataFrame and count the occurrences of malignant cases in the 'diagnosis' column.

data\_frame.describe()

(data\_frame.diagnosis=='M').sum()

# (data\_frame.diagnosis==1).sum()

Output: 212

data\_frame.columns

output

Index(['id', 'diagnosis', 'radius\_mean', 'texture\_mean', 'perimeter\_mean',

'area\_mean', 'smoothness\_mean', 'compactness\_mean', 'concavity\_mean',

'concave points\_mean', 'symmetry\_mean', 'fractal\_dimension\_mean',

'radius\_se', 'texture\_se', 'perimeter\_se', 'area\_se', 'smoothness\_se',

'compactness\_se', 'concavity\_se', 'concave points\_se', 'symmetry\_se',

'fractal\_dimension\_se', 'radius\_worst', 'texture\_worst',

'perimeter\_worst', 'area\_worst', 'smoothness\_worst',

'compactness\_worst', 'concavity\_worst', 'concave points\_worst',

'symmetry\_worst', 'fractal\_dimension\_worst', 'Unnamed: 32'],

dtype='object')

### 4.3.2 Encoding the Target Variables

Replacing M by 0 and B by 1

M = data\_frame[data\_frame.diagnosis == "M"]

B = data\_frame[data\_frame.diagnosis == "B"]

data\_frame['diagnosis'] = data\_frame['diagnosis'].apply(lambda x: 0 if x == 'M' else 1)

1 --> Benign

0 --> Malignant

data\_frame.groupby('diagnosis').mean()

A screenshot of a computer

Description automatically generated

### Figure 4.3.2: Encode

In the process of preparing the dataset for machine learning, it is essential to encode categorical variables. The target variable, 'diagnosis,' initially consists of two categories: 'M' for malignant and 'B' for benign. To facilitate model training, a binary encoding is applied to transform the categorical labels into numerical values.

Explanation: Two subsets, M and B, are created based on the original 'diagnosis' values. These subsets represent instances corresponding to malignant ('M') and benign ('B') tumors, respectively. The 'diagnosis' column is then updated using the apply method and a lambda function. This function maps 'M' to 0 (indicating malignant) and 'B' to 1 (indicating benign). The result is a transformed 'diagnosis' column with numerical labels for subsequent machine learning model training.

The target variable, 'diagnosis,' is now encoded with numerical values (0 for malignant, 1 for benign), making it suitable for consumption by machine learning algorithms. This preprocessing step ensures compatibility with models that require numerical inputs and enhances the efficiency of the subsequent classification tasks.

### 4.3.3 Distribution of target classes

Exploring the distribution of the target variable is crucial to understanding the balance between different classes. The following code generates a count plot to visualize the distribution of malignant and benign cases in the dataset.

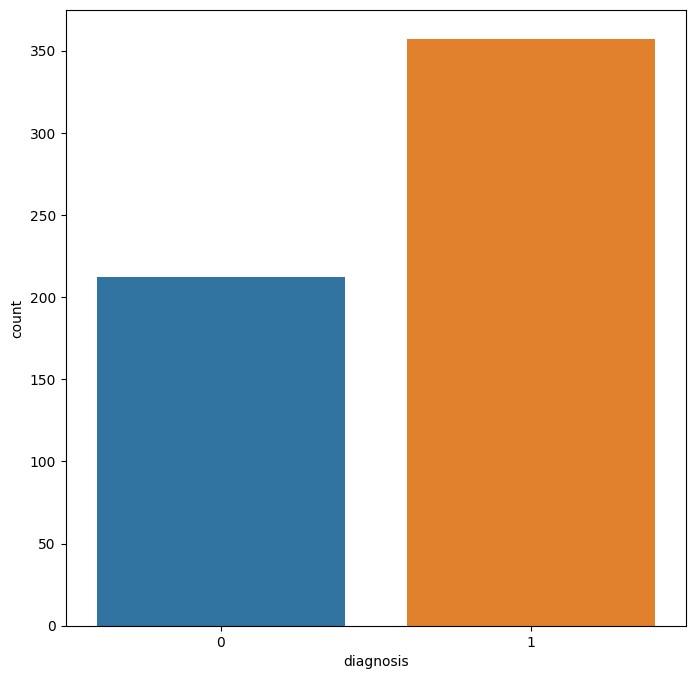
plt.figure(figsize=(8,8))

sns.countplot(data=data\_frame,x='diagnosis')

plt.show()

# 0 = M, 1 = B

Explanation: The countplot function from the Seaborn library is used to create a bar plot. The x-axis represents the target variable 'diagnosis,' while the y-axis displays the count of occurrences for each class (malignant and benign). The figure(figsize=(8,8)) ensures an appropriately sized visualization.



### Figure 4.3.4: Plot

The generated count plot visually represents the distribution of malignant and benign cases in the dataset. It provides an overview of the balance or imbalance between the two classes, which is essential information for understanding potential challenges in training machine learning models. This visualization serves as a preliminary exploration of the target variable's distribution, laying the groundwork for subsequent analyses and model training decisions.

### 4.3.5 Correlation Heat Map

Understanding the correlation between different features in the dataset is crucial for identifying potential relationships and patterns. The following code generates a correlation heatmap to visualize the pairwise correlation coefficients between features.

Explanation: The corr method calculates the correlation matrix for all numerical features in the dataset. The resulting matrix cor holds correlation coefficients between pairs of features. The heatmap function from Seaborn is then used to create a visually informative representation of these correlations.

Heatmap: The heatmap is a grid of color-coded squares, each corresponding to the correlation between two features. The color intensity and annotation values indicate the strength and direction of correlation (positive or negative). The figure(figsize=(25,25)) ensures an appropriately sized visualization.

Annotating Cells: The annot=True parameter adds numerical values to each cell in the heatmap, displaying the correlation coefficients.

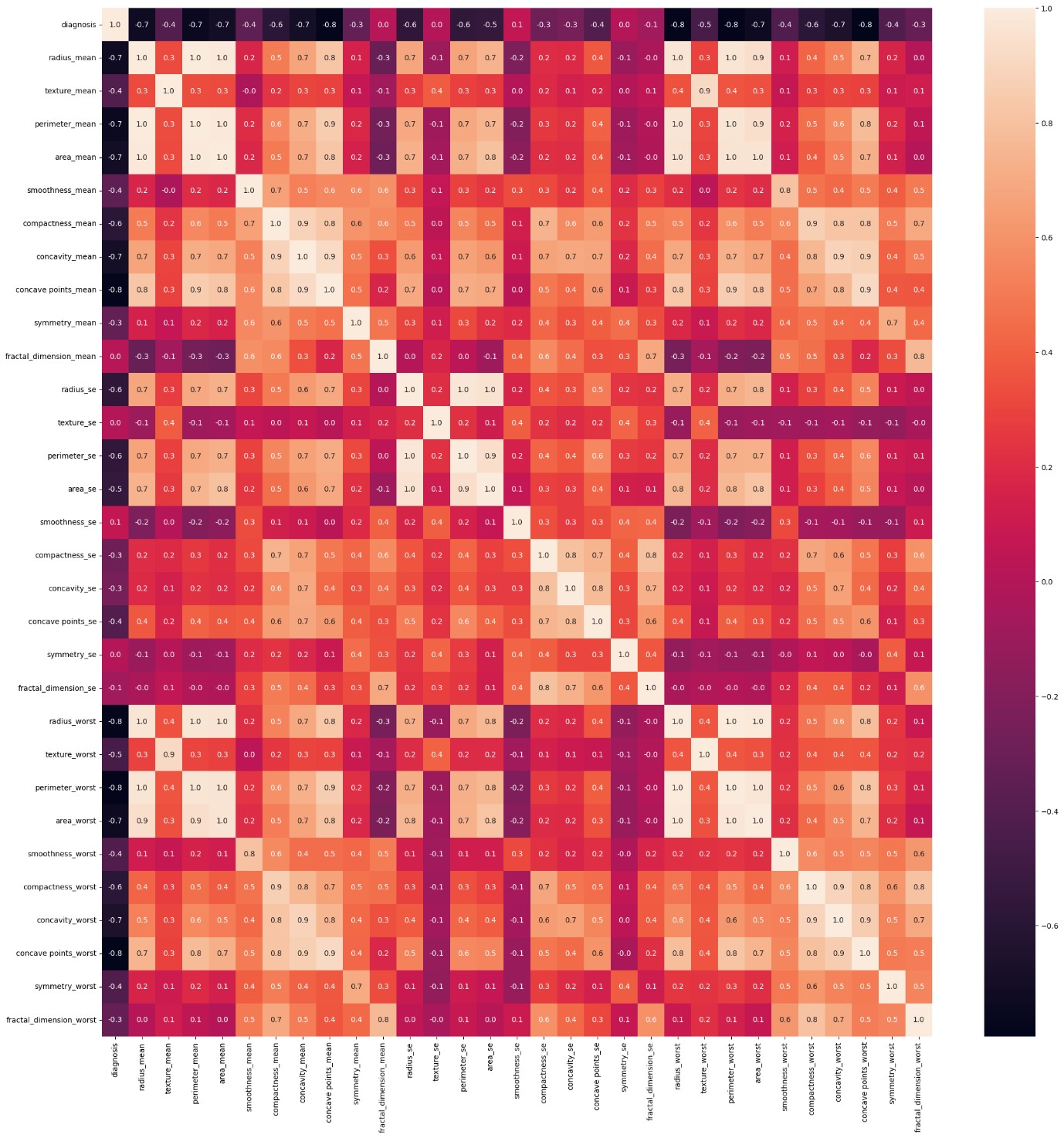
#heatmap of correlation between all columns

cor=data\_frame.corr()

plt.figure(figsize=(25,25))

sns.heatmap(cor, annot=True, fmt=".1f")

plt.show()



### Figure 4.3.6: Heatmap

The generated correlation heatmap provides an insightful visual representation of the pairwise correlations between features in the dataset. High correlation coefficients may suggest strong relationships between certain features, while low or negative correlations indicate weaker associations. This visualization is valuable for feature selection, identifying multicollinearity, and gaining a preliminary understanding of the interplay between different variables in the dataset. The annotation of correlation coefficients enhances the interpretability of the heatmap.

### 

### 4.3.7 Feature Correlation with target variables

Analyzing the correlation between individual features and the target variable is essential for identifying which features have the most influence on the classification task. The following code generates a bar plot to visualize the correlation coefficients between each feature and the target variable 'diagnosis.'

data\_frame\_2=data\_frame.drop(columns='diagnosis')

data\_frame\_2.corrwith(data\_frame['diagnosis']).plot.bar(

figsize=(20,10),title='correlated with diagnosis',rot=45,grid=True

)

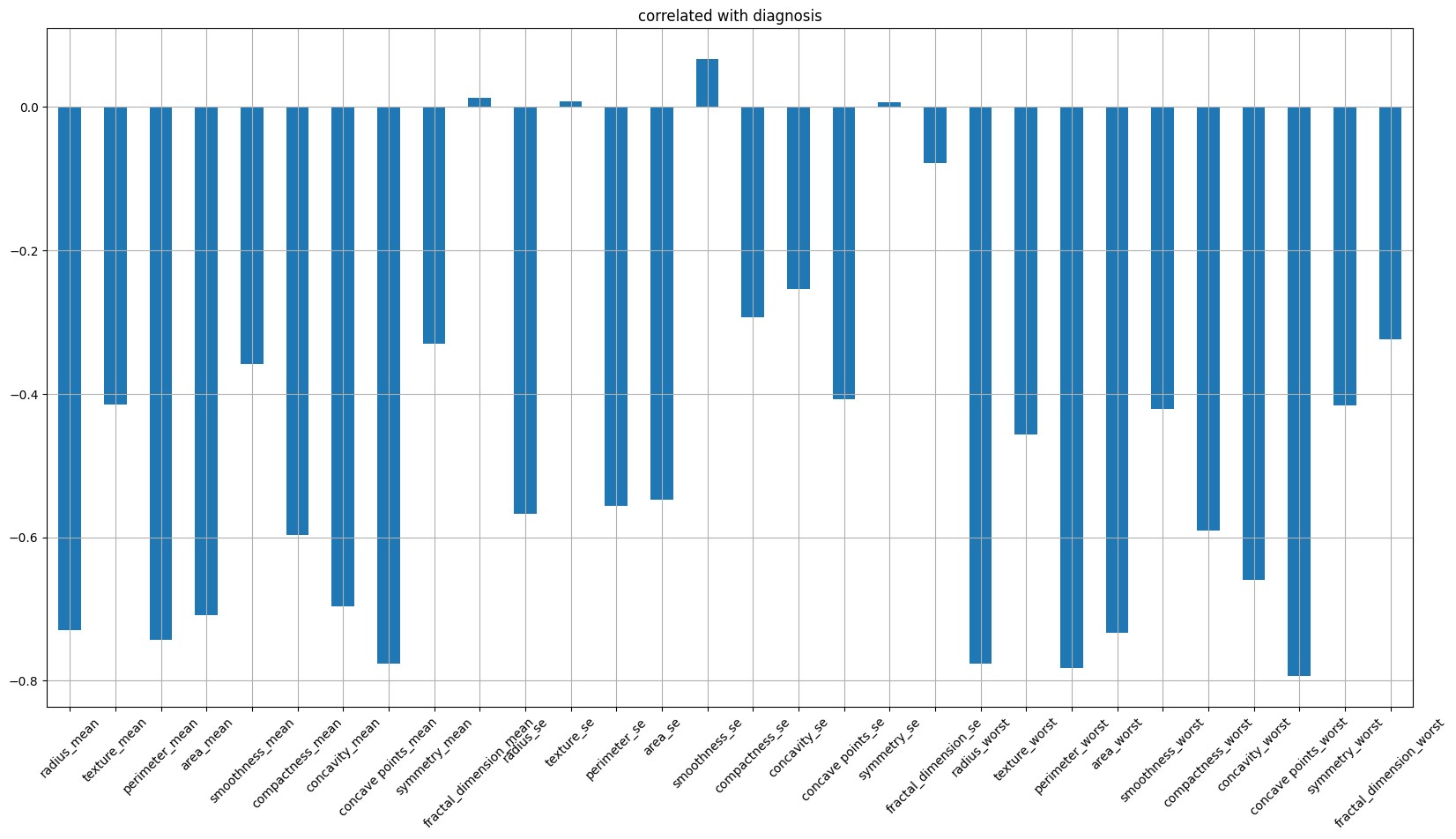
### 

### 4.3.8: Bar plot code

Explanation: The ‘corrwith’ method is used to calculate the correlation between each feature in data\_frame\_2 and the target variable 'diagnosis.' The resulting correlation coefficients are then plotted as a bar chart.

Bar Plot: Each bar in the plot represents the correlation coefficient between a feature and the target variable. The x-axis shows the feature names, and the y-axis displays the correlation values. Positive values indicate a positive correlation with 'diagnosis,' while negative values suggest a negative correlation.

Enhancements: The figure(fig size=(20,10)) sets the size of the plot, and rot=45 rotates the feature names for better readability. The title 'Correlated with Diagnosis' provides context, and grid lines assist in interpreting the correlation values.



### Figure 4.3.9: Bar plot

The generated bar plot visually communicates the correlation between each feature and the target variable 'diagnosis.' Features with high positive or negative correlations may play a significant role in predicting the target variable. This visualization aids in feature selection and understanding the importance of individual features in the breast cancer classification task.

### 4.3.10 Scatter Plot: Malignant vs benign tumor

Visualizing the distribution of data points based on two features, 'Radius Mean' and 'Texture Mean,' is instrumental in understanding the characteristics that distinguish malignant and benign tumors. The following code generates a scatter plot to compare these features for both types of tumors.

plt.title("Malignant vs Benign Tumor")

plt.xlabel("Radius Mean")

plt.ylabel("Texture Mean")

plt.scatter(M.radius\_mean, M.texture\_mean, color = "red", label = 0, alpha = 0.3)

plt.scatter(B.radius\_mean, B.texture\_mean, color = "lime", label = 1, alpha = 0.3)

plt.legend()

plt.show()

**Scatter Plot Code**

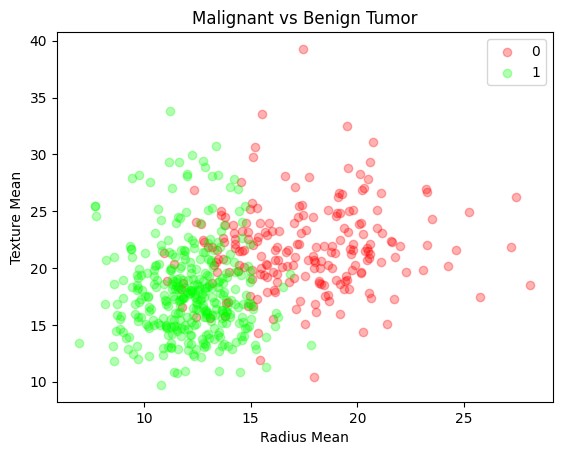
Explanation: This scatter plot compares the 'Radius Mean' and 'Texture Mean' features for malignant (label 0, represented by red points) and benign (label 1, represented by green points) tumors.

Title and Labels: The title "Malignant vs Benign Tumor" provides context for the plot, while the x-axis is labeled "Radius Mean," and the y-axis is labeled "Texture Mean."

Scatter Points: Each point on the plot represents a tumor, with its position determined by 'Radius Mean' on the x-axis and 'Texture Mean' on the y-axis. Malignant tumors are marked in red, and benign tumors are marked in green.

Alpha Parameter: The alpha=0.3 parameter controls the transparency of the points, making it easier to observe overlapping data points.

Legend: The legend indicates the color code for malignant and benign tumors.



### Figure 4.3.11: Scatter Plot

The scatter plot effectively illustrates the distribution of tumors in the feature space defined by 'Radius Mean' and 'Texture Mean.' This visualization allows for a qualitative understanding of how these two features may contribute to distinguishing between malignant and benign tumors. Patterns, clusters, or separations in the plot can provide valuable insights into the potential discriminatory power of these features for the classification task.

### 4.3.12 Separating features and target variables

Preparing the dataset for machine learning involves segregating the features and the target variable. The following code accomplishes this by creating two separate datasets, X and Y.

**Separating the features and target**

X = data\_frame.drop(columns = ['diagnosis'], axis=1)

Y = data\_frame['diagnosis']

Explanation:

Features (X): The X dataset contains all the columns from the original dataset (data\_frame) except for the 'diagnosis' column. It represents the feature matrix that will be used as input for the machine learning models.

Target Variable (Y): The Y dataset contains only the 'diagnosis' column from the original dataset. It represents the target variable that the machine learning models aim to predict.

This separation of features (X) and the target variable (Y) is a standard preprocessing step in machine learning. It allows for a clear distinction between the input features used for prediction and the variable to be predicted. The datasets X and Y are now ready for use in training and evaluating machine learning models for breast cancer classification.

### 4.3.13 Data Splitting

In machine learning, it is crucial to divide the dataset into training and testing sets to assess the model's generalization performance accurately. The following code employs the train\_test\_split function to split the feature matrix (X) and target variable (Y) into training and testing sets.

**Splitting the data into training data & Testing data**

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=2)

print(X.shape, X\_train.shape, X\_test.shape)

Output

(569, 30) (455, 30) (114, 30)

Explanation:

The train\_test\_split function from the scikit-learn library is used to randomly split the dataset into training and testing sets.

X represents the features, and Y is the target variable.

The test\_size=0.2 parameter specifies that 20% of the data will be reserved for testing, while the remaining 80% will be used for training.

The random\_state=2 parameter ensures reproducibility by setting a seed for the random number generator.

Result: X\_train and Y\_train represent the feature matrix and target variable for the training set.

X\_test and Y\_test represent the feature matrix and target variable for the testing set.

print(X.shape, X\_train.shape, X\_test.shape) prints the shapes of the original feature matrix and the training and testing sets for verification.

This data splitting process is critical for evaluating how well the trained machine learning model generalizes to unseen data. The resulting X\_train, X\_test, Y\_train, and Y\_test sets are now ready for use in training and evaluating machine learning models for breast cancer classification.

### 4.3.14 Model Training

The training phase involves fitting machine learning models to the training data, allowing them to learn patterns and relationships between features and the target variable. The following code initializes six models, which are as follows:Logistic Regression and Random Forest, and trains them on the provided training sets.

logisticRegressionModel = LogisticRegression()

randomForestModel = RandomForestClassifier()

knn\_model = KNeighbourClassifier()

svc\_model = SVC()

decision\_tree\_model = DecisionTreeClassifier()

naïve\_bayes\_model = GaussianNB()

# To store performance result of all the algorigthms

result = {

'logistic\_regression\_model': {},

'random\_forest\_model': {},

'knn\_model': {},

'svm\_model': {},

'decision\_tree\_model': {},

'naive\_bayes\_model': {},

}

### 4.3.15 Model Training for Logistic regression & Random Forest

**## Model Training using Logistic Regression Classifier ##**

accuracy\_list = []

precision\_list = []

recall\_list = []

f1\_list = []

for i in range(0,100):

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=i)

logisticRegressionModel.fit(X\_train, Y\_train)

y\_pred = logisticRegressionModel.predict(X\_test)

accuracy = accuracy\_score(Y\_test, y\_pred)\* 100

precision = precision\_score(Y\_test, y\_pred, average='weighted')\* 100

recall = recall\_score(Y\_test, y\_pred, average='weighted')\* 100

f1 = f1\_score(Y\_test, y\_pred, average='weighted')\* 100

accuracy\_list.append(round(accuracy, 2))

precision\_list.append(round(precision, 2))

recall\_list.append(round(recall, 2))

f1\_list.append(round(f1, 2))

result['logistic\_regression\_model'] = {

'accuracy\_list': accuracy\_list,

'precision\_list': precision\_list,

'recall\_list': recall\_list,

'f1\_list': f1\_list,

}

**## Model Training using Random Forest Classifier ##**

accuracy\_list = []

precision\_list = []

recall\_list = []

f1\_list = []

for i in range(0,100):

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=i)

randomForestModel.fit(X\_train, Y\_train)

y\_pred = randomForestModel.predict(X\_test)

accuracy = accuracy\_score(Y\_test, y\_pred) \* 100

precision = precision\_score(Y\_test, y\_pred, average='weighted') \* 100

recall = recall\_score(Y\_test, y\_pred, average='weighted') \* 100

f1 = f1\_score(Y\_test, y\_pred, average='weighted') \* 100

accuracy\_list.append(round(accuracy, 2))

precision\_list.append(round(precision, 2))

recall\_list.append(round(recall, 2))

f1\_list.append(round(f1, 2))

result['random\_forest\_model'] = {

'accuracy\_list': accuracy\_list,

'precision\_list': precision\_list,

'recall\_list': recall\_list,

'f1\_list': f1\_list,

}

### 4.3.16 Training KNN & SVM

### 

**## Model Training using K Nearest Neighbour Classifier ##**

accuracy\_list = []

precision\_list = []

recall\_list = []

f1\_list = []

for i in range(0,100):

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=i)

Knn\_model.fit(X\_train, Y\_train)

y\_pred = knn\_model.predict(X\_test)

accuracy = accuracy\_score(Y\_test, y\_pred)\* 100

precision = precision\_score(Y\_test, y\_pred, average='weighted')\* 100

recall = recall\_score(Y\_test, y\_pred, average='weighted')\* 100

f1 = f1\_score(Y\_test, y\_pred, average='weighted')\* 100

accuracy\_list.append(round(accuracy, 2))

precision\_list.append(round(precision, 2))

recall\_list.append(round(recall, 2))

f1\_list.append(round(f1, 2))

result['knn \_model'] = {

'accuracy\_list': accuracy\_list,

'precision\_list': precision\_list,

'recall\_list': recall\_list,

'f1\_list': f1\_list,

}

**## Model Training using SVM Classifier ##**

accuracy\_list = []

precision\_list = []

recall\_list = []

f1\_list = []

for i in range(0,100):

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=i)

Svm\_model.fit(X\_train, Y\_train)

y\_pred = svm\_model.predict(X\_test)

accuracy = accuracy\_score(Y\_test, y\_pred)\* 100

precision = precision\_score(Y\_test, y\_pred, average='weighted')\* 100

recall = recall\_score(Y\_test, y\_pred, average='weighted')\* 100

f1 = f1\_score(Y\_test, y\_pred, average='weighted')\* 100

accuracy\_list.append(round(accuracy, 2))

precision\_list.append(round(precision, 2))

recall\_list.append(round(recall, 2))

f1\_list.append(round(f1, 2))

result['svm\_model'] = {

'accuracy\_list': accuracy\_list,

'precision\_list': precision\_list,

'recall\_list': recall\_list,

'f1\_list': f1\_list,

}

### 4.3.17 Model Training for Decision tree &Naïve Bayes

**## Model Training using Decision Tree Classifier ##**

accuracy\_list = []

precision\_list = []

recall\_list = []

f1\_list = []

for i in range(0,100):

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=i)

decision\_tree\_model.fit(X\_train, Y\_train)

y\_pred = decision\_tree\_model.predict(X\_test)

accuracy = accuracy\_score(Y\_test, y\_pred)\* 100

precision = precision\_score(Y\_test, y\_pred, average='weighted')\* 100

recall = recall\_score(Y\_test, y\_pred, average='weighted')\* 100

f1 = f1\_score(Y\_test, y\_pred, average='weighted')\* 100

accuracy\_list.append(round(accuracy, 2))

precision\_list.append(round(precision, 2))

recall\_list.append(round(recall, 2))

f1\_list.append(round(f1, 2))

result['decision\_tree\_model'] = {

'accuracy\_list': accuracy\_list,

'precision\_list': precision\_list,

'recall\_list': recall\_list,

'f1\_list': f1\_list,

}

**## Model Training using Naïve Bayes Classifier ##**

accuracy\_list = []

precision\_list = []

recall\_list = []

f1\_list = []

for i in range(0,100):

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=i)

naive\_bayes\_model.fit(X\_train, Y\_train)

y\_pred = naive\_bayes\_model.predict(X\_test)

accuracy = accuracy\_score(Y\_test, y\_pred)\* 100

precision = precision\_score(Y\_test, y\_pred, average='weighted')\* 100

recall = recall\_score(Y\_test, y\_pred, average='weighted')\* 100

f1 = f1\_score(Y\_test, y\_pred, average='weighted')\* 100

accuracy\_list.append(round(accuracy, 2))

precision\_list.append(round(precision, 2))

recall\_list.append(round(recall, 2))

f1\_list.append(round(f1, 2))

result[naive\_bayes\_model'] = {

'accuracy\_list': accuracy\_list,

'precision\_list': precision\_list,

'recall\_list': recall\_list,

'f1\_list': f1\_list,

}

**Explanation:**

Six machine learning models, Logistic Regression, Random Forest, KNearestNeighbour, SVM, Decision tree, Naïve Bayes are initialized using the LogisticRegression(), RandomForestClassifier(), KNeighbourClassifier(), SVC(), DecisionTreeClassifier(), GaussianNB() constructors, respectively.

The fit method is then called on each model, providing the training feature matrix (X\_train) and the corresponding target variable (Y\_train). This process allows the models to learn the relationships between features and the target variable.

At this point, all models have been trained on the provided training data. They have learned to make predictions based on the patterns observed in the training set. The trained models are now ready for evaluation on the testing set to assess their performance in breast cancer classification.

## 4.4Model Performance

4.4.1Logistic Regression

y\_pred = logisticRegressionModel.predict(X\_test)

  accuracy = accuracy\_score(Y\_test, y\_pred)

  precision = precision\_score(Y\_test, y\_pred, average='weighted')

  recall = recall\_score(Y\_test, y\_pred, average='weighted')

  f1 = f1\_score(Y\_test, y\_pred n, average='weighted')

  accuracy\_list.append(accuracy)

  precision\_list.append(precision)

  recall\_list.append(recall)

  f1\_list.append(f1)

### Figure 4.4.2: Model Evaluation

##### performing a repeated train-test split and model evaluation process to assess the stability and variability of the machine learning model's performance. Here's a breakdown of the code and its purpose.

##### Lists Initialization:

##### accuracy\_list, precision\_list, recall\_list, and f1\_list are initialized as empty lists to store performance metrics for each iteration.

1. Loop Iterations (100 times):

* The code runs a loop 100 times, each time splitting the dataset into training and testing sets using train\_test\_split with a different random seed (random\_state=i).
* The logistic regression model (logisticRegressionModel) is trained on the training set (X\_train, Y\_train).
* Predictions are made on the test set (X\_test), and various performance metrics, including accuracy, precision, recall, and F1 score, are computed using scikit-learn functions (accuracy\_score, precision\_score, recall\_score, f1\_score).
* The computed metrics for each iteration are appended to the respective lists.

1. Result Analysis:

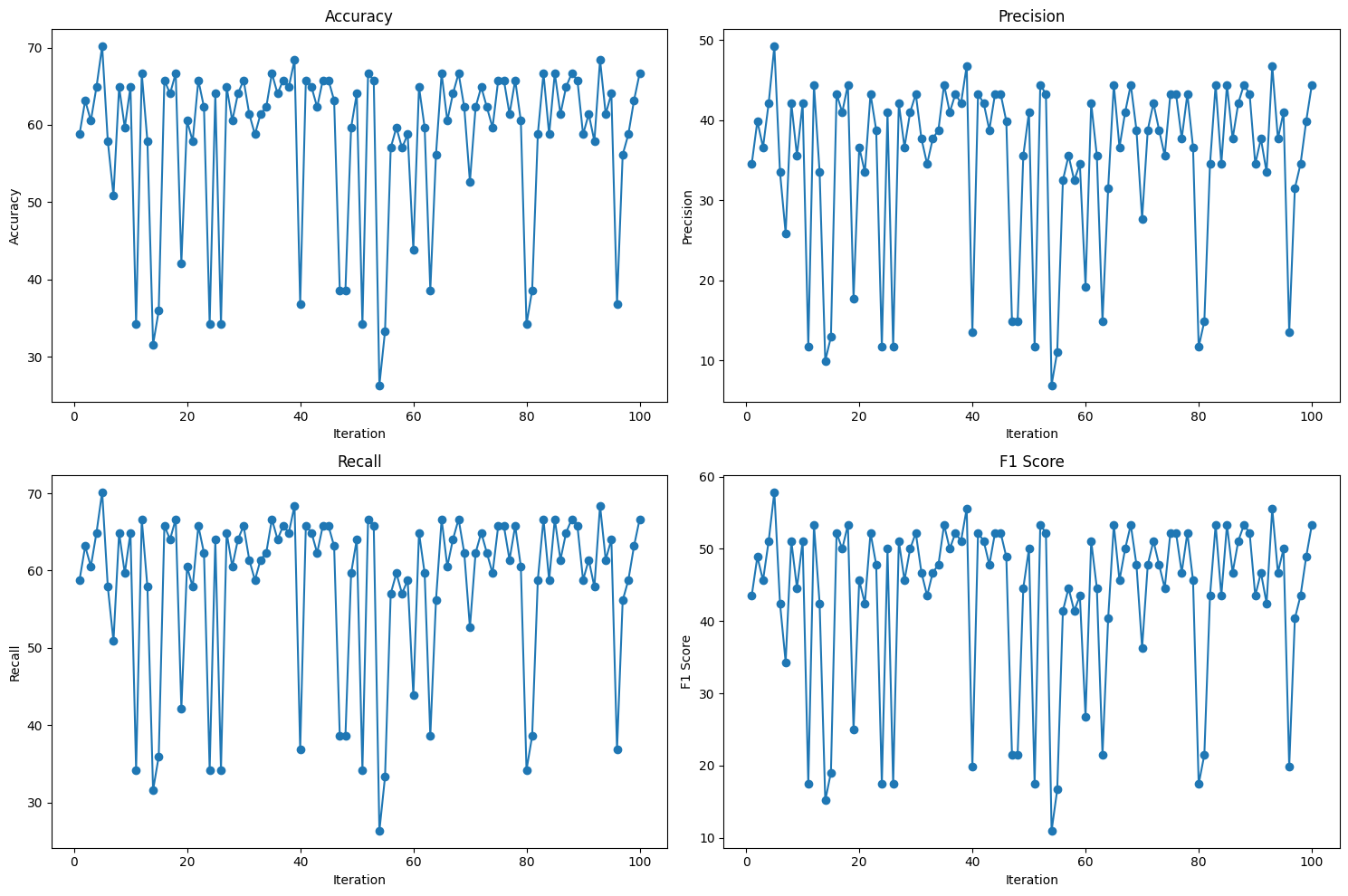
* After the loop completes, the lists (accuracy\_list, precision\_list, recall\_list, f1\_list) contain the performance metrics for each iteration.
* The lists can be analyzed for trends, variations, and consistency in the model's performance across different random splits of the dataset.

To assess the robustness and generalization capability of our machine learning model, we conducted a comprehensive analysis over 100 iterations. In each iteration, the dataset was randomly split into training and testing sets using the train\_test\_split function with different random seeds. The logistic regression model was then trained on the training set, and its performance was evaluated on the corresponding test set.

The resulting performance metrics, including accuracy, precision, recall, and F1 score, were recorded for each iteration. This iterative process allowed us to observe trends and variations in the model's performance across different data subsets. By analyzing these metrics, we gained valuable insights into the stability of our model and its ability to generalize to new and unseen data.

This thorough evaluation provides a comprehensive understanding of the model's behavior under various conditions, contributing to a more robust and reliable assessment of its predictive capabilities.

### 4.4.3 Model Evaluation on Test Data



### Figure 4.4.4: Prediction using logistic regression.

### 4.4.5 Random Forest

y\_pred = logisticRegressionModel.predict(X\_test)

  accuracy = accuracy\_score(Y\_test, y\_pred)

  precision = precision\_score(Y\_test, y\_pred, average='weighted')

  recall = recall\_score(Y\_test, y\_pred, average='weighted')

  f1 = f1\_score(Y\_test, y\_pred, average='weighted')

  accuracy\_list.appen(accuracy)

  precision\_list.append(precision)

  recall\_list.append(recall)

  f1\_list.append(f1)

Comprehensive evaluation of the Random Forest model's performance was conducted over 100 iterations. The dataset was randomly split into training and testing sets using the train\_test\_split function with different random seeds (random\_state=i). During each iteration, the Random Forest model was trained on the training set (randomForestModel.fit(X\_train, Y\_train)), and its performance was assessed on the corresponding test set.

Key performance metrics, including accuracy, precision, recall, and F1 score, were calculated for each iteration using the accuracy\_score, precision\_score, recall\_score, and f1\_score functions, respectively. These metrics provide insights into the model's ability to correctly classify instances, balance between precision and recall, and overall effectiveness.

The resulting lists (accuracy\_list, precision\_list, recall\_list, and f1\_list) contain the metric values for each iteration. This iterative analysis enables a comprehensive understanding of the Random Forest model's consistency and effectiveness across diverse subsets of the dataset, contributing to a more thorough evaluation of its predictive performance.

A graph of a graph

Description automatically generated with medium confidence

### Figure 4.4.6: Prediction using Random Forest

### 4.4.7 KNN Classifier

 y\_pred = knn\_model.predict(X\_test)

  accuracy = accuracy\_score(Y\_test, y\_pred) \* 100

  precision = precision\_score(Y\_test, y\_pred, average='weighted') \* 100

  recall = recall\_score(Y\_test, y\_pred, average='weighted') \* 100

  f1 = f1\_score(Y\_test, y\_pred, average='weighted') \* 100

  accuracy\_list.append(round(accuracy, 2))

  precision\_list.append(round(precision, 2))

  recall\_list.append(round(recall, 2))

  f1\_list.append(round(f1, 2))

### 

To assess the robustness and consistency of the k-Nearest Neighbors (KNN) model, an evaluation loop was implemented. For each iteration (100 in total), the dataset was randomly split into training and testing sets using the 'train\_test\_split' function. The KNN model was then trained on the training set and evaluated on the corresponding test set. Performance metrics, including accuracy, precision, recall, and F1 score, were calculated for each iteration, providing a comprehensive overview of the model's effectiveness across various data partitions. The results were recorded and stored in lists, capturing the variability in the model's performance under different conditions. This iterative evaluation approach ensures a thorough understanding of the KNN model's generalization capabilities and stability across diverse subsets of the dataset.

A graph of a graph

Description automatically generated with medium confidence

### 4.4.8 Prediction using KNN.

### 4.4.9 SVM

 y\_pred = svm\_model.predict(X\_test)

  accuracy = accuracy\_score(Y\_test, y\_pred) \* 100

  precision = precision\_score(Y\_test, y\_pred, average='weighted') \* 100

  recall = recall\_score(Y\_test, y\_pred, average='weighted') \* 100

  f1 = f1\_score(Y\_test, y\_pred, average='weighted') \* 100

  accuracy\_list.append(round(accuracy, 2))

  precision\_list.append(round(precision, 2))

  recall\_list.append(round(recall, 2))

  f1\_list.append(round(f1, 2))

The code is designed to visually represent the performance metrics, including accuracy, precision, recall, and F1 score, of a Support Vector Machine (SVM) model over 100 training iterations. After extracting these metrics from the result dictionary, the code utilizes Matplotlib to create a 2x2 grid of subplots. Each subplot corresponds to one performance metric and displays a line plot depicting how the metric evolves across the iterations. The x-axis represents the iteration number (from 1 to 100), while the y-axis represents the respective metric's value. Titles and labels are appropriately assigned to each subplot, providing a clear indication of the depicted metric. The resulting visualization offers valuable insights into the stability and consistency of the SVM model's performance, enabling stakeholders to observe trends, fluctuations, or convergence patterns over multiple training cycles. This graphical representation serves as a comprehensive tool for interpreting the model's behavior and facilitating informed decision-making regarding its reliability and effectiveness.

A group of graphs showing different types of data

Description automatically generated with medium confidence

### 4.4.10 Figure: Prediction using SVM.

### 4.4.11 Decision Tree

y\_pred = decision\_tree\_model.predict(X\_test)

  accuracy = accuracy\_score(Y\_test, y\_pred) \* 100

  precision = precision\_score(Y\_test, y\_pred, average='weighted') \* 100

  recall = recall\_score(Y\_test, y\_pred, average='weighted') \* 100

  f1 = f1\_score(Y\_test, y\_pred, average='weighted') \* 100

  accuracy\_list.append(round(accuracy, 2))

  precision\_list.append(round(precision, 2))

  recall\_list.append(round(recall, 2))

  f1\_list.append(round(f1, 2))

A decision tree model's performance is systematically evaluated through 100 iterations, each involving the random partitioning of the dataset into training and testing sets. The model is trained on the training set for each iteration, and subsequently, predictions are made on the test set. Performance metrics, including accuracy, precision, recall, and F1 score, are computed for each iteration, providing a comprehensive assessment of the model's classification capabilities. The resulting metrics are then aggregated and stored in lists, which are organized into a dictionary under the label 'decision\_tree\_model'. This systematic evaluation allows for a thorough understanding of the decision tree model's robustness and effectiveness across various training scenarios, offering valuable insights for further analysis and interpretation in your report.

A group of graphs showing different types of data

Description automatically generated with medium confidence

### 4.4.12 Figure: Prediction using Decision Tree

### 4.4.13 Naïve Bayes Model

 y\_pred = naive\_bayes\_model.predict(X\_test)

  accuracy = accuracy\_score(Y\_test, y\_pred) \* 100

  precision = precision\_score(Y\_test, y\_pred, average='weighted') \* 100

  recall = recall\_score(Y\_test, y\_pred, average='weighted') \* 100

  f1 = f1\_score(Y\_test, y\_pred, average='weighted') \* 100

  accuracy\_list.append(round(accuracy, 2))

  precision\_list.append(round(precision, 2))

  recall\_list.append(round(recall, 2))

  f1\_list.append(round(f1, 2))

In this code segment, the performance of a Naive Bayes model is systematically assessed through 100 iterations. For each iteration, the dataset is randomly split into training and testing sets, ensuring diverse scenarios for model evaluation. The Naive Bayes model is trained on the training set, and predictions are subsequently generated for the corresponding test set. Key classification metrics, including accuracy, precision, recall, and F1 score, are computed for each iteration, providing a comprehensive overview of the model's effectiveness under various conditions. The resulting metric values are then rounded and stored in lists, which are organized into a dictionary labeled 'naive\_bayes\_model.' This systematic evaluation allows for a thorough understanding of the Naive Bayes model's robustness and performance across multiple training instances, offering valuable insights for interpretation and analysis in the report.

A group of blue lines

Description automatically generated

### 4.4.14 Figure: Prediction using Naïve Bayes Model

Average Performance result for 100 iterations of both the algorithms

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Accuracy score | Precision score | Recall score | F1 score |
| Random Forest | 95.85 | 95.94 | 95.85 | 95.84 |
| Logistic Regression | 58.13 | 34.93 | 58.13 | 43.38 |
| K Nearest Neighbor (KNN) | 75.36 | 76.22 | 75.36 | 73.62 |
| Support Vector Machine | 62.47 | 40.29 | 62.47 | 48.55 |
| Decision Tree | 92.82 | 92.98 | 92.82 | 92.82 |
| Naïve Bayes | 62.38 | 49.60 | 62.38 | 49.33 |

### Figure 4.4.15: Average Performance for 100 iterations

### 4.4.16 Details of the 100 iterations graph representation explained.

The graph shown in the figures shows ups and down in the accuracy and other graphs. This is because of the random\_state parameter that is used while test\_train\_split, that is used to split the data. This random\_state parameter ensures that the split generated is reproducible. If the same seed i.e random\_state(42) is used through all the model building iterations then the similar training and testing sets are generated. This is useful to generate the similar experiments. In this project, Throughout the iterations, there is not much difference in accuracy number. The difference in minimal.

### 4.4.17 Detailed Graph Explanation of Graphs

**Accuracy Line Graph:**

X-axis: Iterations (from 1 to 100).

Y-axis: Accuracy scores.

Explanation: The precision line graph illustrates the variations in the Random Forest model's accuracy among distinct random divisions of the dataset. A steady line or an increasing trend denotes consistent performance across subsets, much like in logistic regression. Variations could be an indication of data split randomness or sensitivity to specific subsets.

**Precision Line Graph:**

X-axis: Iterations (from 1 to 100).

Y-axis: Precision scores.

Explanation: The accuracy of the model's positive predictions throughout several iterations is shown by the precision line graph. Like accuracy, a steady or rising trend suggests a steady and dependable ability to accurately recognize true positives. Variations may be indicative of variations in accuracy among various subsets of the data.

**Recall Line Graph:**

X-axis: Iterations (from 1 to 100).

Y-axis: Recall scores.

Explanation: The model's capacity to identify genuine positives in comparison to actual positives in various subsets is illustrated by the recall line graph. Consistent or increasing trend indicates a stable ability to capture pertinent occurrences, like accuracy and precision. Variations may suggest susceptibility to various subsets or distributions of the data.

**F1 Score Line Graph:**

X-axis: Iterations (from 1 to 100).

Y-axis: F1 scores.

Explanation: Combining precision and recall, the F1 score line graph offers a comprehensive assessment of the model's performance. A steady or rising trend in F1 scores, like the other metrics, indicates a steady trade-off between recall and precision over several rounds. Variations may indicate situations in which the model finds it difficult to strike a balance between these two metrics.

Understanding the stability, consistency, and generalization capacity of the models is facilitated by interpreting these line graphs collectively. While significant variances may point to areas for improvement or subgroups where the model may not generalize well, consistent trends across repetitions suggest resilience.

### 4.4.18 Box Plot Performance comparison of all the machine learning algorithms used

* **Accuracy Score**: In machine learning, the accuracy score is a performance metric that measures the percentage of properly predicted instances out of all the predictions a classification model has made. It can be expressed as a number between 0 and 1, where 1 denotes complete accuracy. The calculation involves dividing the total number of guesses by the number of right forecasts. Although simple, accuracy might not be the best choice for datasets that are unbalanced. Effective performance is not always ensured by high accuracy, particularly in situations when class distribution is not uniform.
* As we can see in the graph, Random Forest classifier gave the highest average accuracy score which is 95.85.

d\_1 = result['logistic\_regression\_model']['accuracy\_list']

d\_2 = result['random\_forest\_model']['accuracy\_list']

d\_3 = result['knn\_model']['accuracy\_list']

d\_4 = result['svm\_model']['accuracy\_list']

d\_5 = result['decision\_tree\_model']['accuracy\_list']

d\_6 = result['naive\_bayes\_model']['accuracy\_list']

data\_to\_plot = [d\_1, d\_2, d\_3, d\_4, d\_5, d\_6]

plt.boxplot(data\_to\_plot, vert=False, widths=0, medianprops=dict(marker='o', color='blue', markersize=5), showfliers=False)

for i in range(0, 100):

if i%5 == 0:

plt.axvline(x=i, linestyle='--', color='lightgray', label='Vertical Line 1')

plt.title('Accuracy comparison of Machine Learning Algorithms')

plt.yticks([1, 2, 3, 4, 5, 6], ['Logistic Regression', 'Random Forest', 'K Nearest Neighbour', 'SVM', 'Decision Tree', 'Naive Bayes'])

plt.xlabel('Accuracy')

plt.ylabel('Machine Learning Algorithms')

plt.show()

A graph with numbers and lines

Description automatically generated

* **Precision Score:** In machine learning, precision is a statistic that evaluates how well a classification model makes positive predictions. It is the proportion of correctly predicted true positives to the total of correctly predicted false positives. Greater precision, which is expressed as a number between 0 and 1, emphasizes the dependability of positive predictions by showing fewer false positives. In situations like fraud detection or medical diagnosis, where reducing false positives is critical, precision is vital. A model that successfully detects affirmative cases with few misclassifications is indicated by a precision score that is closer to 1, which enhances the validity of the model's performance evaluation.
* As we can see in the graph, Random Forest classifier gave the highest precision score which is 95.94.

d\_1 = result['logistic\_regression\_model']['precision\_list']

d\_2 = result['random\_forest\_model']['precision\_list']

d\_3 = result['knn\_model']['precision\_list']

d\_4 = result['svm\_model']['precision\_list']

d\_5 = result['decision\_tree\_model']['precision\_list']

d\_6 = result['naive\_bayes\_model']['precision\_list']

data\_to\_plot = [d\_1, d\_2, d\_3, d\_4, d\_5, d\_6]

plt.boxplot(data\_to\_plot, vert=False, widths=0, medianprops=dict(marker='o', color='blue', markersize=5), showfliers=False)

for i in range(0, 100):

if i%5 == 0:

plt.axvline(x=i, linestyle='--', color='lightgray', label='Vertical Line 1')

plt.title('Precision comparison of Machine Learning Algorithms')

plt.yticks([1, 2, 3, 4, 5, 6], ['Logistic Regression', 'Random Forest', 'K Nearest Neighbour', 'SVM', 'Decision Tree', 'Naive Bayes'])

plt.xlabel('Precision')

plt.ylabel('Machine Learning Algorithms')

plt.show()

A graph with numbers and lines

Description automatically generated

* **F1 Score:** In machine learning, the F1 score is a balanced metric that assesses a classification model's overall performance by combining precision and recall. The F1 score, which is expressed as a number between 0 and 1, takes into account both false positives and false negatives and provides a harmonic mean of recall and precision. It comes in very handy when working with unbalanced datasets. Higher F1 scores signify a model's capacity to strike a balance between recall and precision, which qualifies it for applications like spam detection or medical diagnosis where the repercussions of false positives and negatives can be severe.
* As we can see in the graph, Random Forest classifier gave the highest F1 score which is 95.85.

d\_1 = result['logistic\_regression\_model']['f1\_list']

d\_2 = result['random\_forest\_model']['f1\_list']

d\_3 = result['knn\_model']['f1\_list']

d\_4 = result['svm\_model']['f1\_list']

d\_5 = result['decision\_tree\_model']['f1\_list']

d\_6 = result['naive\_bayes\_model']['f1\_list']

data\_to\_plot = [d\_1, d\_2, d\_3, d\_4, d\_5, d\_6]

plt.boxplot(data\_to\_plot, vert=False, widths=0, medianprops=dict(marker='o', color='blue', markersize=5), showfliers=False)

for i in range(0, 100):

if i%5 == 0:

plt.axvline(x=i, linestyle='--', color='lightgray', label='Vertical Line 1')

plt.title('Recall comparison of Machine Learning Algorithms')

plt.yticks([1, 2, 3, 4, 5, 6], ['Logistic Regression', 'Random Forest', 'K Nearest Neighbour', 'SVM', 'Decision Tree', 'Naive Bayes'])

plt.xlabel('Recall')

plt.ylabel('Machine Learning Algorithms')

plt.show()

A graph with numbers and lines

Description automatically generated

* **Recall Score:** In machine learning, recall is a performance metric that evaluates a classification model's accuracy in identifying all pertinent instances of a given class. It determines the ratio of true positives to the total of true positives and false negatives, and is also referred to as sensitivity or true positive rate. A higher recall, denoted by a number between 0 and 1, suggests that the model is more successful in identifying a greater percentage of true positive cases.
* As we can see in the graph, Random Forest classifier gave the highest F1 score which is 95.85.

d\_1 = result['logistic\_regression\_model']['recall\_list']

d\_2 = result['random\_forest\_model']['recall\_list']

d\_3 = result['knn\_model']['recall\_list']

d\_4 = result['svm\_model']['recall\_list']

d\_5 = result['decision\_tree\_model']['recall\_list']

d\_6 = result['naive\_bayes\_model']['recall\_list']

data\_to\_plot = [d\_1, d\_2, d\_3, d\_4, d\_5, d\_6]

plt.boxplot(data\_to\_plot, vert=False, widths=0, medianprops=dict(marker='o', color='blue', markersize=5), showfliers=False)

for i in range(0, 100):

if i%5 == 0:

plt.axvline(x=i, linestyle='--', color='lightgray', label='Vertical Line 1')

plt.title('Recall comparison of Machine Learning Algorithms')

plt.yticks([1, 2, 3, 4, 5, 6], ['Logistic Regression', 'Random Forest', 'K Nearest Neighbour', 'SVM', 'Decision Tree', 'Naive Bayes'])

plt.xlabel('Recall')

plt.ylabel('Machine Learning Algorithms')

plt.show()

A graph with blue dots and white text

Description automatically generated

### 4.4.19 Model Serialization

**Create Pickle File**

import pickle

pickle.dump(randomForestModel, open('breast\_cancer.pkl','wb'))

Pickle

Explanation:

The code uses the pickle module in Python to serialize and save the trained Random Forest model (randomForestModel) to a file. pickle.dump is a method that serializes the model and writes it to a file in binary mode ('wb').

The file is named 'breast\_cancer.pkl', and it will be saved in the current working directory.

## 5. Results

The implementation of the breast cancer detection framework is hosted on a live server accessible via the following IP address: **http://3.84.179.19/index.** This deployment has been achieved using Flask, a web framework in Python, allowing seamless interaction with the predictive model. The deployment is facilitated through Amazon Web Services (AWS), demonstrating the scalability and reliability of the solution. Users can access the web interface, create accounts, and utilize the 'Predict Cancer' tab to input relevant values for breast cancer diagnosis, providing a user-friendly and accessible platform for the deployment of the machine learning model.

## 5.1 Significance of this project

The integration of machine learning models into medical diagnostics holds the potential to streamline decision-making processes and improve the accuracy of breast cancer classifications. By automating aspects of tumor classification, medical professionals can dedicate more time to patient care and personalized treatment strategies.

This project aligns with the broader initiative to harness the power of data-driven technologies in the field of healthcare, contributing to advancements that can positively impact patient outcomes and the overall efficiency of diagnostic procedures.

In the subsequent sections, we will delve into the results obtained, providing a comprehensive exploration of the breast cancer classification project.

## 5.2 Project Demonstration and User Interface

To showcase the practical implementation of the breast cancer classification models, a user-friendly interface was developed using Flask, a web framework for Python. The interface allows users to input values for relevant variables and receive real-time predictions regarding the malignancy or benignancy of breast tumors. This interactive component serves as an intuitive tool for medical professionals and stakeholders to understand the application of machine learning in cancer diagnosis.

### 5.2.1 Flask Application Details

The Flask application provides a seamless experience for users to interact with the underlying machine learning models. By accessing the application, users can input specific feature values and observe the predicted outcome generated by the logistic regression and random forest models. The simplicity and accessibility of the interface contribute to the project's goal of making machine learning technologies more comprehensible and applicable in a healthcare context.

**1. Initialization:**

* Flask is imported, and the application is created.
* A secret key for session management is set.
* A connection to a MongoDB database is established using the pymongo library.
* The trained machine learning model (Random Forest) is loaded from a pickle file.

**2. Authentication:**

**-** Basic user authentication is implemented with sign-in and sign-up routes. **-** Passwords are hashed using bcrypt before storing them in the database.

**3.** Routes:

**-** Several routes handle different aspects of the application:

* / and /signin: Routes related to user authentication.
* /signup: Route for user registration. **o** /index and /home: Routes for rendering home pages. **o** /getArray: Route for receiving JSON data containing feature values.
* /predict: Route for making predictions based on the provided feature values.

**4. Prediction:**

* The /predict route receives feature values as JSON, reshapes them, and sends them to the machine learning model for prediction.
* The predicted result is then rendered in the HTML template.

**5. User Interface:**

* HTML templates are used for rendering pages, including sign-in, sign-up, home, prediction, and result pages.
* The user interface allows users to input values, submit forms, and view predictions.

**6. MongoDB Integration:**

- User information is stored and retrieved from a MongoDB database.

**7. Model Information**

- There is a route /model-info for rendering a page that presumably contains information about the machine learning model.

**8. Contact Form:**

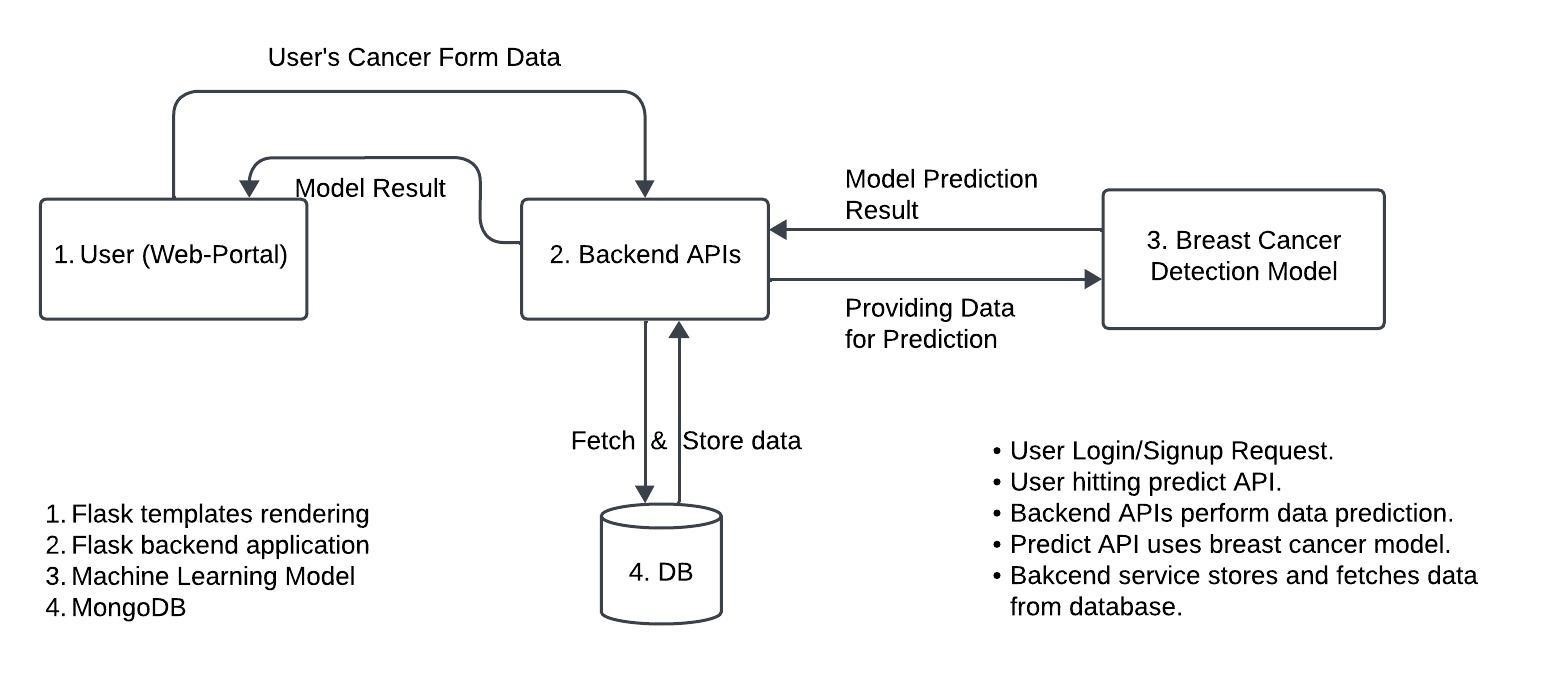
- There is a route /contact for rendering a contact form.

**9. Run the Application:**

- The application runs when the script is executed directly (if \_\_name\_\_ == '\_\_main\_\_':).

**10. Deployment:**

- The application is set to run in debug mode (app.run(debug=True)), which is suitable for development but should be disabled in a production environment.



### Figure 5.2.2 : Flow Chart

### 5.2.3 Deployment on Amazon Service

For broader accessibility, the Flask application was deployed using Amazon Web Services (AWS). Leveraging the scalability and reliability of AWS, the deployment ensures that the breast cancer classification framework is available through a dedicated IP address. Interested parties can access the project by visiting **http://3.84.179.19/index.** This deployment strategy enhances the reach and availability of the project, enabling seamless utilization by medical professionals, researchers, and other stakeholders.

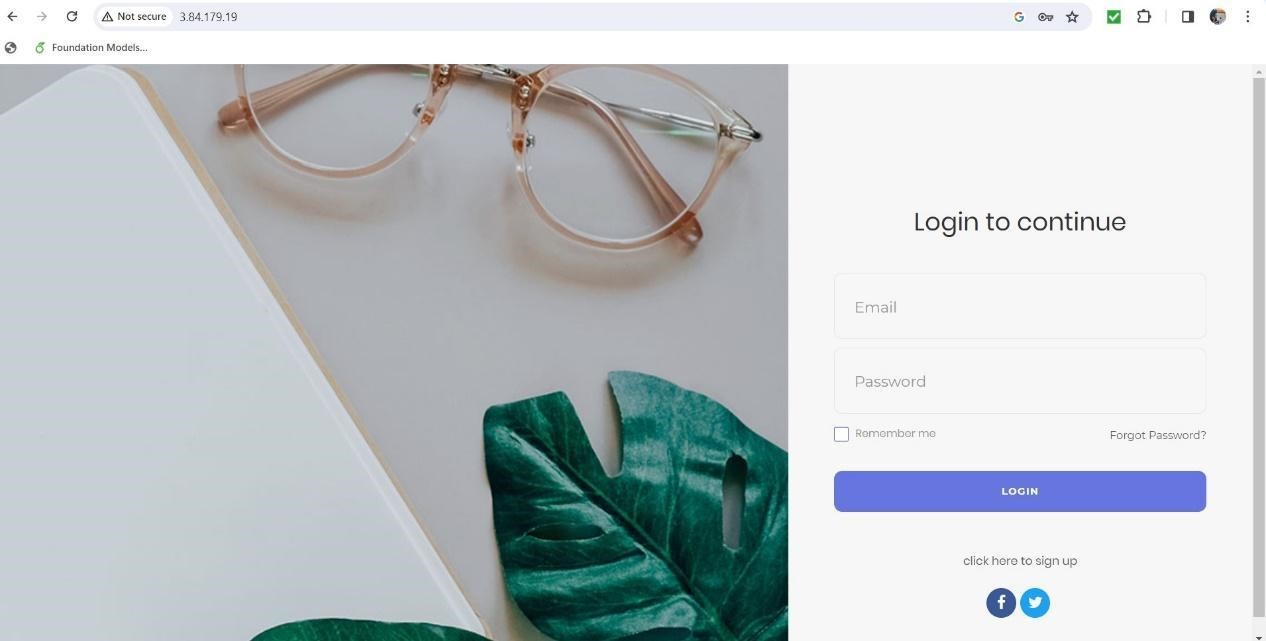
### 5.2.4 Utilization of Machine Learning Models

The project encompasses the implementation of two machine learning models, logistic regression, and random forest, for breast cancer classification. Performance metrics, including accuracy, precision, recall, and ROC AUC, were evaluated on both training and testing datasets, providing insights into the models' capabilities and generalization performance.

In the subsequent sections, we delve into the detailed methodologies, presenting the steps taken during data preprocessing, model training, and evaluation. The project's outcomes, limitations, and potential avenues for future exploration are also discussed, offering a comprehensive overview of the breast cancer classification project.

### 5.2.5 Implementation using Flask.

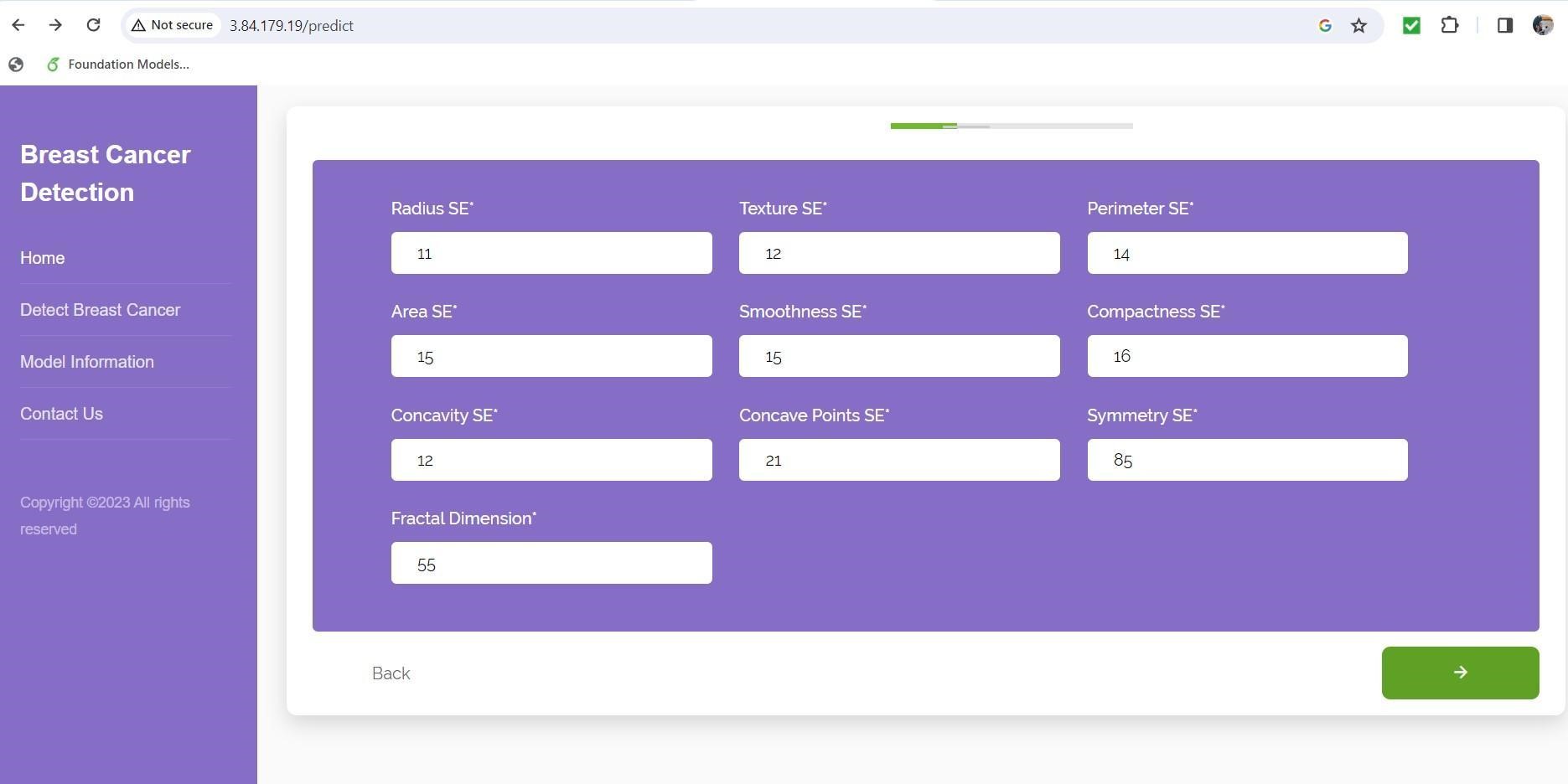
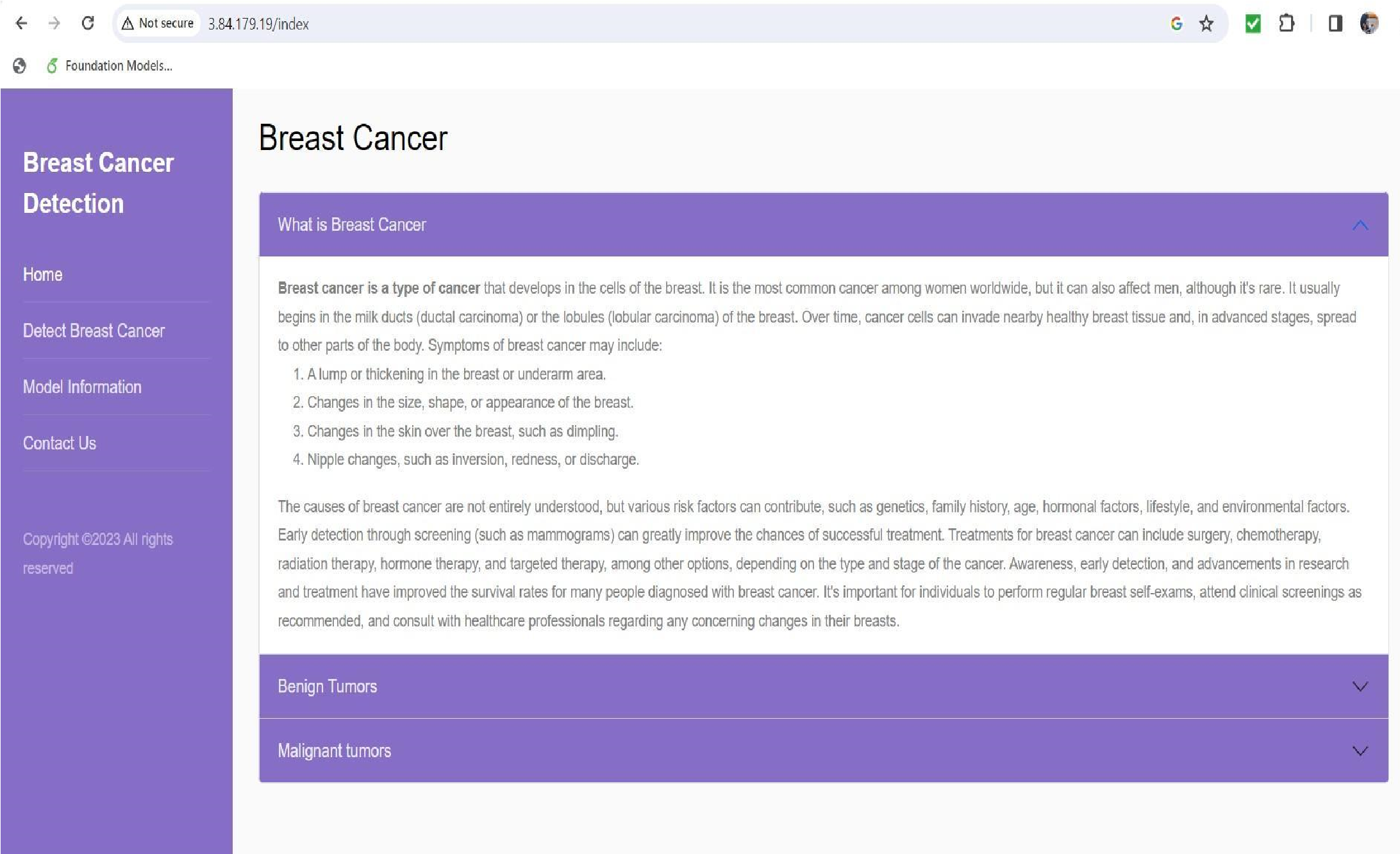
The project features a web-based interface created with Flask, allowing users to input relevant parameters for breast cancer prediction. The interface is designed to be intuitive and accessible, facilitating a seamless user experience. To ensure secure access to the application, user authentication mechanisms were implemented using bcrypt for password hashing. MongoDB was employed to store user credentials securely, contributing to a robust authentication system. The Flask application seamlessly integrates with the breast cancer prediction model, allowing users to receive real-time predictions based on input parameters. The model, trained using machine learning techniques, accurately predicts the likelihood of breast cancer, providing valuable insights for medical diagnosis. A dedicated predictive system was developed within the Flask application, enabling users to obtain predictions swiftly and efficiently. Users can input specific diagnostic features, and the system returns predictions promptly, enhancing the accessibility of breast cancer prediction. The application includes dedicated sections for displaying information about the machine learning model (/model-info) and a contact form (/contact). These sections enhance the user experience by providing insights into the underlying technology and offering a means for users to interact with the project team. These images has been captured from the deployed service that is currently running on the AWS EC2 machine.



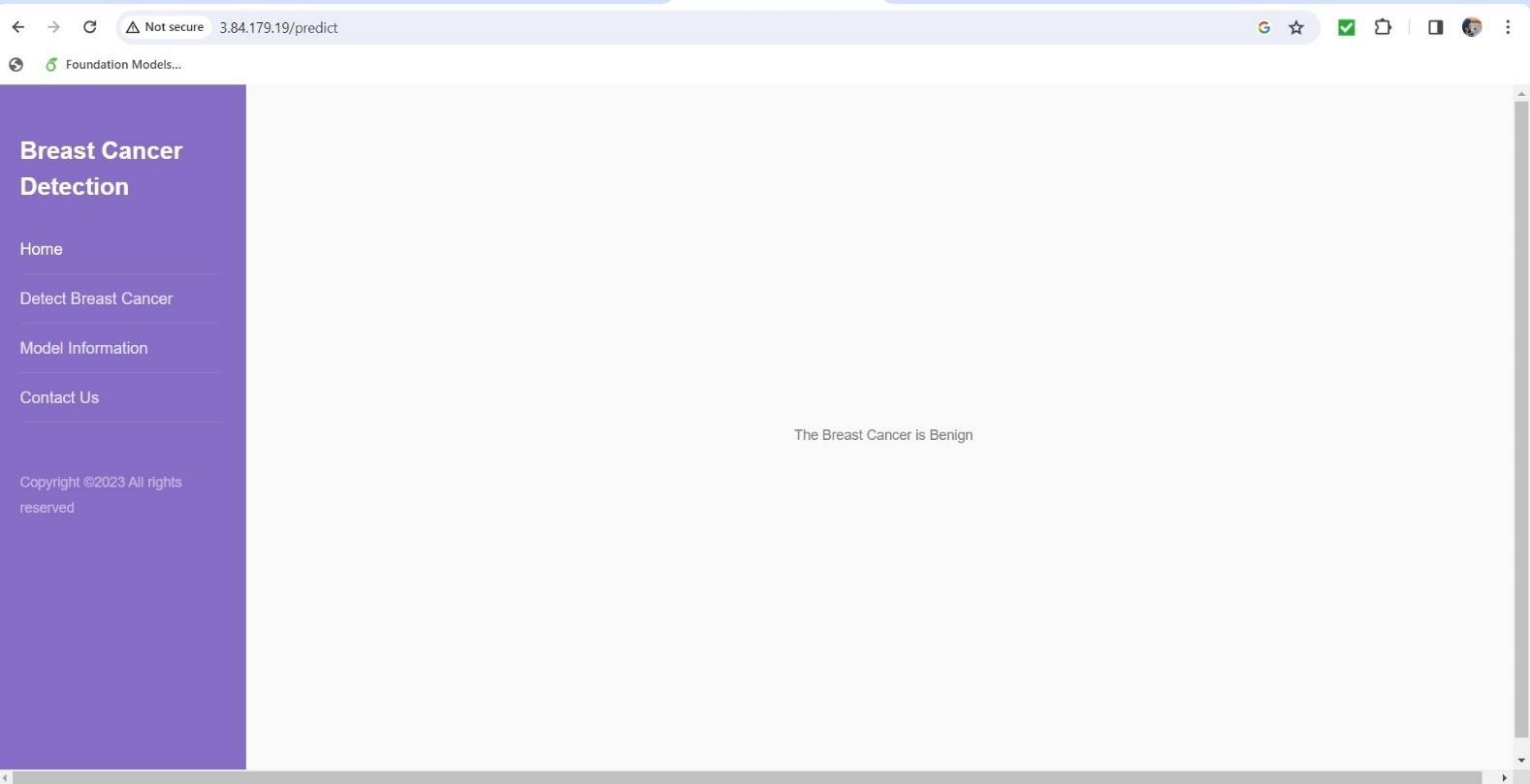
### Figure 5.2.6 : Login Page

- First we have login or create a new account to access the UI. Fill the blanks with username and password and click login.

### Figure 5.*2.7* Home page



### Figure 5.*2.8* Detect Cancer Page



### Figure 5.2.9: Result page

1. **Account Creation and Login**

Users have the option to create an account by providing essential details such as email, name, and contact information. This information is securely stored in the MongoDB database.

Once registered, users can log in to the system using their email and password. The Flask application employs bcrypt for secure password hashing, ensuring the confidentiality of user credentials.

1. **Dashboard Navigation**

After successfully logging in, users are directed to a personalized dashboard where they can access various features of the application.

The dashboard prominently features a "Predict Cancer" tab, providing users with the ability to input diagnostic parameters for breast cancer prediction.

1. **Breast Cancer Prediction**

Within the "Predict Cancer" tab, users encounter an interactive form that prompts them to enter specific diagnostic values, including features such as radius mean, texture mean, perimeter mean, and more.

Users submit the form, triggering the Flask application to process the input through the integrated machine learning model.

1. **Real-Time Prediction**

The machine learning model swiftly analyzes the user-input diagnostic parameters and provides a real-time prediction regarding the likelihood of breast cancer.

The prediction is then presented to the user on the UI, clearly indicating whether the condition is classified as malignant or benign.

1. **Model Information and Contact Form**

Users have the option to explore additional sections of the application, such as the "Model Information" and "Contact" pages.

The "Model Information" page offers insights into the underlying machine learning model, providing users with a better understanding of the technology driving the predictions.

The "Contact" page serves as a communication channel, allowing users to interact with the project team for inquiries or additional information.

**Code understanding:**

from flask import Flask, render\_template, url\_for, request, session, redirect, flash, jsonify

import pickle

import numpy as np

import bcrypt

from pymongo import MongoClient

from time import sleep

app = Flask('\_\_name\_\_')

app.secret\_key = 'icXjsFoQxp'

app.config['SESSION\_TYPE'] = 'filesystem'

app.debug = True

client = MongoClient('localhost', 27017) #host uri

db = client.breast\_cancer

model=pickle.load(open('breast\_cancer.pkl','rb'))

### Authentication APIS ###

@app.route('/')

@app.route('/signin', methods=['GET', 'POST'])

def signin():

try:

if request.method == 'POST':

users = db.users

signin\_user = users.find\_one({'email': request.form['email']})

if signin\_user:

if bcrypt.hashpw(request.form['pass'].encode('utf-8'), signin\_user['password']) == \

signin\_user['password']:

session['email'] = request.form['email']

return redirect(url\_for('main'))

else:

flash('Email or password is incorrect !')

else:

flash('User with this email does not exist !')

# return render\_template('login/index.html')

return render\_template('login/index.html')

except Exception as e:

print(e)

return render\_template('login/index.html')

@app.route("/index")

def main():

return render\_template('home.html')

@app.route('/home')

def home():

return render\_template('home.html')

@app.route("/signup", methods=['POST', 'GET'])

def signup():

print(db.list\_collection\_names())

if request.method == 'POST':

users = db.users

print(users)

signup\_user = users.find\_one({'email': request.form['email']})

if signup\_user:

flash(request.form['email'] + ' email is already exist')

return redirect(url\_for('signup'))

hashed = bcrypt.hashpw(request.form['pass'].encode('utf-8'), bcrypt.gensalt(14))

db.users.insert\_one({

'password': hashed,

'email': request.form['email'],

'name': request.form['name'],

'contact': request.form['contact']

})

return redirect(url\_for('signin'))

return render\_template('signup/index.html')

### Authentication APIS ENDS HERE ###

@app.route('/getArray', methods=['GET', 'POST'])

def getArray():

feature\_columns = ["radius\_mean", "texture\_mean", "perimeter\_mean", "area\_mean", "smoothness\_mean", "compactness\_mean",

"concavity\_mean", "concave\_points\_mean", "symmetry\_mean", "fractal\_dimension\_mean", "radius\_se", "texture\_se",

"perimeter\_se", "area\_se", "smoothness\_se", "compactness\_se", "concavity\_se", "concave\_points\_se", "symmetry\_se",

"fractal\_dimension\_se", "radius\_worst", "texture\_worst", "perimeter\_worst", "area\_worst", "smoothness\_worst",

"compactness\_worst", "concavity\_worst", "concave\_points\_worst", "symmetry\_worst", "fractal\_dimension\_worst",

]

print(request.get\_json())

data = request.get\_json()

data\_arr = data['data']

res = {}

for i in range(0,30):

res[feature\_columns[i]] = data\_arr[i]

return jsonify(

res

)

@app.route('/predict',methods=["GET", "POST"])

def predict():

if request.method == 'POST':

feature\_columns = ["radius\_mean", "texture\_mean", "perimeter\_mean", "area\_mean", "smoothness\_mean", "compactness\_mean",

"concavity\_mean", "concave\_points\_mean", "symmetry\_mean", "fractal\_dimension\_mean", "radius\_se", "texture\_se",

"perimeter\_se", "area\_se", "smoothness\_se", "compactness\_se", "concavity\_se", "concave\_points\_se", "symmetry\_se",

"fractal\_dimension\_se", "radius\_worst", "texture\_worst", "perimeter\_worst", "area\_worst", "smoothness\_worst",

"compactness\_worst", "concavity\_worst", "concave\_points\_worst", "symmetry\_worst", "fractal\_dimension\_worst",

]

feature = []

for column in feature\_columns:

feature.append(float(request.form.get(column)))

# return jsonify(data="1")

feature\_tuple = tuple(feature)

# input\_data = (13.54, 14.36, 87.46, 566.3, 0.09779, 0.08129, 0.06664, 0.04781, 0.1885, 0.05766, 0.2699, 0.7886, 2.058, 23.56, 0.008462, 0.0146, 0.02387, 0.01315, 0.0198, 0.0023, 15.11, 19.26, 99.7, 711.2, 0.144, 0.1773, 0.239, 0.1288, 0.2977, 0.07259)

input\_data\_as\_numpy\_array = np.asarray(feature\_tuple)

input\_data\_reshaped = input\_data\_as\_numpy\_array.reshape(1,-1)

prediction=model.predict(input\_data\_reshaped)

print(prediction)

sleep(5)

if (prediction[0] == "0"):

print('The Breast cancer is Malignant')

return render\_template('result/index.html',result='The Breast Cancer is Malignant')

return jsonify({

"result": 'The Breast Cancer is Malignant'

})

else:

print('The Breast Cancer is Benign')

return render\_template('result/index.html',result='The Breast Cancer is Benign')

# return render\_template('index.html',prediction\_text='The Breast Cancer is Benign')

return jsonify({

"result": 'The Breast Cancer is Benign'

})

return render\_template('predict-cancer/index.html')

@app.route('/predict\_test',methods=["POST"])

def login\_user():

data = [

"diagnosis","radius\_mean","texture\_mean","perimeter\_mean","area\_mean","smoothness\_mean",

"compactness\_mean","concavity\_mean","concave points\_mean","symmetry\_mean",

"fractal\_dimension\_mean","radius\_se","texture\_se","perimeter\_se","area\_se","smoothness\_se",

"compactness\_se","concavity\_se","concave points\_se","symmetry\_se","fractal\_dimension\_se",

"radius\_worst","texture\_worst","perimeter\_worst","area\_worst","smoothness\_worst",

"compactness\_worst","concavity\_worst","concave points\_worst","symmetry\_worst",

"fractal\_dimension\_worst",]

print("request", request)

input\_data = (13.54,14.36,87.46,566.3,0.09779,0.08129,0.06664,0.04781,0.1885,0.05766,0.2699,0.7886,2.058,23.56,0.008462,0.0146,0.02387,0.01315,0.0198,0.0023,15.11,19.26,99.7,711.2,0.144,0.1773,0.239,0.1288,0.2977,0.07259, 0, 0)

input\_data\_as\_numpy\_array = np.asarray(input\_data)

input\_data\_reshaped = input\_data\_as\_numpy\_array.reshape(1,-1)

out, acc, t = model.predict(input\_data\_reshaped)

if(out==1):

output = 'Malignant'

else:

output = 'Benign'

acc\_x = acc[0][0]

acc\_y = acc[0][1]

if(acc\_x>acc\_y):

acc1 = acc\_x

else:

acc1=acc\_y

return render\_template('result.html', output=output, time=t)

@app.route("/model-info")

def model\_info():

return render\_template('model-information/index.html')

@app.route("/contact")

def contact():

return render\_template('contact-form-04/index.html')

if(\_\_name\_\_=='\_\_main\_\_'):

app.run(debug=True, port=8000)

# 6.Conclusion

In summary, this project successfully combines machine learning methodologies, employing logistic regression and random forest classification, to accurately predict breast cancer based on diagnostic parameters. The deployed Flask web application streamlines user interaction, allowing for account creation, secure login, and instant cancer predictions. The utilization of Amazon services ensures a reliable and scalable deployment environment. The inclusion of a comprehensive model information section enhances user transparency and understanding. Overall, this project signifies a cohesive integration of machine learning and web development, contributing to the creation of an accessible and effective tool for breast cancer prediction.

# 7.Scope and Future Works

Scope:

This project lays the groundwork for further advancements in the domain of breast cancer prediction and detection. The integration of machine learning techniques into a user-friendly web application opens avenues for widespread use and accessibility. The scope includes:

1. Enhanced User Experience Continuous improvement of the web application's interface to ensure a seamless and intuitive user experience.
2. Model Refinement: Iterative refinement of the machine learning model to increase accuracy and accommodate a broader range of diagnostic data
3. Feature Expansion: Integration of additional features and diagnostic parameters to enhance the model's predictive capabilities.
4. Collaboration with Healthcare Professionals: Collaborative efforts with medical professionals for real-world validation and incorporation of expert insights into the model.

Future Works

Building on the current achievements, the project's future endeavors include:

1. Integration of Advanced Models: Exploration and integration of advanced machine learning and deep learning models for more nuanced and accurate predictions.
2. Big Data Implementation: Scaling the project to handle large datasets and leveraging big data analytics for more comprehensive analysis.
3. Deployment in Clinical Settings: Collaborating with healthcare institutions to deploy the application in clinical settings for real-time patient support and diagnosis.
4. Incorporation of Genetic Data: Extending the model to incorporate genetic data for a more personalized and precise prediction of breast cancer susceptibility.
5. Mobile Application Development: Expanding accessibility by developing a mobile application version, reaching a broader user base.

In summary, the project's scope encompasses continuous improvement in user experience, model refinement, and feature expansion. Future works involve the integration of advanced models, big data analytics, deployment in clinical settings, incorporation of genetic data, and the development of a mobile application, aiming to make impactful contributions to breast cancer prediction and diagnosis.

# 8. References

<https://datahacker.rs/004-machine-learning-logistic-regression-model/><https://anasbrital98.github.io/blog/2021/Random-Forest/><https://link.springer.com/article/10.1007/s00432-023-04956-z><https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-023-05235-x>https://news.mit.edu/2019/using-ai-predict-breast-cancer-and-personalize-care-0507 <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0262349>

[https://www.mtu.edu/news/2021/11/machine-learning-reduces-uncertainty-in-breast-cancer-diagn oses.html](https://www.mtu.edu/news/2021/11/machine-learning-reduces-uncertainty-in-breast-cancer-diagnoses.html) <https://medium.com/swlh/breast-cancer-detection-using-logistic-regression-7bef0110aff5><https://www.mdpi.com/2076-3417/12/22/11455>

[https://mdpi-res.com/d\_attachment/biomedinformatics/biomedinformatics-02-00022/article\_deplo y/biomedinformatics-02-00022-v2.pdf?version=1656295070](https://mdpi-res.com/d_attachment/biomedinformatics/biomedinformatics-02-00022/article_deploy/biomedinformatics-02-00022-v2.pdf?version=1656295070) <https://www.kaggle.com/code/pratikkgandhi/predicting-breast-cancer-with-random-forest-95><https://ieeexplore.ieee.org/document/8644835>

<https://www.frontiersin.org/articles/10.3389/fonc.2023.1043463/full>[https://www.hindawi.com/journals/jhe/2021/5528622/ https://www.hindawi.com/journals/jhe/2021/5528622/ https://www.hindawi.com/journals/jhe/2021/5528622/](https://www.hindawi.com/journals/jhe/2021/5528622/)

[https://www.digitalocean.com/community/tutorials/how-to-make-a-web-application-using-flask-in-p ython-3](https://www.digitalocean.com/community/tutorials/how-to-make-a-web-application-using-flask-in-python-3)

[https://www.digitalocean.com/community/tutorials/how-to-create-your-first-web-application-usingflask-and-python-3](https://www.digitalocean.com/community/tutorials/how-to-create-your-first-web-application-using-flask-and-python-3) <https://codedamn.com/news/web-development/flask-python-web-app><https://www.geeksforgeeks.org/flask-tutorial/>

<https://www.youtube.com/watch?v=dhHOzye-Rms><https://www.youtube.com/watch?v=eOqlLa6paCI><https://aws.amazon.com/tutorials/serve-a-flask-app/><https://aws.amazon.com/blogs/opensource/automate-python-flask-deployment-to-the-aws-cloud/><https://www.twilio.com/blog/deploy-flask-python-app-aws><https://realpython.com/linear-regression-in-python/><https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html><https://www.nickmccullum.com/python-machine-learning/linear-regression-python/>

[https://www.freecodecamp.org/news/how-to-build-and-train-linear-and-logistic-regression-ml-mod els-in-python/](https://www.freecodecamp.org/news/how-to-build-and-train-linear-and-logistic-regression-ml-models-in-python/) <https://www.justintodata.com/linear-regression-machine-learning-python-tutorial/><https://www.geeksforgeeks.org/random-forest-regression-in-python/><https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html><https://stackabuse.com/random-forest-algorithm-with-python-and-scikit-learn/><https://www.geeksforgeeks.org/random-forest-classifier-using-scikit-learn/>

[https://machinelearningmastery.com/random-forest-ensemble-in-python](https://machinelearningmastery.com/random-forest-ensemble-in-python/)[/https://www.kaggle.com/ datasets/yasserh/breast-cancer-dataset https://www.kaggle.com/datasets/yasserh/breast-cancer-dataset](https://www.kaggle.com/datasets/yasserh/breast-cancer-dataset) <https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data><https://bmcresnotes.biomedcentral.com/articles/10.1186/s13104-019-4121-7><https://archive.ics.uci.edu/ml/datasets/Breast+Cancer><https://link.springer.com/chapter/10.1007/978-981-19-6450-3_21><https://365datascience.com/tutorials/python-tutorials/data-analysis-python/>

<https://www.geeksforgeeks.org/data-analysis-with-python/><https://www.freecodecamp.org/learn/data-analysis-with-python><https://www.udacity.com/blog/2020/04/python-and-data-analysis-cheat-sheet.html><https://www.youtube.com/watch?v=IxxGqoOksJ4><https://www.youtube.com/watch?v=KdmPHEnPJPs>

<https://www.youtube.com/watch?v=vPaXbA_xQFQ><https://realpython.com/python-data-cleaning-numpy-pandas/><https://www.youtube.com/watch?v=vPaXbA_xQFQ><https://datagy.io/pandas-data-cleaning/><https://datascientyst.com/data-cleaning-steps-python-example/>

<https://datahacker.rs/004-machine-learning-logistic-regression-model/>

<https://www.analyticsvidhya.com/blog/2021/05/bagging-25-questions-to-test-your-skills-on-random-forest-algorithm/>