Title: **Effective Classification of Wisconsin Breast Cancer diagnosis by using dimensionality reduction and feature selection approaches with ensemble learning.**

Table of Contents

List of Figures

List of Tables

ABSTRACT

This dissertation focused on the dimension reduction and feature selection methods to improve the Random Forest (RF) ensemble classifier for breast cancer detection. As a dimensionality reduction techniques, the Principal Component Analysis (PCA) and Factor Analysis (FA) are implemented. Whereas Chi-Square (Chi2) and Recursive Feature Elimination (RFE) approaches for feature selection have been deployed. These methodologies are used to analyze WBC and WDBC datasets. To address the problem of classifying with high accuracy and computational efficiency, a new hybrid method integrating PCA with RFE for an RF classifier is proposed. The results suggest that the hybrid model gained a high accuracy of 98.57% on the WBC dataset and 98.24% on the WDBC dataset, and it outperformed traditional methods.This study has been investigated and evaluated by using the precision, recall, and F1-score, and these measures also saw a considerable improvement. Our method also showed the shortest training and prediction time, again indicating its computational efficiency. A detailed comparison with some related existing works on identical datasets, indicated that the proposed method not only achieves better classification accuracy, but also in terms of computational cost, it is much better than other studies and thus proposed method is best choice to diagnose breast cancer. Results show that PCA + RFE with RF outcomes to be a balanced hybrid that provides high performance while maintaining a less computational cost, which can be a suitable endgame for the real-world applications in medical diagnostics.

Chapter 1 Introduction

1.1 Over View of Research:

Cancer is one of the widely prevalent and high-risk diseases around the world. There are more than 100 different types of disorders under this umbrella but they mostly happen when the cells in your body starts to multiplies out of control. Some of these include breast cancer, one of the most common and lethal forms of cancer. Although men and women are affected, the main victims are women (by far). Breast cancer is the second leading cause of cancer death among women in the United States, after lung cancer. Breast cancer is the most prevalent malignancyamong women and is projected to be responsible for 15% of all cancer deathsin the United States in 2020, with more than 42,000 women expected to diefrom the disease (*Cancer Facts & Figures 2020 | American Cancer Society*, 2020). Breast cancer is the most common cause of cancer death among women in Europe, with more than 130 000 women dying annually from the disease and accounting for nearly 18% of all cancer deaths throughout the continent (*Breast cancer statistics | Cancer Research UK*, 2020).

Breast cancer is a tumor that develops due to the uncontrolled proliferation of cells in the breast. All tumors are not cancerous; they are either benign or malignant. Malignant tumors are cancerous tumors that can metastasize to other sites, while benign tumors are not cancerous and usually do not spread. This rise in mortality has led to a growing interest in their early detection and the use of advanced medical technology including, machine learning models, for diagnosis and screening strategies (Siegel, Miller and Jemal, 2017; Sung *et al.*, 2021).

1.2 Research motivation:

The world population is growing at a very fast rate day by day. All of these factors contribute to new diseases and constitute an imminent challenge to the health and care system of the world. The basic health facilities are the basic and essential rights of all the citizens. Still, due to the new health challenges and diseases, the attention and treatments of all major and minor patients couldn't be provided by the Medical staff. The shortage in medical and health & care facilities calls researchers focus on designing such methodology through which would be able to rapidly diagnose the disease in an efficient and accurate way. Researchers are focused on implementing ML algorithms and knowledge-based techniques on medical data for the breast cancer as it has

potential to classify the healthy and non-healthy cells or tumors faster and more accurately. So the early detection of the disease in patient allows to take the necessary steps to save millions of lives worldwide and help doctors in the successful treatment of the disease.

1.3 Problem Statement:

This dissertation mainly aimed to produce a method for colossal and extensive classification of malignant (cancerous cell) and benign (non-cancerous cell) accurately, precisely and finally effectively. The proposed method involves implementation of commonly used ensembling algorithm, pre-processing steps, data analysing and cleaning, impact of feature selection, dimensionality reduction on the performance of supervised classifiers.

1.4 Research Objective:

This experimental and analytical research study has set up one main research objective: the cleaning of the used Wisconsin breast cancer datasets and the analysis of it, namely WBC and WDBC. How to deal with missing values problem. Cleaning the data and eliminating the useless features What feature scaling technique works best This research work used two different datasets with different and large features and we not only find which feature selection technique shows comparatively better results but also how proper selection of attributes can help the ensembling model (Random Forest) to achieve increased accuracy. Two methods for dimensionality reduction can allows classifiers to perform better.

1.5 Application:

Machine learning classifiers for early illness diagnosis and disease classification are growing extremely rapidly. These approaches help medical personnel precisely locate and recognize primary problems in patients through early prediction techniques. Alterations in testing and training data amounts also induce extinct transformations in model accuracy (Odajima and Pawlovsky, 2014). Computer-assisted diagnosis combined with the latest big data-based machine learning methods is becoming popular among paramedics to decrease misclassification errors. A single radiologist screening mammograms has greater chances of error, thus machine learning strategies are now widely applied not just in medicine but also for other fields like computer vision, treatment statistics, and medication type applications (Shaikh and Ali, 2019). Numerous

tools have been created to simplify the classification and diagnosis process for users, such as WEKA. Researchers utilized this tool with built-in popular algorithms and techniques for various undertakings, including applying it for disease diagnosis involving breast cancer (Asri *et al.*, 2016).

1.6 Contribution:

This work provides the brief analysis of few popular attribute selection, dimensionality reduction and the combination of the both techniques with Principal Component Analysis (PCA) with Recursive Features Elimination (RFE) technique along with Random Forest ensemble classifier. The outputs have been scored using the various important evaluation metrics related to the performances of each built approach. While the combination of both dimensionality reduction and the specific selection of the attributes are able to reach decent performance with highest accuracy, a detailed comparison has been made and our approach attains the best results.

1.7 Thesis Organization:

This dissertation is structured into five chapters. The Chapter 1 explains the background of cancer disease, top three cancer types with high mortality rate, research motivation, problem statement, research objectives, research background and layout of this dissertation. Chapter 2 shows approximately cancer tumor, breast cancer and its main classes, indications, symptoms or symptoms for breast cancer, separate types of breast cancer remedy, misclassification or error in prognosis of breast cancer, device studying based breast cancer diagnosis. The chapter 3 contains datasets description, architecture for proposed methodology, implemented ensemble classifier details, feature selection techniques used, implementation of dimensionality reduction techniques. In Chapter 4 experimental settings, results of Random Forest (RF) algorithm with features selection approaches, with dimensionality reduction approaches and hybrid combination of both RFE with PCA, comparative analysis between all the approaches that have been implemented and comparative analysis among the results of the highest achiever in this work with the existing work is presented. Chapter 5 presents the conclusion and future work of this research study.

Chapter 2 Research Background & Literature Review

2.1 Breast Tumor Description:

The human body consists of trillions, potentially millions of cells. These cells live and die within themselves, going through a cell cycle while being replaced in the process. There is tremendous explosive growth of these cells until they reach a certain age when they are being replaced much faster than cells are being supplanted (*Overall Cancer Statistics - Annual Report to the Nation*, 2025). This leads to the development of cancer. These cells multiply delicately, nurture, and die in cycle which leads to the formation of cancer. Usually, the scenario is less favorable for the fundamental cell: the cell will die and transform into a tumor. These cells are not static in their position and, with time, proliferate using the blood flow as their vessel to spread. They rapidly replace living cells with pathological cells while obliterating healthy tissues. There are many types of oncological diseases a patient may suffer from: each of them progresses along its path, which is normally dictated by the behavioral traits of the cells. For instance, in breast cancer and lung cancer, both diseases have distinct rates of growth and thus require different treatment plans. It is customary to classify tumor masses into two categories, these being 'abnormal tumor growth' and 'cancerous cell growth,' meaning while some are malignant others are benign. Although benign- uncontrolled body tissue- metastasizes into lower surrounding structures devoid of healthy tissue, they do not transform into other tissues.

Because of this behavior, they cannot metastasize to different parts of the body and do not invade surrounding normal tissues. This is the reason why benign tumors are not life-threatening (Mehran Emadi, 2015).

2.2 Basic Types of Breast cancer:

Although breast cancer is divided into many types, the least common are described below. Ductal carcinoma in situ (DICS), Infiltrating ductal carcinoma (IDC), Invasive lobular carcinoma (ILC), and Inflammatory breast cancer (IBC) (Khandezamin, Naderan and Rashti, 2020). In the Figure 1.

The images of a mammogram showing (a) Normal breast bone mammogram (b) Benign tumor (not cancerous) (c) Tumor (cancer) (*Overall Cancer Statistics - Annual Report to the Nation*, 2025).

Figure 2. 1. Mammogram images of breast cancer, normal, benign and malignant (*Overall Cancer Statistics - Annual Report to the Nation*, 2025).

2.3 Signs or symptoms for breast cancer:

Advances in cancer treatment through early detection and widespread availability of mammography have succeeded in flagging or locating numerous cases prior to their metastatic potential; however, reliance solely on screening can overlook intricate cases and still leave multifaceted issues unaddressed. A large number of them show some hard, non painful and gritty edged lumps, which are a new mass, also one of the major signs of breast cancer. Additionally, some lumps may be soft, rounded without edges, and seem tender. Thus, in the presence of any alarming signs, proper medical attention must be sought. These are some of the more common symptoms associated with breast cancer.

• Pain in the breast accompanied by other symptoms may occur as mild, moderate, or quite severe.

• Blood or other fluids may exude from the nipple apart from the milky fluid.

• Changes in the coloration or skin texture of nipple or breast so as to become red or pit-like or thickening of breast or nipple

• Deposits of subcutaneous tissue

• Development of giant breasts is another sign This condition is known as Gigantomastia.

• Pain in the nipple of the breast.

The important symptoms of breast cancer are presented in Figure 2. (Fletcher Jenna, 2024).

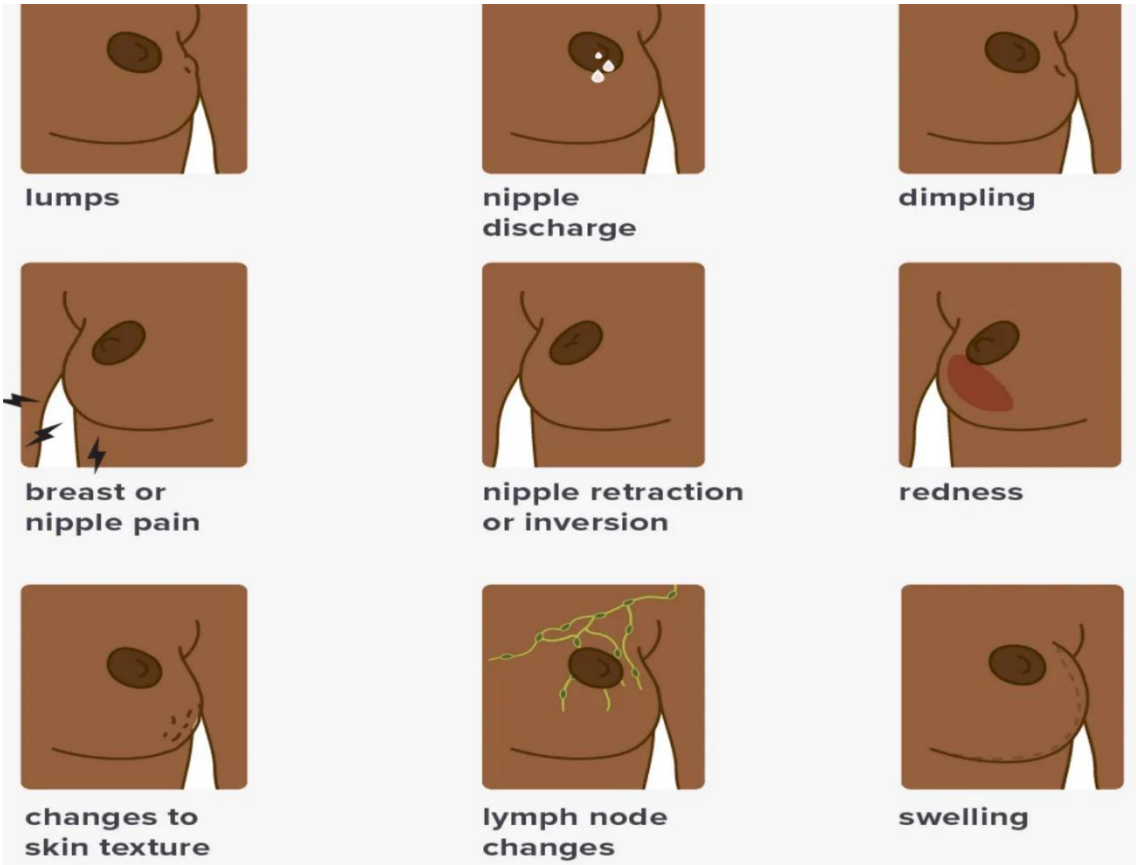


Figure 2. 2 Symptoms and signs of breast cancer are shown in (Fletcher Jenna, 2024).

2.4 Types of Breast Cancer Treatment:

Although there are other methods, the initial approach to dealing with the issue lies in performing a physical exam alongside advanced technology, as this is vital for proper diagnosis and early intervention.

- Targeted therapy
- Surgery
- Hormone therapy
- Chemotherapy

- Radiation

2.5 Misclassification of Breast Cancer:

Systematic bias associated with the diagnosis is one of the critical hindrances in the classification of diseases. Surgery sampling is one of the globally most common methods which is also very painful and time-consuming. Other methods of diagnosing breast cancer include imaging such as mammographies. However, this method has many limitations. For example, there is a chance of 10-30% misclassification by radiologists which can lead to: benign versus malignant misdiagnosis. Such an error in diagnosis can lead to disastrous consequences such as unnecessary surgical interventions which can put the life of the patient at risk. Therefore, there is an urgent requirement to develop the most effective, faster, and advanced methods of diagnosing breast cancer to initiate treatment earlier to save more lives (Kharya, 2012).

This is one of the main reasons why researchers these days are focusing on the latest knowledge, readily accessible information, and artificial intelligence (AI) to improve disease classification precision and efficacy. AI can also be beneficial in the early prediction and classification of breast cancer. The first analysis with AI tools and ML algorithms is useful for distinguishing between malignant and benign tumors without any surgical biopsy (Noble and Smith, 2013).

2.6 Diagnosing Breast Cancer Using Machine Learning:

In (Abdel-Ilah, 2017), the authors used the WBC dataset to apply the Feed Forward Back Propagation (FFBP) neural network along with the Artificial Neural Network (ANN), which resulted in an accuracy rate of 98%. In (Bihis and Roychowdhury, 2015), a unique generalized flow was proposed in Microsoft Azure - a cloud-based machine learning studio - which implements the algorithm on multiple datasets including WBC, and achieved accuracy between 78% to 97.5%. In (Abdel-zaher and Eldeib, 2015), Author Abdel et al. used Levenberg Marquardt (LM) and Deep Belief Neural Network (DBNN) that produced better results. In (Juneja and Rana, 2018), the Chi-2 feature selection was implemented on WBC and WDBC using NB, DT, and Random Forest (RF) to increase the accuracy. In this work, a comparative study of different classifiers was performed for WDBC. An ANN with Radial Basis Function (RBF) and Back Propagation Algorithm (BPA) along with multi-layer Perceptron (MLP) was used and achieved an accuracy of 99% as cited in (Helwan, Idoko and Abiyev, 2017). In this research article, the rough set SVM was presented (Chen

et al., 2011). To evaluate performance, confusion matrix-based metrics, the ROC curves, were employed.

The optimal performance accuracy was 96.87%. In this research work (You and Rumbe, 2010), various classifiers have been applied to the WDBC dataset and KNN showcases the highest accuracy. Mushtaq et al. utilizes several ML classifiers on WBC dataset with a PCA approach and achieved 99.2% accuracy using NB sigmoid (Mushtaq *et al.*, 2019). In article (Khandezamin, Naderan and Rashti, 2020) authors applied logistic regression based attributes selection and also applied a novel method of group method data handling (GMDH) on the three well known Wisconsin breast cancer datasets WBC, WDBC and Wisconsin prognostic breast cancer (WPBC). With the stated approach, they achieved 99.4% precision on WBC, 99.6% on WDBC, and 96.7% on WPBC breast cancer datasets.

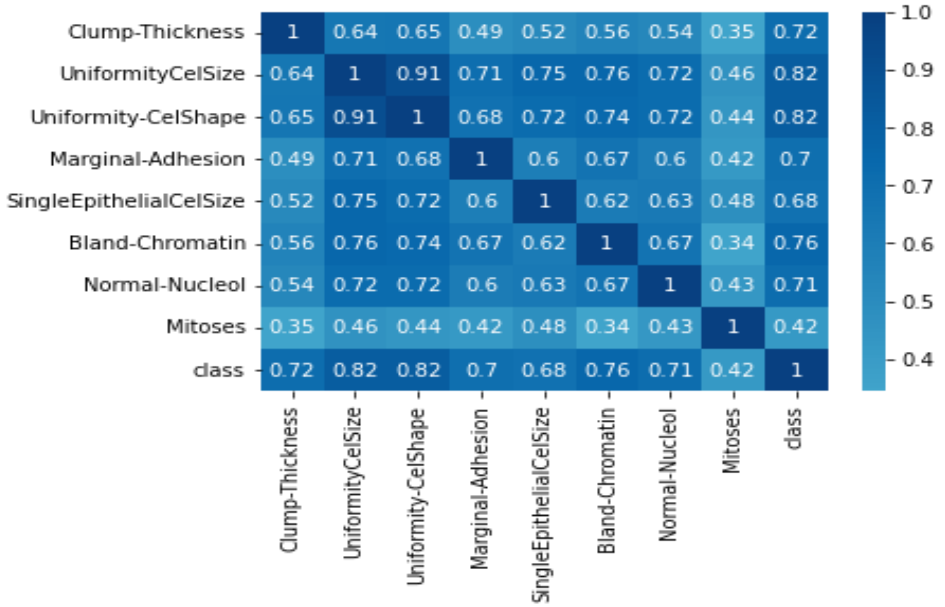
Chapter 3 Methodology

3.1 Datasets Description:

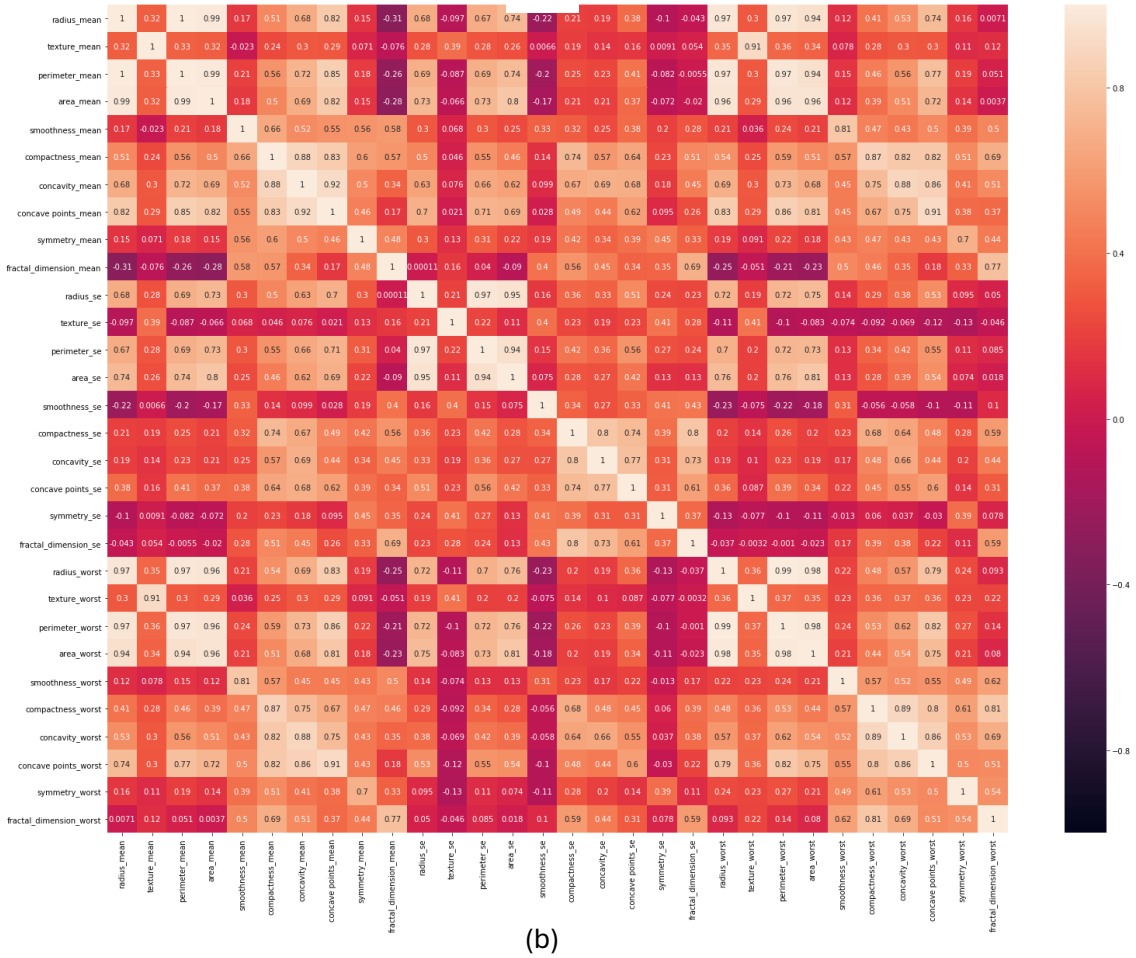
This study utilized two datasets from the UCI repository, which is very popular in the fields of machine learning and data science. These datasets are referred to as the Wisconsin Breast Cancer (WBC) in (WIlliam Wolberg, 1992) and the Wisconsin Diagnostics Breast Cancer (WDBC) in (WIlliam Wolberg, 1992). Both datasets are publicly available along with other datasets in the UCI machine learning repository for researchers. The WBC consists of 699 entries, with 10 feature columns and 1 predictive class column. A common detail of WBC is 16 missing values in the attribute number 7 “bare nuclei” which is termed as “attributes”. These gaps are shown in table 1 which is represented by ”?” . The 11th feature is called class column, out of which, two classes, are 4 representing (malignant) and 2 as (benign) with 65.5% of the total class column. Other class denoted as 34.5%.

The second dataset used is the WDBC. It contains 32 attributes and 569 instances . This dataset does not have any missing values. The diagnosis column has two classes: ‘B’ for benign and ‘M’ for malignant. The class distribution in the WDBC contains 357 instances that are benign and 212 instances that are malignant.

The confusion matrix style related the correlation heatmap is an extremely powerful and efficient way to check the highly correlated feature variables with the target or class variable. This explanation was the reason why the attributes and the target variable were so closely related. It is much important factor particularly in choosing the main and vital features in feature selection methods. The significance of independent attributes with one another is very much dependent to their positive or negative correlation score. Figure 3.1 shows the correlation heatmap for the WBC and WDBC, parts (a) and (b) respectively.



(a)



(b)

Figure 3. 1 Correlation heatmap between features and target variables for (a) WBC (b) WDBC

3.2 Dataset Exploration and Visualization:

The scatter matrix plot creates a matrix of scatterplots for pairwise relationships between features of the Wisconsin Breast Cancer dataset while omitting the "class" variable. The off-diagonal elements of the grid are scatter plots on 2D space that help in understanding the correlation between features, and the diagonal elements show Kernel Density Estimation (KDE) for each of the features. The semi-transparent red dots plot the data points, and this visualization helps reveal possible linear or non-linear relationships between features, such as the predicted relationship between "radius" and "perimeter." Hence, this visualization is favorable to identify patterns, examine the distributions of the features, and locate possible outliers as part of exploratory analysis and feature selection for future ML tasks. Fig.3.2. shows the scatter plot for the WBC dataset.

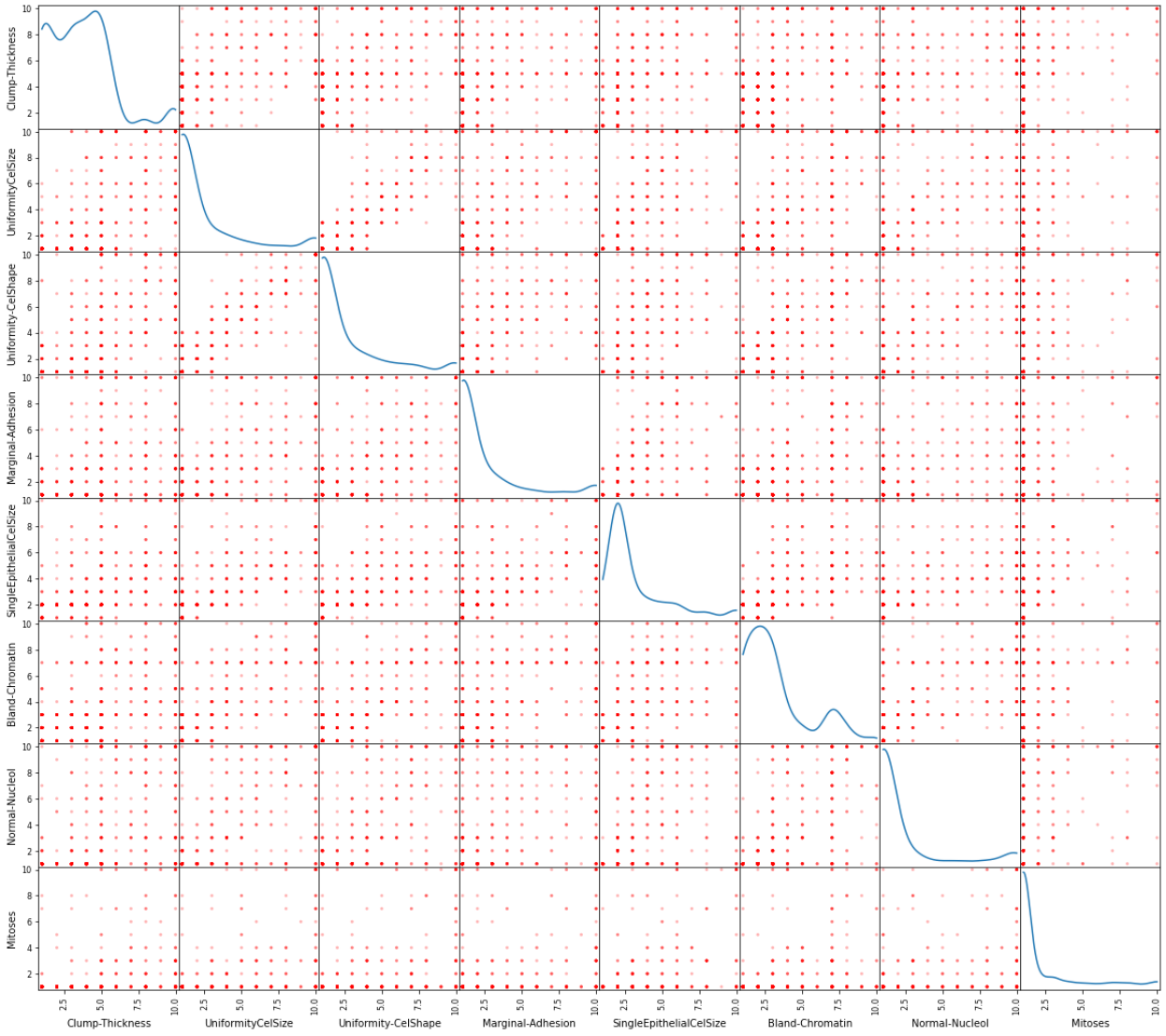


Figure 3. 2 Scatter matrix plot showing the relationships and distributions of features in the Wisconsin Breast Cancer Original dataset.

3.2.1 KDE-based Histogram visualization of the WBC input features:

The Fig.3.3. illustrates the pair plots for the Wisconsin Breast Cancer dataset with histograms and KDE plots for each feature. The diagonal cells are the histogram distribution of individual features over a KDE curve that indicates the frequency and smooth distribution of individual features. The off-diagonal cells show the relationships between pairs of features, with histograms and KDEs showing the distribution of values, and how they might correlate or be different. It gives you an idea of the skewness, modality and how spread out each feature is, therefore accounting for the properties that will be useful for its statistical nature. It is helpful to use a select number of performance metrics to understand feature behavior, identify possible outliers, and guide preparation and feature engineering for machine learning models.

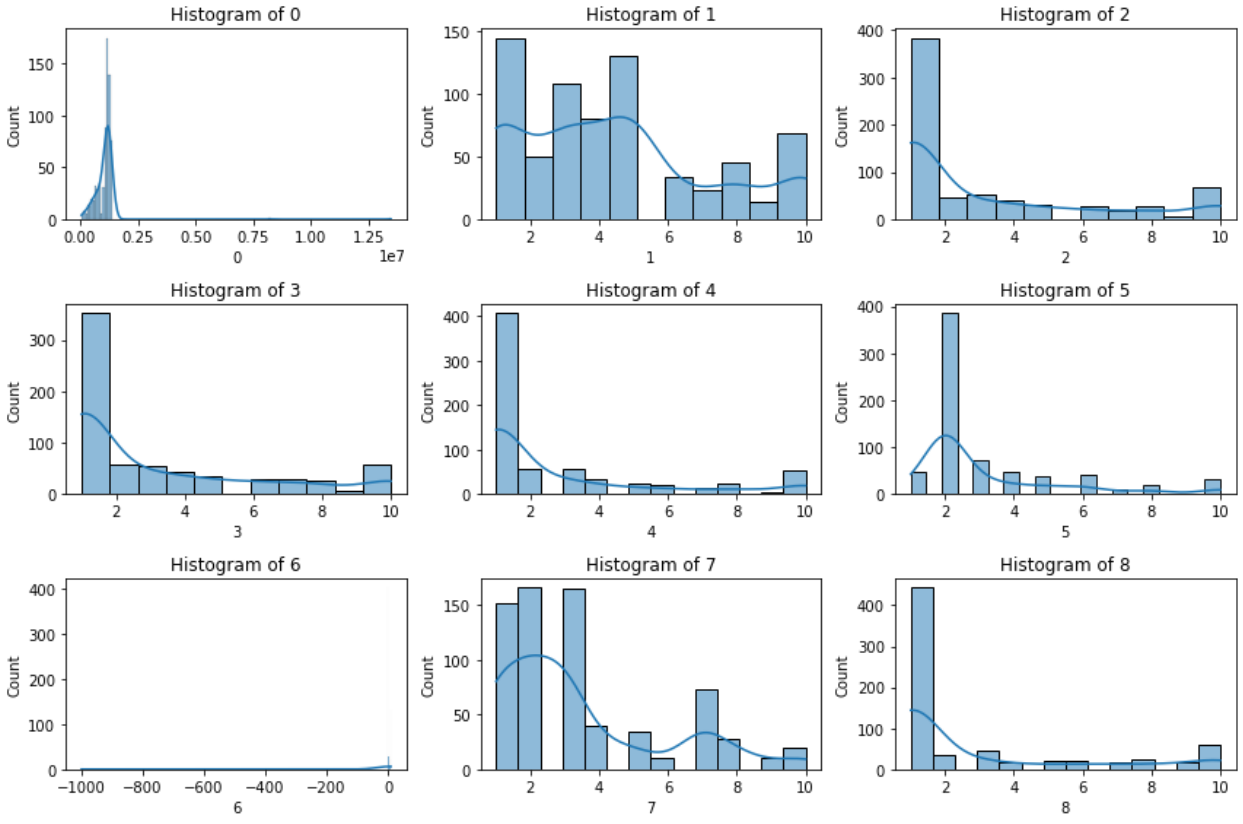


Figure 3. 3 Pairwise feature distribution visualization of the Wisconsin Breast Cancer dataset, showing histograms with Kernel Density Estimation (KDE) plots for each feature.

3.2.2 Pairwise Boxplots for the WBC dataset:

The plot displays in Fig.3.4. is a combined pairwise boxplot for the Wisconsin Breast Cancer dataset variables, where variables deeper feature distributions are plotted along the diagonal and variable pairs are represented on the off-diagonal Each boxplot shows the interquartile range (IQR) with a line indicating the median and the whiskers extend to any values within 1.5 times the IQR. Outliers are represented as single points beyond the whiskers. Such visualizations enable one to see the spread, and also, the central tendency and variability of each feature, along with any skewness, and you can spot outliers also. We can see relationships between the features from the boxplots, for example, distributions that are similar in nature, or differ from those of other features, which will help explore data and prepare it for further analysis or modeling.

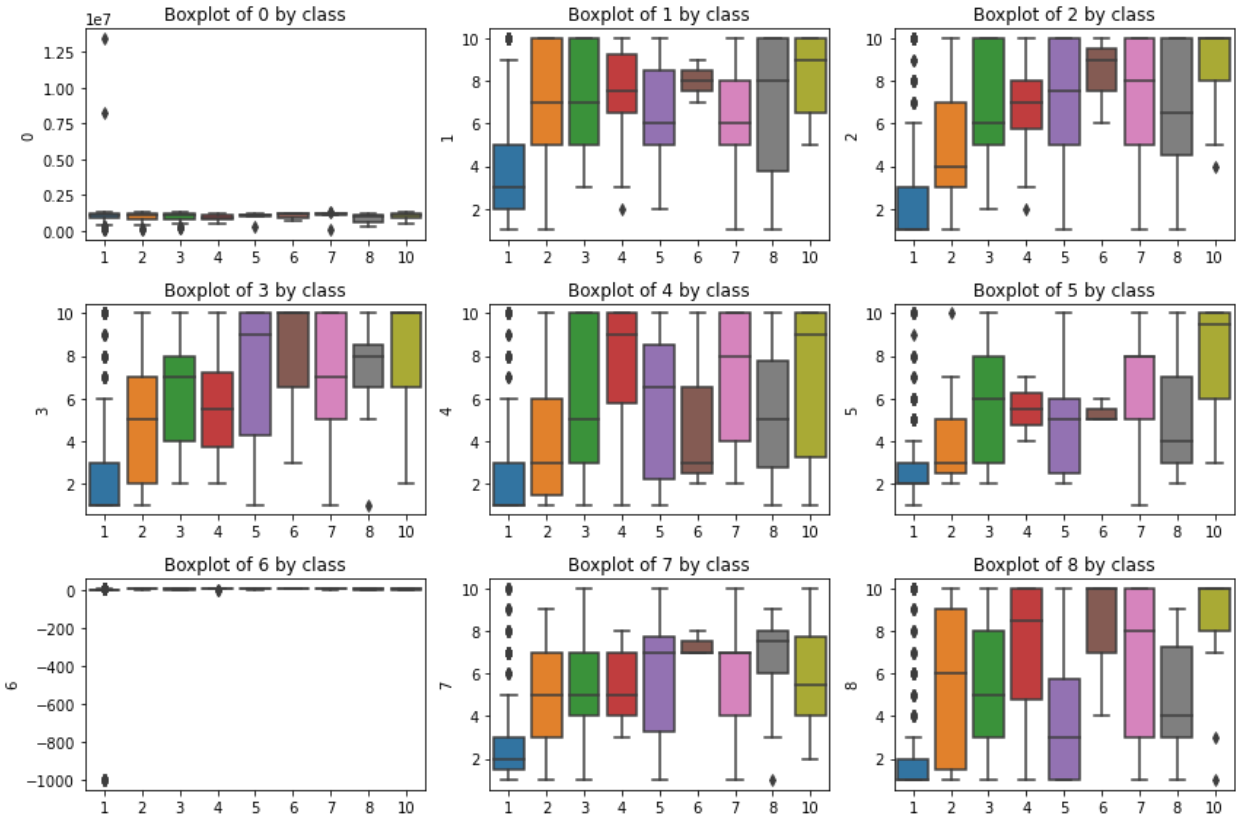


Figure 3. 4 Pairwise boxplot matrix showing the distribution, spread, and relationships between features in the Wisconsin Breast Cancer dataset.

3.2.3 Density Plots for the WDBC:

The below Fig.3.5. is showing a pair plot of Wisconsin Diagnostic Breast Cancer dataset which is used to show the spread of features for every class value of the dataset. Each small plot

(histogram) on the diagonal is depicting the univariate distribution of a single feature, rest of the plots are the scatter plot for pairs of features. The bluer the color, the wider the range of values that the distribution of each feature takes. This plot is instrumental to visualize the shape of the data, see trends and relation between the variables, and is extremely helpful to look at when we are working on classification problems (tumor = malignant/benign).

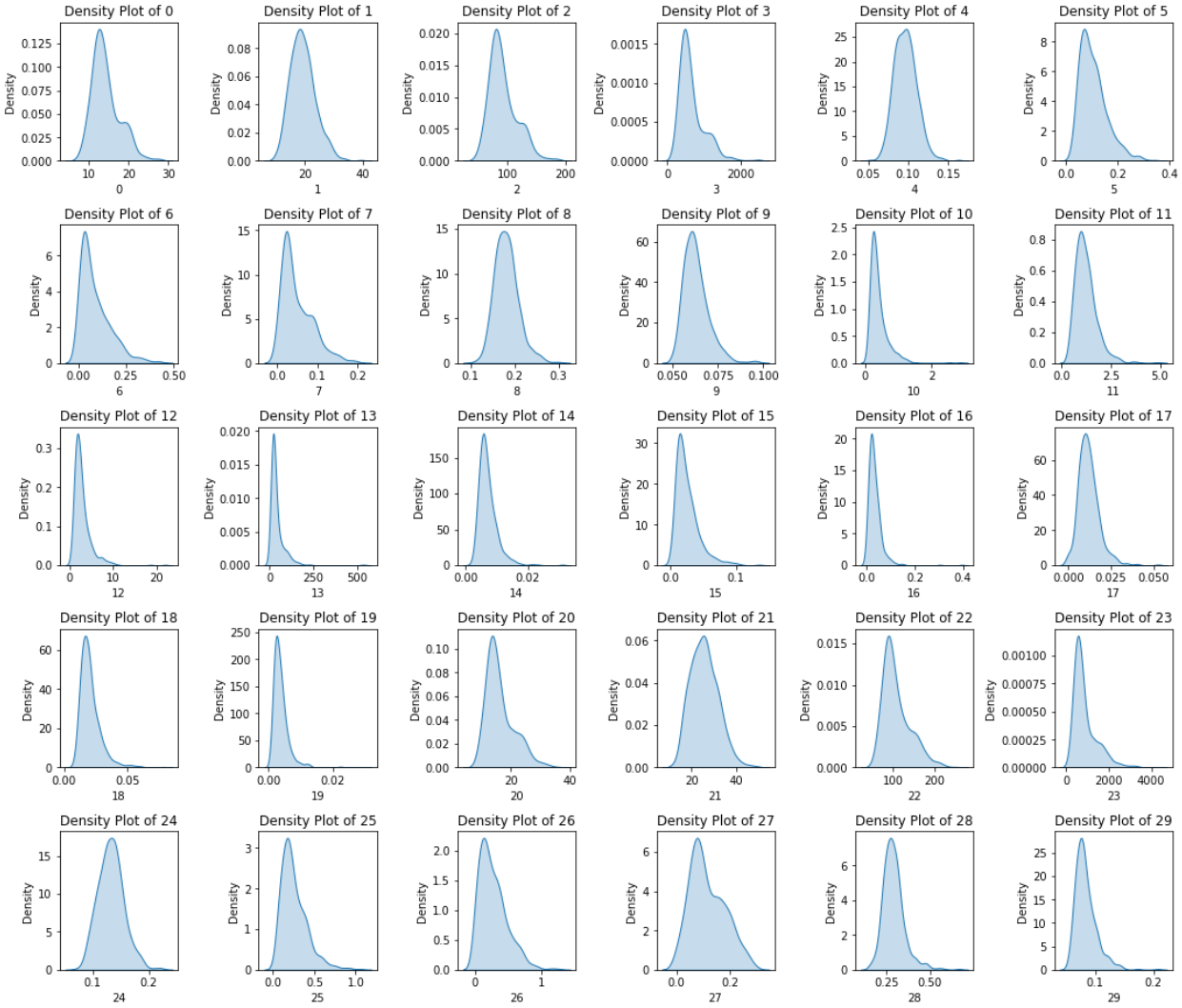


Figure 3. 5 Density Plot for Wisconsin Diagnostic Breast Cancer dataset, showing the distribution of features and pairwise relationships between variables.

3.3 Proposed Methodology:

The proposed methodology defines a strategic machine-learning pipeline that streamlines the whole data acquisition, analysis and prediction process. A pipeline is initialized by the Dataset block, which collects raw data and organizes it so that it is suitable for further inspection in a repository. This data is then cleaned and regularized in the Cleaning step, also known as pre-processing. The next process consists of dimensional reduction methods including Principal Component Analysis (PCA) and Factor Analysis. After that an important step of feature selection considers the techniques with best attributes for the classifier. Ensemble Learning then functions on the model by training it with several learning models to predict better, to let those models' outputs converge and for there to be a joint ensemble effect (depending on how many systems participated in each mix). The model's performance is finally gauged through Performance Evaluation metrics, including model accuracy, precision, recall and F1-score, in order to test its true merits. Finally, the pipeline concludes with the Predicted Output block, which makes predictions based on the data processed and presented throughout the entire process. This process has brought the data to a state fit for prediction. The Fig.3.6. shows the general block diagram of the proposed methodology.

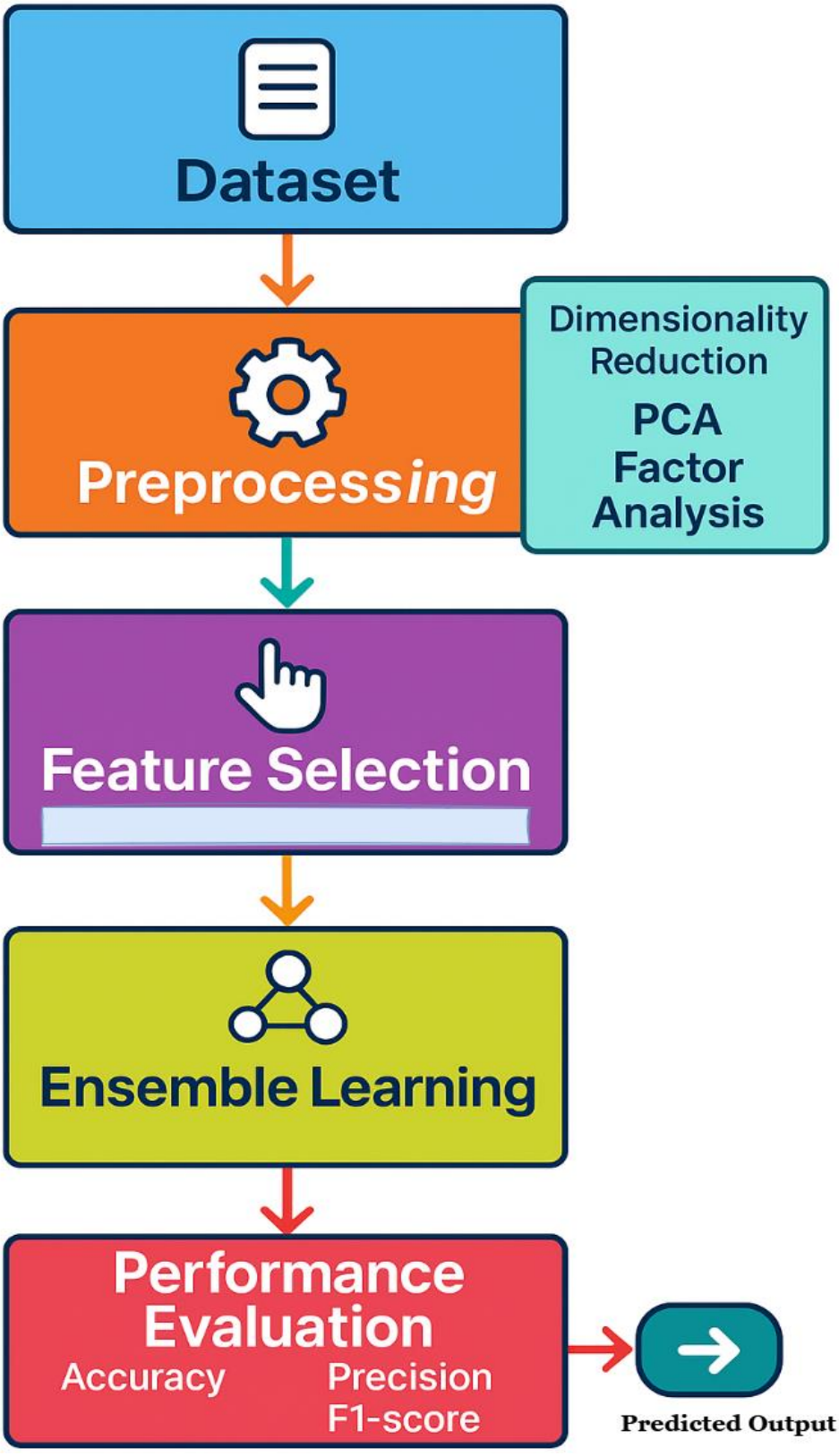


Figure 3. 6 General Block Diagram of Proposed Methodology.

3.3.1 Chi-squared (Chi2) Feature Selection:

The Chi-squared attribute selection technique is known as the statistical test based feature selection technique for interpretation of a variable dependence. Let us consider a dataset which contains one class variable and other attributes. In this technique an association is formed with the target variables and the attributes. All attributes that are considered as independent and have no affiliation with the target variables are gotten rid of from the selected attributes list. The nature of the relationship among the features and class variable indicates their significant significance and are regarded as critical features to be included in the list (Putra, Wardhani and Busman, 2019). The detailed mathematical formulation will now be provided.

$W_{TS} = Total\ \ number\ of\ instances$

$W_{+} = Total\ number\ of\ possitive\ instances\ \ \ that\ accomodate\ attribute\ W$

$W_{-} = Total\ number\ of\ negative\ instances\ that\ accomodat\ \ attribute\ W$

$W_{\mp} = Total\ number\ of\ possitive\ instances\ that\ do\ not\ have\ attribute\ W$

$W_{=} = Total\ number\ of\ negative\ that\ do\ not\ accomodate\ attribute\ W$

$W_m = \ W_{+} + W_{-} = Total\ number\ of\ instances\ that\ have\ attribute\ W$

$W_{TT} = W_{+} + W_{-} = W_{TS} - W_m = instances\ that\ do\ not\ have\ atrribute\ W$

$$W_{\hat{\mp}} = \frac{(W_{+} + W_{\mp})\ (W_{+} + W_{-})}{W_{TS}} \qquad (11)$$

$$M^2 = \sum_{t=1}^{n} \frac{(C_{v-S_m})^2}{W_{TS}} \qquad (12)$$

$M^2 = Chi - square\ test, C_V = Conformed\ value, S_m = sample\ mean\ val$ue

$$M^2 = \frac{W_{TS}[(W_+ \times W_-) - (W_- \times W_=)]^2}{(W_+ + W_\mp)\,(W_- - W_=)\,(W_+ + W_-)\,(W_\mp + W_=)} \quad (13)$$

3.3.2 Recursive Feature Elimination (RFE) feature selection:

The RFE method of feature selection works on the phenomena of eliminating the least relevant features and amplifying the most important features. The criteria for selection of features vary; some used weight vector using Linear SVC based RFE (Jeon and Oh, 2020). This exploratory study relied on feature scoring as the dominating criterion to be processed by the machine learning block for subsequent enlistment on each selected dataset. The RFE process continues iteratively until it has removed all weak features or 'scored' features and re-ranked the remaining features. Scikit learn package was used to implement RFE (Pedregosa, Weiss and Brucher, 2011). First, a feature set (FS) is taken, the features are ranked, and a model using Random Forest (RF) is built and the score of the features contained in FS is checked. These scores are then fed to the machine learning models employed in the RFE feature selection process based on the scored features. Figure 10. (a) captures the step of preparing a feature set whereas Figure 10. (b) depicts the expert judging and selection of an optimal approach from a pool of multiple strategies for efficient feature selection after accuracy comparisons with WBC and WDBC real-time datasets. Selected feature details are presented in Table 3.1.

Table 3. 1 Original and Selected Features for both WBC and WDBC datasets.

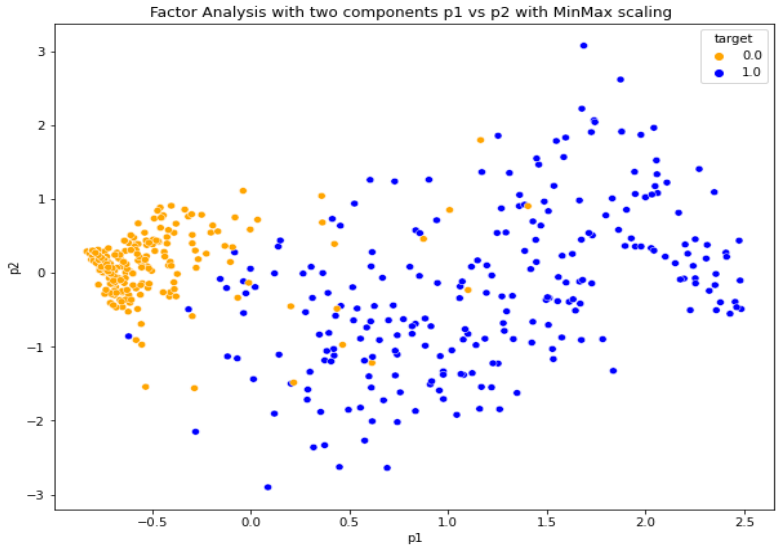
Name of Datasets	**Original features**	**Recursive Feature Elimination (RFE)**	**Chi-squared (Chi2)**	**PCA+RFE proposed**
WBC	09	07	06	05
WDBC	32	24	22	05

3.3.3 Factor Analysis (FA) Dimensionality Reduction:

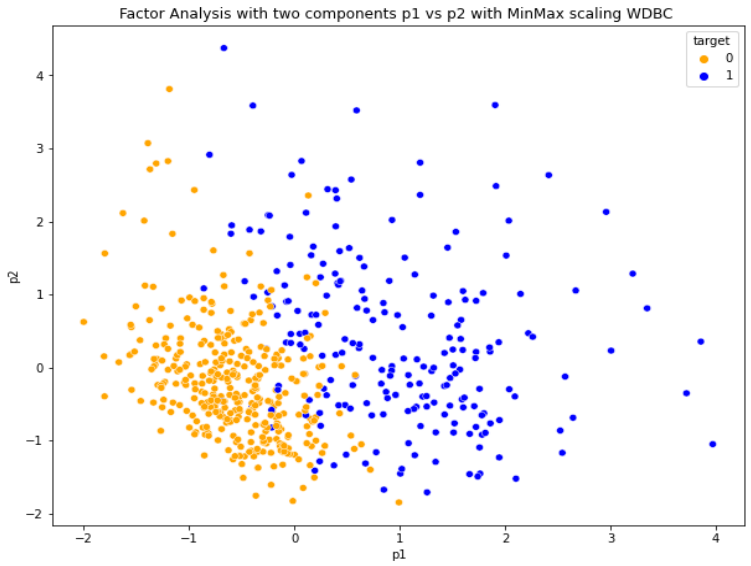
The other big data dimensionality reduction technique is Factor Analysis (FA). This strategy used observed variables to generate the factors to be used in the ordinary variance (Decoster and Hall, 1998). Some people mistook it for PCA, whereas the two techniques slightly differed as they worked on different models, and actually implemented on the same datasets yielding different

results. One of the main differences between PCA and FA is that FA is based on the true underlying factors while PCA is based on measured responses. In brief PCA should be known if you focused towards the data reduction method while FA should be used when you want to make statements about those factors that are committed to the observed responses (Weng and Young, 2017). The Figure 3.7. shows the dimension reduction for all the data points available on WBC and WDBC datasets.

The 2D factor analysis plot shows the scaled white blood cell (WBC) data, projected onto the two most important components. The scatter plot of the data points colored by two labels, orange for target 0.0 and blue for target 1.0. The separation between the two target classes can be seen in the plot of factor analysis along the x-axis, which means that factor analysis has successfully separated the two groups according to the most important features in the data. The orange class is superimposed tightly on right hand side, on the contrary, the blue class is more spread in the right hand side, which implies distinctive property between the two classes. Such clear distinction shows that our dataset can be easily applied for purposes of classification task as the principal components identified by factor analysis contain variance being required to separate both classes.



(a)



(b)

Figure 3. 7 FA based 2-dimensional scatter plots on (a) WBC & (b) WDBC dataset points

3.3.4 Principal Component Analysis (PCA) Dimensionality Reduction:

The PCA is a powerful and impressive technique, dating back a long time. It is widely used to extract features, for unsupervised clustering–and to reduce the dimensionality of data. It further has many other applications: for example decomposing corrupted signals into component waves; understanding it may be used to analyze and visualize different kinds of genome data graphically (with applications beyond our scope here); its power to perform Exploratory Data Analysis (EDA) in most cases if not all cases where we understand what we want out from an experiment or project beforehand over given input. It was first proposed in 1901 by Pearson (1901) (Salem and Hussein, 2019); independently it also became known as eigen vector analysis (EVA). The operational principle of the PCA is to convert predictor variables into principal components such that the first component has largest variance, and any variant attribute after this of equal importance is captured by a subsequent subordinate component (Ali *et al.*, 2017).

Let consider the covariance matrix denoted by '**W**' and '***l***' represents the linear combination of of the **'W'** in such a way that they don't have any correlation with each other.

- Let's suppose first principal component: $P_{C1}=a_j^T W$, Expressed as a_j such that variance (P_{C1}) $=a_j^T \sum a_j$ =maximum $|a|=1$ $a^T \sum a$:
- 2_{nd} principal component: $P_{C2}= a_2 W$, variance $(P_{C2}) = a_2^T \sum a_2$=maximum $|a|=1$; $a^T \sum a$ with

$a_j^T \sum a_2 = 0$;

- L_{th} principal component: $P_{CL} = a_L^T W$ variance $(P_{CP}) = a_L^T \sum a_L$ = maximum $|a| = 1$; $a^T \sum a$ with $a_L^T \sum a_j = 0$, J=1,…P-1

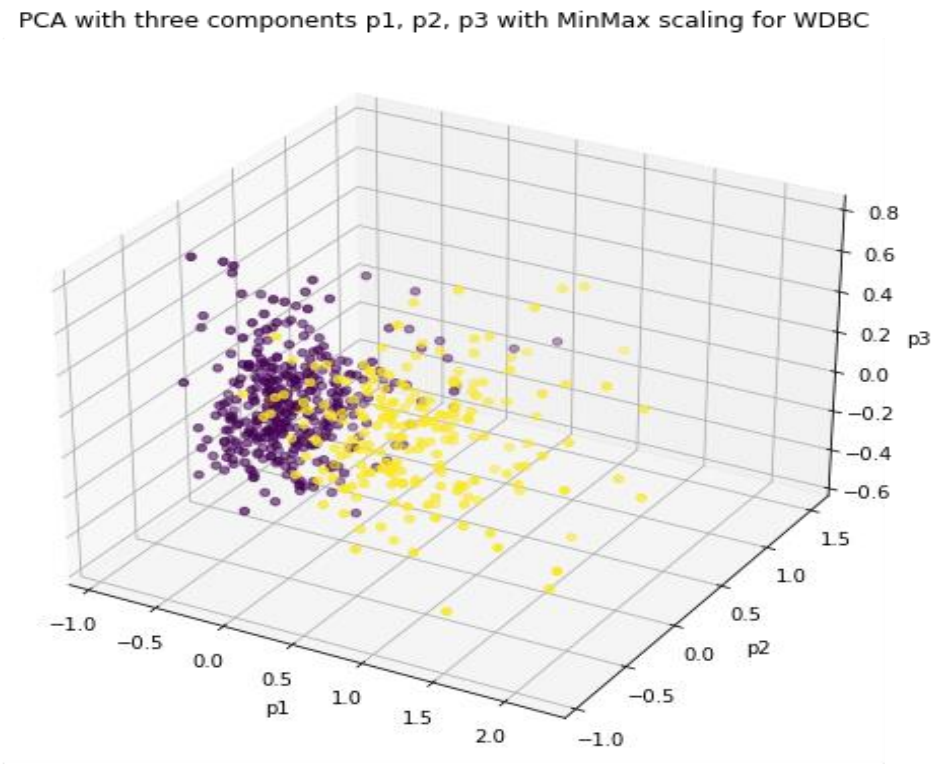
The construction of the proposition is represented as

Proposition: let (δ, k_i) for i= 1…L be the eigen structure for $\sum$, where $\delta_1 \geq \ldots \geq \delta_L \geq 0$, then I_{th} Principal component is given by $P_{ci} = K_i^T W$ where $1 \leq i \leq m$ and variance $(P_{Ci}) = K_i^T \sum k_i = \delta_i$ covariance $(P_{ci}, P_{cm}) = 0$ for $I \neq m$.

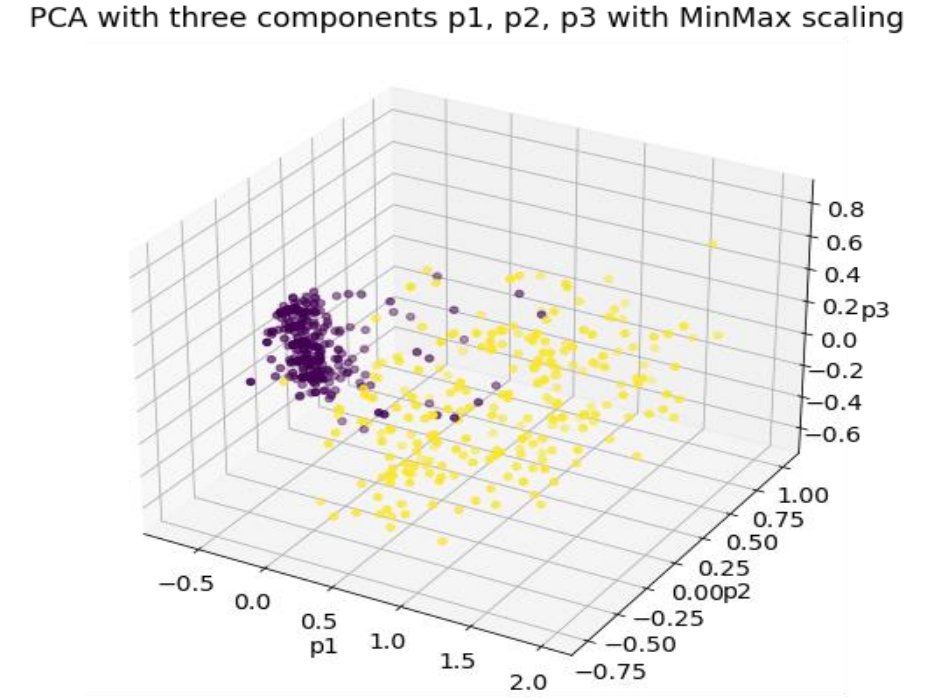
The data dimensionality for WBC and WDBC by using PCA technique is illustrated in Figure 3.8.

This 3D graph in Fig. 3.8(a) depicts a PCA analysis transformation of the Wisconsin Diagnostic Breast Cancer dataset, looking at the 2 classes of tumors: malignant (purple) and benign (yellow). The figure indicates that the first three principal components (p1, p2, and p3) represent the highest variance in the data, therefore, contributing to dimension reduction. The malignant and benign tumor clusters are distinctly separated from one another meaning that PCA has successfully separated the classes for these data points. Malignant tumors are concentrated more in the central part along the p1 axis, compared to benign tumors, turning outly distributed over the plot. This gap indicates the possibility to use PCA with classifier in order to be able to successfully distinguish malignant from benign masses according to their shape.

The Fig.3.8(b) represents a PCA (Principal Component Analysis) transformation applied to the Wisconsin Breast Cancer (WBC) dataset, visualizing two classes: malignant tumors (represented by purple points) and benign tumors (represented by yellow points). Similar to the previous plot, the axes p1, p2, and p3 correspond to the first three principal components, which capture the most significant variance in the dataset after dimensionality reduction. The plot shows a clear separation between the two tumor types, with the malignant tumors clustering together in one region of the 3D space, while the benign tumors are spread across a different region. The points for malignant tumors are concentrated around the lower left of the plot along the p1 axis, while benign tumor points occupy a separate area, showing that PCA has effectively reduced the dimensions and highlighted the differences between the two classes. This indicates that dimensionality reduction has successfully captured the variance needed to distinguish between malignant and benign tumors in the WBC dataset.


PCA with three components p1, p2, p3 with MinMax scaling for WDBC
p1
p2
p3

(a)


PCA with three components p1, p2, p3 with MinMax scaling
p1
p2
p3

(b)

Figure 3. 8 PCA plots of the WDBC (a) and WBC (b) datasets, showing distinct separations between malignant (purple) and benign (yellow) tumors in 3D space.

3.3.5 RANDOM FOREST (RF):

The Random Forest classifier is an ensemble learning method which generates many decision trees during trainings and reports the mode of the classes (for classification) or the mean prediction (for regression) of the individual trees. Specifically, in our approaches, each tree in the forest is trained on a bootstrapped sample of the dataset and at each split in the tree a different random set of features is used to form the split, leading to reduced overfitting and better generalization (Breiman, 2001).

3.3.6. Performance Evaluation Metrics:

Precision:

The statement elucidates the accurate identification of expected instances, particularly those classified as True Positives, within the entirety of Positive predictions, encompassing both True and False Positives. The mathematical representation of the concept under discussion is as follows:

$$Precision_{(pr)} = \frac{T_P}{T_P + F_P} \times 100\% \qquad (6)$$

Accuracy (AC):

The metric accuracy is the percentage of correctly classified examples compared to the total number of correct and incorrectly classified examples.. The mathematical expression can be expressed as follows:

$$Accuracy_{(AC)} = \frac{T_P + T_N}{Total\ number\ of\ samples} \times 100\% \qquad (7)$$

Recall (RE):

After analyzing the model against False Negatives prevention, it's important to go deep into a full analysis of the model. Recall, which is Trues Positives/(False Positives + True Positives), is a measure to measure what proportion of the positive examples are correctly classified. Mathematically, recall is the amount of true positives (hit) divided by the number of true positives and the number of false negatives (miss) (namely true positives plus false negatives):

$$\mathrm{Recall}_{(\mathrm{RE})} = \frac{\mathrm{T_P}}{\mathrm{T_P + F}_N} \times 100\% \qquad (8)$$

Chapter 4 Experimental Results

In this part of the thesis the detailed assessment and analysis of our developed and proposed techniques. The data comprise the three main parts of the experimental studies. The first part describes feature selection methods, the second one presents the results of attributes dimensionality reduction methods, and the last part covers the hybrid combination of dimensionality reduction along with fattribute selection. A thorough comparison has been developed between adopted methods and our best ranked performance method is also compared in accuracy for already existing work on same datasets. All coding for the experimental work in this study is implemented with python.

4.1 Experimental Settings:

The experimental work involves software and hardware aspects of the system under use. The pc's hardware is a DESKTOP-SYSTEM that runs on an Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz. The RAM of the system is 16-gigabytes and the Graphics card installed in the system is Nvidia GeForce, GTX 1050 Ti with 4-gigabytes VRAM. Though the software component of the system consists of a few widely used machine learning libraries namely, NumPy (Harris *et al.*, 2020), Pandas (The pandas development team, 2020), Scikit-learn (Pedregosa, Weiss and Brucher, 2011), Matplotlib (Thiruvathukal and Hunter, 2007) and Seaborn (Waskom, 2021).

4.2 Results with Factor Analysis (FA) on WBC and WDBC:

The Confusion matrix in Fig.4.1. (a) of the WBC dataset obtained using Random Forest as a classifier and Factor Analysis as the extractor is performing well with 86 True Positives (cases correctly identified as having cancer), 3 False Positives (cases that were classified as having cancer but didn’t), 0 False Negatives (cases that were classified as not having cancer but does), and 51 True Negatives (cases that were correctly classified as not having cancer). Accuracy of the model is 97.86% that means classifier has correctly predicted whether the employee will leave in almost 98% of the cases. The precision is 96.7%, so that 96.7% of the predicted malignant cases are actually malignant. The 100% recall suggests that no malignant case missed by the model (no false negatives). The F1 score, which takes into account both precision and recall, is 98.3%, indicating that the model is well capable to classify both malignant and benign tumors effectively even though a feature selection might still be useful for improving model interpretability.

The confusion matrix on the WDBC dataset, for the combination with Random Forest and Factor Analysis is illustrated in Fig.4.1. (b) that presents 63 true positives (malignant tumors that were correctly classified), 2 false positives (benign tumor that were incorrectly classified as malignant), 5 false negatives (malignant tumors that were incorrectly classified as benign), and 44 true negatives (benign tumors that were correctly classified). The estimate of the accuracy of the model is 93.86% meaning that the model predicted correctly almost 94 cases. Precision is 96.9%, which implies that in 96.9% cases model predicted malignant, it was actually malignant and recall is 92.6% that indicate that model has been successful in identifying 92.6 percent of actual malignant tumor. The F1 score is 94.7%, which balances precision and recall, indicating that the algorithm can identify the malignant and benign tumors much better, although higher recall is required to minimize the occurrence of false negatives.

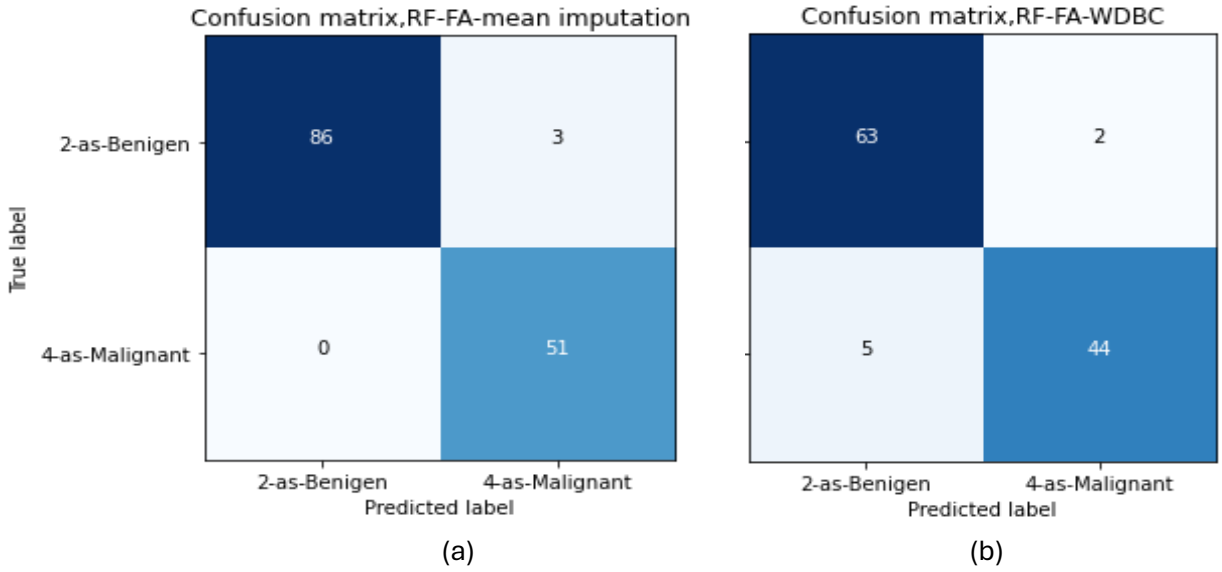


Figure 4. 1 Confusion Matrix of RF classifier on (a) WBC and (b) WDBC with FA.

4.3 Results with Principal Component Analysis (PCA) on the WBC and WDBC:

The confusion matrix in a Fig.4.2. (a) is the results of applying a classifier to the dataset Wisconsin Diagnostic Breast Cancer (WBC) dataset with the dimensionality reduction technique Principal Component Analysis (PCA). The matrix shows 88 true positives (correctly predicted malignant), 5 false positives (false predicted as malignant when benign), 2 false negatives (false predicted as benign when malignant), and 45 true negatives (correctly predicted benign). According to these values, the classifier's accuracy is 95% of the predictions labeled as 'Excellent' were true. Note

that precision is 94.6%, so 94.6% of the positive predictions are malignant. The recall (sensitivity) value is 97.8%, which indicates the percentage of patients with the actual malignant condition being predicted by the classifier, i.e. 97.8% of the malignant tumor cases are correctly recalled. Finally, the the F1 score, which consider both precision and recall, is 96.2%, indicating that the classifier performs well overall in term of distinguishing between the malignant and benign tumors.

For the WDBC using the Random Forest classifier with PCA for dimensionality reduction, the confusion matrix in Fig.4.2.(b). shows (73 true positives(malignant missed), 2 false positives (Benign marked as malignant) 5 false negatives (malignant marked as benign), 34 true negatives(benign missed). The accuracy of the model is 93.7 percent meaning it has predicted 93.9 percent the cases correctly. Precision is equal to 97.3%; that is, 97.3% of those that the model predicted as malignant, were actually malignant, and recall is equal to 93.6%; that is, 93.6% of actual malignant tumors were predicted by the model. The F1 score, a compromise between precision and recall is 95.4%, demonstrating coherent efficiency of the model with room for improvement regarding false negative errors.

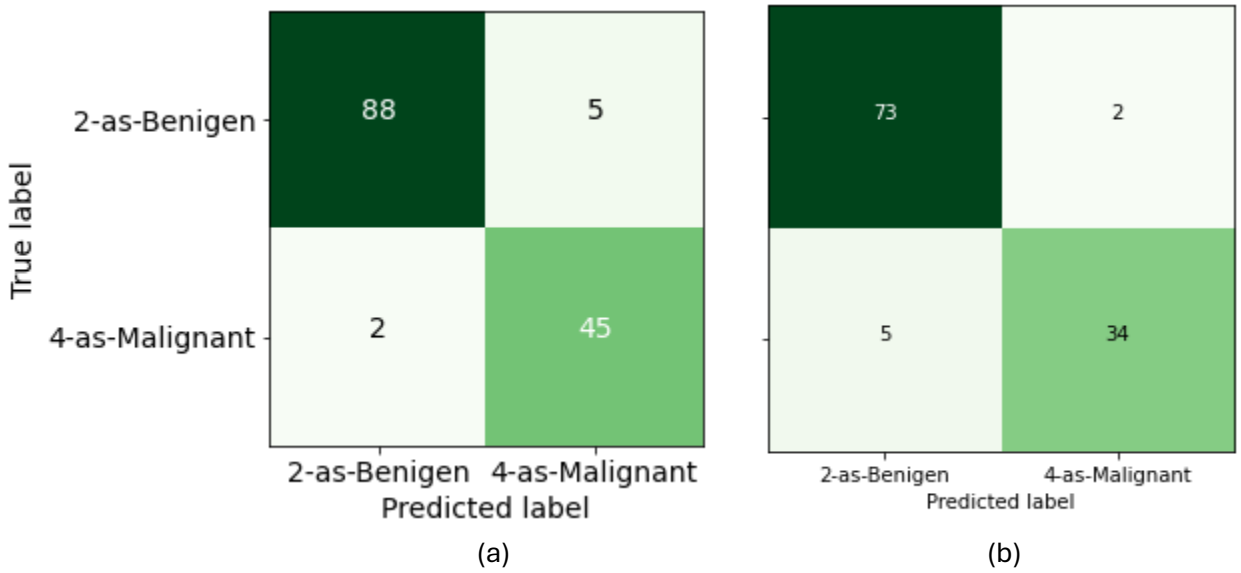


Figure 4. 2 Confusion Matrix of RF classifier on (a) WBC and (b) WDBC with PCA

4.4 Results with Chi square (Chi2) feature selection on the WBC and WDBC datasets:

The confusion matrix in 4.3(a) implemented on WBC dataset where feature selection was applied using Chi-Square feature selection and 6 features were selected for this work. The confusion matrix indicates that the classifier accurately classified 91 malignant tumors (True Positives) and 42 benign tumors (True Negatives). It had 4 false predictions (repeated the same process of a cell above), where it classified benign tumors as malignant (False-Positives) and 3 false predictions, classified malignant tumors as benign (False-Negatives). Continuing this, the accuracy of the model comes out to be 95 % (predicting correctly 95 % of the cases.) Precision is 95.8%, meaning 95.8% of detected cases are actually CA. The recall, or sensitivity, is 96.7%, i.e. the classifier has been able to detect 96.7% of the actual malignant tumors. The F1 score which weights between precision and recall is 96.2% indicating very good performance of differentiation between the malignant and benign tumors. The model overall does very well (high precision is reached, false positive and false negatives are very low) thanks to the feature selection.

The confusion matrix shown in the Fig 4.3(b) has been obtained from the performance of a Random Forest classifier to the WDBC dataset by using Chi-Square feature selection. In this work the model accurately predicted 74 malignant tumours (True Positives) and 36 benign tumours (True Negatives) It was 1 times misclassified as benign (False Positive) and 3 benign cases were misclassified as malignant (False Negatives). The 96.4% accuracy indicates that the model was able to accurately predict 96.4% of all cases. Precision, representing the capability of a model to predict the positive class, equals 98.7%, demonstrating that almost all those predicted as malignant were truly malignant. The recall, or sensitivity, is 96.1%, indicating that the classifier was able to accurately detect as malignant 96.1% of all actual malignant tumors. The F1 score is 97.4%, thus also a good performance in discriminating the malignant and benign tumors. Overall, the model has a good accuracy and stability (few false-positives and few false-negatives).Total number of the selected attributes are 22 for this work.

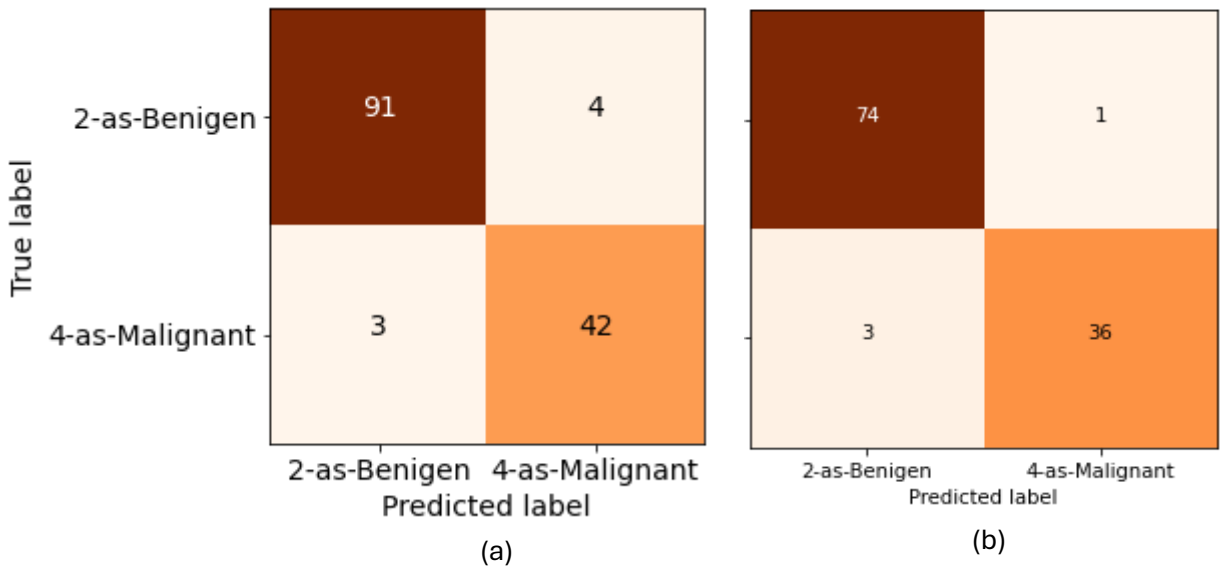


Figure 4. 3 Confusion Matrix of RF classifier on (a) WBC and (b) WDBC with Chi2 feature selection

4.5 Results with RFE feature selection on the WBC and WDBC datasets:

The confusion matrix in Fig 4.4(a) about the WBC dataset which was trained (tested) with the use of the RFE feature selection method with the selection of 7 features, it can be seen that the classifier shows a strong performance. Of the total predictions, 93 negative results representing malignant tumors were correctly predicted as true positives and 44 negative- representing benign tumors were accurately predicted as true negatives. The system had 2 false positives (benign tumors misclassified as malignant) and 1 false negative (malignant tumors that had erroneously been classified as benign). Now this is an accuracy of 97.86% which means the classifier predicts accurately in 97.8% of the cases. This accuracy indicates that most predicted malignant cases were true positives, the 98.9% recall shows excellent performance of the model for detecting actual malignant tumors. 98.3% as F1 score, which is an indicator of overall performance of classifier in separating malignant and benign tumors with high accuracy and few errors.

The Fig 4.4(b) with RF classifier run on the WDBC dataset using RFE feature selection to select 24 features. The matrix reveals that 70 malignant tumors (TP) were correctly classified by the model, along with 40 benign ones (TN). It was mistaken 1 time to say that benign tumor is malignant (False Positive), and 3 times to say that malignant tumor is benign (False Negative). The accuracy of the model is 96.4%, which tells us that our model is able to correctly predict

96.4% of the times. Precision, which represents the correctness of positive prediction, is 98.6%, indicating that 98.6% of predicted malignant cases were cancerous. The value for Recall (Sensitivity) is 95.8% indicating that 95.8% of the actual malignant tumors have been correctly categorized as malignant tumors by the classifier. The F1 score is 97.2%, indicating a high differentiation capability of malignant vs. benign tumor label. Because of the feature selection, these models have high prediction accuracy and low false positive and negative rates.

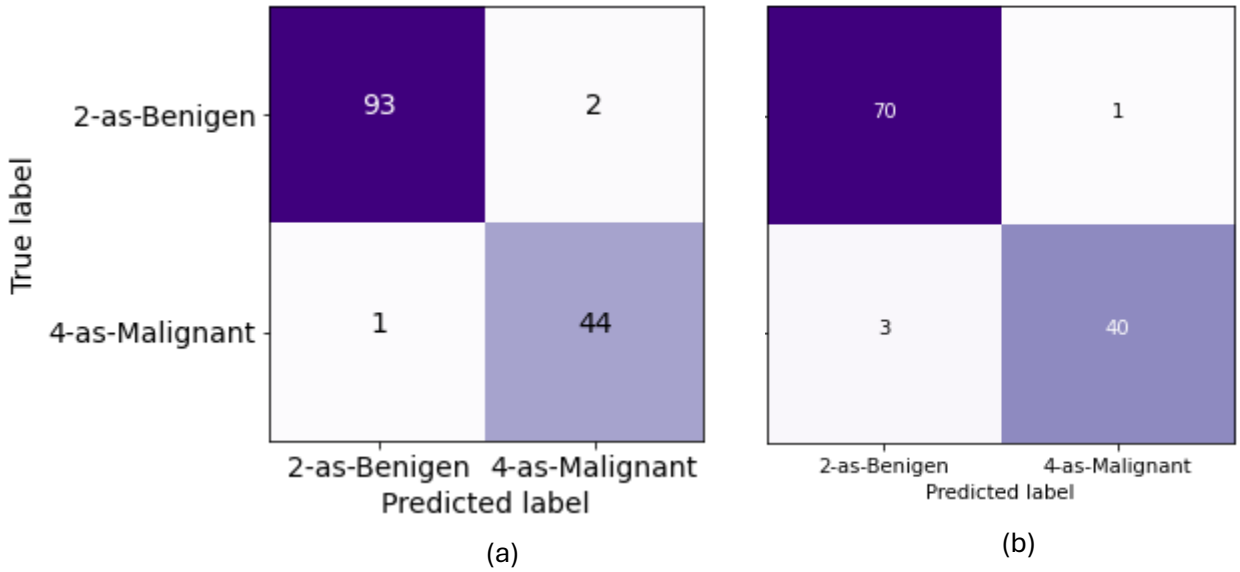


Figure 4. 4 Confusion Matrix of RF classifier on (a) WBC and (b) WDBC with RFE feature selection

4.6 Results with Hybrid Combination of RFE feature selection along with PCA based dimensionality reduction on the WBC and WDBC datasets:

Figure 4.5(a) below illustrates the confusion matrices for the proposed hybrid approach on WDBC dataset. The performance of a classifier on WDBC combining RFE for feature selection an PCA for dimensionality reduction. In this hybrid model, 5 features were chosen and the explained variance ratios of the components are 0.44, 0.19, 0.09, 0.07, 0.05 (i.e., the significance of each composing component in explaining the total variance). The model was cross-validated using a 5-fold cross-validation, and the optimal hyperparameters of the Random Forest classifier were determined as follows with a maximum depth of : 10, minimum samples leaf are: 2, minimum

samples split values is five, and total number of estimators are: 100. The classifier correctly classified 73 malignant (True Positives) and 39 benign (True Negatives). It produced no false positives (0) and only 2 false negatives (malignant tumors predicted to be benign). The accuracy of the model 98.25% achieved, highest among all methods tried, an efficient performance with least amount of errors.

The confusion matrix shows the performance of a classifier (WBC) dataset integrating Recursive Feature Elimination (RFE), for feature selection, and Principle Component Analysis (PCA), for dimension reduction. For this hybrid model, only 5 features were chosen, and the explained variance ratios for the components are 0.65, 0.09, 0.06, 0.05, and 0.04, which measure how much of the variability in the original features is accounted for by the current principal component. The model was optimized using 5-fold cross-validation and the optimal hyperparameters for the Random Forest classifier were: max_depth = 10, min_samples_leaf = 2, min_samples_split = 10, and n_estimators = 100. The classifier to identify 84 (True Positives) malignant tumors and 54 (True Negatives) benign, had only 1 false-positive (benign tumor), and 1 false-negative (malignant tumor). The accuracy achieved by this model is the highest in all the models tested - 98.57%, demonstrating the effectiveness of combining RFE and PCA with a well-tuned Random Forest classifier.

The Table 4.1. presents the results comparing different Random Forest (RF) methods and feature selection methods on the WBC and WDBC datasets. RF+PCA+RFE achieves the highest Accuracy (98.6% for WBC, 98.2% for WDBC), Precision (98.8% for WBC, 100% for WDBC), Recall (98.8% for WBC, 97.3% for WDBC) and F1-score (98.8% for WBC, 98.6% for WDBC), however, it also has the lowest Training Time (0.37 seconds for WBC, 0.21 seconds for WDBC) and Prediction Time (0.018 seconds for WBC, 0.011 seconds for WDBC). Other methods as RF+FA, RF+RFE also have good performance and approximate, but do not outperform the proposed RF+PCA+RFE in terms of accuracy and efficiency, and computational cost.

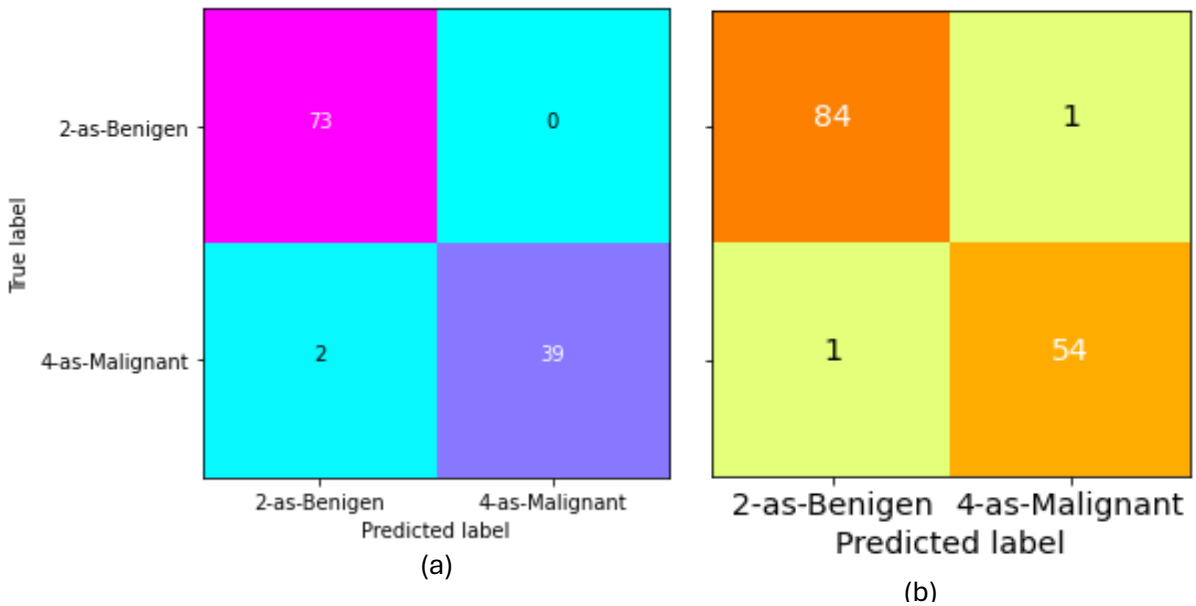


Figure 4. 5 Confusion Matrix of RF classifier on (a) WDBC and (b) WBC with Hybrid approach of RFE feature selection with PCA dimensionality reduction and RF (Random Forest).

Table 4. 1 Performance evaluation comparison of implemented techniques on WBC & WDBC.

Performance Evaluation Metrics for (WBC)						
Implemented Technique	Accuracy %	Precision %	Recall %	F1-score %	Training Time (secs)	Prediction Time (secs)
RF+FA	97.8	96.7	100	98.3	0.53	0.06
RF+PCA	95	94.6	97.8	96.2	0.50	0.050
RF+Chi2	95	95.79	96.7	96.24	0.67	0.040
RF+RFE	97.8	97.8	98.9	98.3	0.48	0.044
RF+PCA+RFE	**98.6**	**98.8**	**98.8**	**98.8**	**0.37**	**0.018**
Performance Evaluation Metrics for (WDBC)						
RF+FA	93.86	96.92	92.6	94.7	0.17	0.015
RF+PCA	93.7	97.33	93.6	95.4	0.26	0.02
RF+Chi2	96.49	98.67	96.1	97.37	0.22	0.016
RF+RFE	96.49	98.59	95.89	97.22	0.23	0.021
RF+PCA+RFE	**98.2**	**100**	**97.3**	**98.6**	**0.21**	**0.011**

4.7 Comparison with existing studies:

The Table 4.2. shows a detailed comparative analysis of this study with already existing work. The performance comparison of different methods over two data-set (WBC and WDBC), varying 97.30% to 98.57% accuracy %(age), is given in Table. The maximum accuracy is attained by the PCA+RFE+RF method in this work, which obtains 98.57% and 98.24% on the WBC dataset and WDBC dataset, respectively, higher than other methods. The Nested Ensemble method (Abdar et al., 2020) is a close runner up in terms of accuracy (attaining 98.07% on WDBC) whilst SVM+AR (Ed-daoudy and Maalmi, 2020) and BN+RBF (Jabbar, 2021) has also achieved lower accuracy at 98.00% and 97.42%, respectively. Overall, PCA+RFE+RF the proposed approach is the best method on these datasets, especially on the WBC dataset and the other methods performed close but lower then our proposed approach.

Table 4. 2 Comparison of this study best results with others existing work on both dataset

References	Methodologies	Accuracy%	Dataset
(Ed-daoudy and Maalmi, 2020)	SVM + AR	98.00	WBC
(Morkonda Gunasekaran and Dhandayudam, 2021)	MFU	97.30	WDBC
(Abdar *et al.*, 2020)	Nested Ensemble	98.07	WDBC
(Jabbar, 2021)	BN+RBF	97.42	WBC
This study	PCA+RFE+RF	98.57 98.24	WBC WDBC

Chapter 5 Conclusion

This thesis examined the combined effect of RFE and Chi2 FS methods with two dimensionality reduction methods (PCA and FA) on the Wisconsin Breast Cancer (WBC) and Wisconsin Diagnostic Breast Cancer (WDBC) datasets. The objective was to find the best pair of feature selection and dimensionality reduction methods, to enhance the performance of breast cancer detection models, in the context of Random Forest classifier.

On WBC dataset, other feature selection algorithms such as RFE and Chi2 were experimented and two dimensionality reduction methods PCA and FA were also tested individually. RFE was strong in feature selection, meanwhile Chi2 also gave us a sense of the statistical significance of each feature. PCA was employed to lower the data dimensionality of a new measurement by maintaining most of the variance and FA was attentive to the common variables which contribute to feature correlation. All these individual methods yielded acceptable results, but the hybrid approach incorporating RFE for feature selection and PCA for dimensionality reduction produced the best result (accuracy = 98.25%).

For the WDBC dataset, analogous procedures were performed using RFE and Chi2 for feature selection, and PCA and FA for dimensionality reduction. Having RFE selected the important features which has more weighted importance and Chi2 would focus on the meaningfulness of features via our statistical testing. The noise and the complexity of the dataset were then reduced by using PCA and FA for dimensionality reduction. However, the hybrid-model (RFE-PCA) was superior to all single methods, getting 98.57% accuracy, which was the best results among the tested methods.

In this dissertation we employed a hybrid strategy for feature selection using RFE, and dimensionality reduction with PCA applied to the Wisconsin Diagnostic Breast Cancer (WDBC) dataset using a Random Forest classifier. Among various machine learning datasets the WDBC is a very popular dataset representing the records and features of a group of cancer patients through features from digitized images of fine needle aspirates of breast mass. Such features are textural, smoothness, compactness, and symmetry of the cell nuclei which are crucial in differentiating the malignant and benign tumors. Hybrid feature selection and dimensionality reduction methods chose only 5 features, and significantly reduced the dimension of the dataset meanwhile

maintaining excellent classification ability. Using this model, the classifier obtained an impressive accuracy of 98.57% (highest of all methods tested). The model achieved good performance with few errors by detecting 84 malignant tumors and 54 benign tumors, which demonstrated the effectiveness of employing RFE and PCA integrated with Random Forest. Similarly a hybridization framework incorporating Recursive Feature Elimination (RFE) based feature selection and Principal Component Analysis (PCA) based dimensionality reduction had been implemented on Wisconsin Breast Cancer (WBC) dataset with a Random Forest classifier. Five key features were determined using RFE+PCA hybrid combination, that will reduce the feature space dimensionality without sacrificing the classification accuracy. The implemented classifier attained a very high accuracy at 98.25%, which was the highest of all methods applied. It had a relatively good performance in a small number of errors, that is, a correct detection of 73 out of 135 and 39 out of 259 malignant and benign tumors, respectively, which showed the improvement of Random Forest performance by RFE and PCA.

Although the hybrid method using RFE and PCA yielded the best results, other combinations of feature selection technique, for example L1 regularization and t-SNE, would be also investigated and tested by future work to check whether the accuracy can be more improved. Exploring other advanced classifiers, such as support vector machines (SVMs) or deep learning algorithms, can also yield more stable results, especially with complex or noisy datasets. Moreover, further expanding the datasets by covering more diverse cases or by including some clinical information (as genetic information) may assist the models in generalizing across different populations. Finally, applying this model in a clinic for real-time prediction is necessary to assess its feasibility and effect in assisting early diagnosis of breast cancer in future study.

References

Abdar, M. *et al.* (2020) 'A new nested ensemble technique for automated diagnosis of breast cancer', *Pattern Recognition Letters*, 132, pp. 123–131. Available at: https://doi.org/10.1016/j.patrec.2018.11.004.

Abdel-Ilah, and H.Š. (2017) 'Using Machine Learning Tool inClassification of Breast Cancer', in *CMBEBIH 2017, IFMBE Proceedings 62*.

Abdel-zaher, A.M. and Eldeib, A.M. (2015) 'Breast Cancer Classification Using Deep Belief Networks', *Expert Systems With Applications* [Preprint]. Available at: https://doi.org/10.1016/j.eswa.2015.10.015.

Ali, M.U. *et al.* (2017) 'Using PCA and Factor Analysis for Dimensionality Reduction of Bio-informatics Data', *International Journal of Advanced Computer Science and Applications*, 8(5), pp. 415–426.

Asri, H. *et al.* (2016) 'Using Machine Learning Algorithms for Breast Cancer Risk Prediction and Diagnosis', in *Procedia Computer Science*. Elsevier Masson SAS, pp. 1064–1069. Available at: https://doi.org/10.1016/j.procs.2016.04.224.

Bihis, M. and Roychowdhury, S. (2015) 'A generalized flow for multi-class and binary classification tasks: An Azure ML approach', in *Proceedings - 2015 IEEE International Conference on Big Data, IEEE Big Data 2015*, pp. 1728–1737. Available at: https://doi.org/10.1109/BigData.2015.7363944.

Breast cancer statistics | Cancer Research UK (2020). Available at: https://www.cancerresearchuk.org/health-professional/cancer-statistics/statistics-by-cancer-type/breast-cancer (Accessed: 22 April 2025).

Breast Cancer Wisconsin (Diagnostic) - UCI Machine Learning Repository (no date). Available at: https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic (Accessed: 26 April 2025).

Breiman, L.E.O. (2001) 'Random Forests', *Machine Learning*, pp. 5–32.

Cancer Facts & Figures 2020 | American Cancer Society (2020). Available at: https://www.cancer.org/research/cancer-facts-statistics/all-cancer-facts-figures/cancer-facts-figures-2020.html (Accessed: 22 April 2025).

Chen, H.L. *et al.* (2011) 'A support vector machine classifier with rough set-based feature selection for breast cancer diagnosis', *Expert Systems with Applications*, 38(7), pp. 9014–9022. Available at: https://doi.org/10.1016/j.eswa.2011.01.120.

Decoster, J. and Hall, G.P. (1998) 'Overview of Factor Analysis', *In Practice*, 37(2), p. 141. Available at: https://doi.org/10.2307/2685875.

Ed-daoudy, A. and Maalmi, K. (2020) 'Breast cancer classification with reduced feature set using association rules and support vector machine', *Network Modeling Analysis in Health Informatics and Bioinformatics*, 9(1), pp. 1–10. Available at: https://doi.org/10.1007/s13721-020-00237-8.

Fletcher Jenna (2024) *Breast cancer symptoms: Early signs, pictures, and more*. Available at: https://www.medicalnewstoday.com/articles/327488 (Accessed: 22 April 2025).

Harris, C.R. *et al.* (2020) 'Array programming with NumPy', *Nature*, 585(February), pp. 357–362. Available at: https://doi.org/10.1038/s41586-020-2649-2.

Helwan, A., Idoko, J.B. and Abiyev, R.H. (2017) 'Machine learning techniques for classification of breast tissue', in *Procedia Computer Science*. Elsevier B.V., pp. 402–410. Available at: https://doi.org/10.1016/j.procs.2017.11.256.

Jabbar, M.A. (2021) 'Breast cancer data classification using ensemble machine learning', *Engineering and Applied Science Research*, 48(1), pp. 65–72. Available at: https://doi.org/10.14456/easr.2021.8.

Jeon, H. and Oh, S. (2020) 'applied sciences Hybrid-Recursive Feature Elimination for Efficient Feature Selection', *Applied Sciences (Switzerland)*, pp. 1–8.

Juneja, K. and Rana, C. (2018) 'An improved weighted decision tree approach for breast cancer prediction', *International Journal of Information Technology*, pp. 1–8. Available at: https://doi.org/10.1007/s41870-018-0184-2.

Khandezamin, Z., Naderan, M. and Rashti, M.J. (2020) 'Detection and classification of breast

cancer using logistic regression feature selection and GMDH classifier', *Journal of Biomedical Informatics*, 111(October), p. 103591. Available at: https://doi.org/10.1016/j.jbi.2020.103591.

Kharya, S. (2012) 'Using Data Mining Techniques for Diagnosis and Prognosis of Cancer Disease', *International Journal of Computer Science, Engineering and Information Technology*, 2(2), pp. 55–66. Available at: https://doi.org/10.5121/ijcseit.2012.2206.

Mehran Emadi, S.S.H. (2015) 'Breast Cancer Tumour Diagnosis from Mammography Images Using Wavelet Transform and Hidden Markov Model', *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, 4(8), pp. 6815–6823. Available at: https://doi.org/10.15662/ijareeie.2015.0408002.

Morkonda Gunasekaran, D. and Dhandayudam, P. (2021) 'Design of novel multi filter union feature selection framework for breast cancer dataset', *Concurrent Engineering: Research and Applications*, pp. 1–6. Available at: https://doi.org/10.1177/1063293x211016046.

Mushtaq, Z. *et al.* (2019) 'Performance analysis of supervised classifiers using PCA based techniques on breast cancer', in *International Conference on Engineering and Emerging Technologies, ICEET 2019*. Available at: https://doi.org/10.1109/CEET1.2019.8711868.

Noble, H. and Smith, J. (2013) 'Machine learning for predicting the response of breast cancer to neoadjuvant chemotherapy', *Journal of the American Medical Informatics Association*, pp. 1–8.

Odajima, K. and Pawlovsky, A.P. (2014) 'A detailed description of the use of the kNN method for breast cancer diagnosis', in *Proceedings - 2014 7th International Conference on BioMedical Engineering and Informatics, BMEI 2014*. IEEE, pp. 688–692. Available at: https://doi.org/10.1109/BMEI.2014.7002861.

Overall Cancer Statistics - Annual Report to the Nation (2025). Available at: https://seer.cancer.gov/report_to_nation/stats.html (Accessed: 22 April 2025).

Pedregosa, F., Weiss, R. and Brucher, M. (2011) 'Scikit-learn : Machine Learning in Python', *Journal ofMachine Learning Research 12*, 12, pp. 2825–2830.

Putra, A.E., Wardhani, L.K. and Busman (2019) 'Chi-Square Feature Selection Effect On Naive Bayes Classifier Algorithm Performance For Sentiment Analysis Document', in *The 7th International Conference on Cyber and IT Service Management (CITSM 2019)*. Available at:

https://doi.org/10.1109/CITSM47753.2019.8965332.

Salem, N. and Hussein, S. (2019) 'Data dimensional reduction and principal components analysis', in *Procedia Computer Science*. Elsevier B.V., pp. 292–299. Available at: https://doi.org/10.1016/j.procs.2019.12.111.

Shaikh, T.A. and Ali, R. (2019) *Applying Machine Learning Algorithms for Early Diagnosis and Prediction of Breast Cancer Risk*, *Lecture Notes in Networks and Systems*. Springer Singapore. Available at: https://doi.org/10.1007/978-981-13-1217-5_57.

Siegel, R.L., Miller, K.D. and Jemal, A. (2017) 'Cancer statistics, 2017', *CA: A Cancer Journal for Clinicians*, 67(1), pp. 7–30. Available at: https://doi.org/10.3322/caac.21387.

Sung, H. *et al.* (2021) 'Global Cancer Statistics 2020: GLOBOCAN Estimates of Incidence and Mortality Worldwide for 36 Cancers in 185 Countries', *CA: A Cancer Journal for Clinicians*, 71(3), pp. 209–249. Available at: https://doi.org/10.3322/caac.21660.

The pandas development team (2020) 'pandas-dev/pandas: Pandas', *Zenodo* [Preprint]. Available at: https://doi.org/10.5281/zenodo.3509134.

Thiruvathukal, E.G.K. and Hunter, B.J.D. (2007) 'MATPLOTLIB : A 2D G RAPHICS E NVIRONMENT', *Computing in Science & Engineering*, 9(3), pp. 90–95. Available at: https://doi.org/10.1109/MCSE.2007.55.

Waskom, M.L. (2021) 'seaborn : statistical data visualization Statement of need', *The Journal of Open Source Software*, 6, pp. 1–4. Available at: https://doi.org/10.21105/joss.03021.

Weng, J. and Young, D.S. (2017) 'Some dimension reduction strategies for the analysis of survey data', *Journal of Big Data*, pp. 1–19. Available at: https://doi.org/10.1186/s40537-017-0103-6.

WIlliam Wolberg (1992) *Breast Cancer Wisconsin (Original) - UCI Machine Learning Repository*. Available at: https://archive.ics.uci.edu/dataset/15/breast+cancer+wisconsin+original (Accessed: 26 April 2025).

You, H. and Rumbe, G. (2010) 'Comparative Study of Classification Techniques on Breast Cancer FNA Biopsy Data', *International Journal of Interactive Multimedia and Artificial Intelligence*, 1(3), p. 5. Available at: https://doi.org/10.9781/ijimai.2010.131.

Appendix:

```
import itertools
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import classification_report,accuracy_score
#from sklearn import preprocessing,cross_validation,neighbors
from sklearn.model_selection import learning_curve
from sklearn.model_selection import ShuffleSplit
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from time import time
from sklearn.impute import SimpleImputer, KNNImputer
get_ipython().run_line_magic('matplotlib', 'inline')

dataset = pd.read_csv('breastcancer-wisconsin.data.txt')
dataset.replace('?', -999, inplace=True)

dataset.shape

dataset.describe()
dataset
# # PREPROCESSING
# to generate heat map with out samplecode number
dataset.drop(['Samplecodenumber'],1,inplace=True)
#X = dataset.iloc[:, 0:9].values
#Y = dataset.iloc[:, 9].values
```

```
for i in range(0,699):
   if dataset.iloc[i]["Bare Nuclei"]== -999:
      print (i,dataset.iloc[i]["Bare Nuclei"])

# # Checking missing values in the suspected columns

import pandas as pd
import numpy as np

print((dataset[["Clump-Thickness","Uniformity-CelShape","Bare Nuclei"]] == -999).sum())
# checking missing values in suspected columns

# # Checking the imputation methods MSE values, smaller is better

# To use this experimental feature, we need to explicitly ask for it:
from sklearn.experimental import enable_iterative_imputer  # noqa
from sklearn.datasets import fetch_california_housing
from sklearn.impute import SimpleImputer
from sklearn.impute import IterativeImputer
from sklearn.linear_model import BayesianRidge, Ridge
from sklearn.kernel_approximation import Nystroem
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import cross_val_score

N_SPLITS = 5
```

```
rng = np.random.RandomState(0)

#X_full, y_full = fetch_california_housing(return_X_y=True)
# ~2k samples is enough for the purpose of the example.
# Remove the following two lines for a slower run with different error bars.

X_full = dataset.iloc[:, 0:9].values
y_full = dataset.iloc[:, 9].values
#X_full = X_full[::10]
#y_full = y_full[::10]
n_samples, n_features = X_full.shape

# Estimate the score on the entire dataset, with no missing values
br_estimator = BayesianRidge()
score_full_data = pd.DataFrame(
   cross_val_score(
      br_estimator, X_full, y_full, scoring="neg_mean_squared_error", cv=N_SPLITS
   ),
   columns=["Full Data"],
)

# Add a single missing value to each row
X_missing = X_full.copy()
y_missing = y_full
missing_samples = np.arange(n_samples)
missing_features = rng.choice(n_features, n_samples, replace=True)
X_missing[missing_samples, missing_features] = -999

# Estimate the score after imputation (mean and median strategies)
```

```
score_simple_imputer = pd.DataFrame()
for strategy in ("mean", "median"):
    estimator = make_pipeline(
        SimpleImputer(missing_values=-999, strategy=strategy), br_estimator
    )
    score_simple_imputer[strategy] = cross_val_score(
        estimator, X_missing, y_missing, scoring="neg_mean_squared_error", cv=N_SPLITS
    )

# Estimate the score after iterative imputation of the missing values
# with different estimators
estimators = [
    BayesianRidge(),
    RandomForestRegressor(
        # We tuned the hyperparameters of the RandomForestRegressor to get a good
        # enough predictive performance for a restricted execution time.
        n_estimators=40,
        max_depth=17,
        bootstrap=True,
        max_samples=0.5,
        n_jobs=2,
        random_state=0,
    ),
    make_pipeline(
        Nystroem(kernel="polynomial", degree=2, random_state=0), Ridge(alpha=1e3)
    ),
    KNeighborsRegressor(n_neighbors=19),
]
score_iterative_imputer = pd.DataFrame()
```

```
# iterative imputer is sensible to the tolerance and
# dependent on the estimator used internally.
# we tuned the tolerance to keep this example run with limited computational
# resources while not changing the results too much compared to keeping the
# stricter default value for the tolerance parameter.
tolerances = (1e-3, 1e-1, 1e-1, 1e-2)
for impute_estimator, tol in zip(estimators, tolerances):
    estimator = make_pipeline(
        IterativeImputer(
            random_state=0, estimator=impute_estimator, max_iter=25, tol=tol
        ),
        br_estimator,
    )
    score_iterative_imputer[impute_estimator.__class__.__name__] = cross_val_score(
        estimator, X_missing, y_missing, scoring="neg_mean_squared_error", cv=N_SPLITS
    )

scores = pd.concat(
    [score_full_data, score_simple_imputer, score_iterative_imputer],
    keys=["Original", "SimpleImputer", "IterativeImputer"],
    axis=1,
)

# plot california housing results
fig, ax = plt.subplots(figsize=(15, 10))
means = -scores.mean()
errors = scores.std()
means.plot.barh(xerr=errors, ax=ax)
ax.set_title("Wisconsin breast cancer original with Different Imputation Methods")
```

```
ax.set_xlabel("MSE (smaller is better)")
ax.set_yticks(np.arange(means.shape[0]))
ax.set_yticklabels([" w/ ".join(label) for label in means.index.tolist()])
plt.tight_layout(pad=1)
plt.show()

# # Handling with Missing values

import seaborn as sns
sns.heatmap(dataset.corr(), center = 0, annot =True, cmap='GnBu');

import seaborn as sns
dataset_mean = dataset.describe().loc['mean']
dataset_mean.plot(kind='bar', figsize=(14,12))


# # DATASET VISUALIZATION

dataset_M = dataset[dataset['class'] == 2]
dataset_B = dataset[dataset['class'] == 4]
#dataset_HY = dataset[dataset['Class'] == 2]
B_M_data = {'Malignant': [dataset_M.shape[0]], 'Benign': [dataset_B.shape[0]]}
B_M_df = pd.DataFrame(data=B_M_data)
B_M_df.plot(kind='bar', figsize=(10,6), color=['green','orange'])
plt.title(f"Plotting total number of target variables-CLASS")
plt.show()


classs = dataset['class']
```

```
features = dataset.drop(['class'], axis = 1)
pd.plotting.scatter_matrix(features, alpha = 0.3, figsize = (20,18), diagonal = 'kde', c='red');

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler

# Load dataset
dataset = pd.read_csv('breastcancer-wisconsin.data.txt')
dataset.replace('?', -999, inplace=True)

# Splitting features and target
X_full = dataset.iloc[:, 0:9].values
y_full = dataset.iloc[:, 9].values

# Handling missing values by replacing them with the median (for simplicity)
X_full = pd.DataFrame(X_full)
X_full = X_full.apply(pd.to_numeric, errors='coerce')  # Convert to numeric (handle non-numeric values)
#X_full.fillna(X_full.median(), inplace=True)  # Fill missing values with median

# 1. Histograms (distribution of features)
plt.figure(figsize=(12, 8))
for i, feature in enumerate(X_full.columns):
    plt.subplot(3, 3, i+1)
    sns.histplot(X_full[feature], kde=True)
```

```
    plt.title(f"Histogram of {feature}")
plt.tight_layout()
plt.show()

# 2. Box plots (checking outliers)
plt.figure(figsize=(12, 8))
for i, feature in enumerate(X_full.columns):
    plt.subplot(3, 3, i+1)
    sns.boxplot(x=y_full, y=X_full[feature])
    plt.title(f"Boxplot of {feature} by class")
plt.tight_layout()
plt.show()

# 9. Density plots (Kernel Density Estimate)
plt.figure(figsize=(12, 8))
for i, feature in enumerate(X_full.columns):
    plt.subplot(3, 3, i+1)
    sns.kdeplot(X_full[feature], shade=True)
    plt.title(f"Density plot of {feature}")
plt.tight_layout()
plt.show()

# 10. Pie chart (categorical distribution)
dataset['class'].value_counts().plot(kind='pie', autopct='%1.1f%%', figsize=(6, 6))
plt.title("Pie chart of class distribution")
plt.ylabel('')
plt.show()
```

```
import matplotlib.pyplot as plt
import numpy as np

# Getting value counts of the target variable
counts = dataset['class'].value_counts()

# Creating the pie chart with enhanced 3D-like effects
fig, ax = plt.subplots(figsize=(8, 8))

# Data for the pie chart
labels = counts.index
sizes = counts.values
explode = [0.1] * len(labels)  # Exploding the slices for better visuals

# Plotting the pie chart
wedges, texts, autotexts = ax.pie(sizes, labels=labels, autopct='%1.1f%%', explode=explode,
                                  startangle=90, shadow=True, wedgeprops={'edgecolor': 'black', 'linewidth': 1.5})

# Adding 3D-like effect with a shadow and setting the angle for a dynamic appearance
for wedge in wedges:
    wedge.set_linewidth(1.5)
    wedge.set_edgecolor('black')

# Customizing the appearance of the plot
ax.set_title("Simulated 3D Pie Chart of Class Distribution")
ax.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

# Show the plot
plt.show()
```

```
# to generat heat map with out samplecode number

dataset.drop(['Samplecodenumber'],1,inplace=True)


dataset=dataset.replace({'class':{2:0, 4:1}})


# # mean imputation


# In[23]:



mean_imputer = SimpleImputer(missing_values= -999, strategy='mean')# fill missing values from imputter, which uses means

mean_imputer = mean_imputer.fit(dataset)

imputed_dataset1 = mean_imputer.transform(dataset.values)

imputed_dataset1[23: 24]


Xil= imputed_dataset1[:,0:9]# initializing XI AND YI to the new imputed_dataset1, which dont have missing values

Yil= imputed_dataset1[:,9]

print(Xil)

Xil.shape


# # Factor Analysis 2D and 3D visualization


# In[28]:

import matplotlib.pyplot as plt

import seaborn as sns

from matplotlib.colors import ListedColormap
```

```
from sklearn.decomposition import FactorAnalysis

from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.metrics import accuracy_score, confusion_matrix

#from sklearn.neighbors import KNeighborsClassifier,NeighborhoodComponentsAnalysis,
LocalOutlierFactor

from sklearn.decomposition import PCA, TruncatedSVD

from sklearn.preprocessing import MinMaxScaler, StandardScaler

scaler = MinMaxScaler()

x_scaled = scaler.fit_transform(Xil)


fa = FactorAnalysis(n_components=2, random_state=42)

fa.fit(x_scaled)

X_reduced_fa = fa.transform(x_scaled)

fa_data = pd.DataFrame(X_reduced_fa, columns=["p1", "p2"])

fa_data["target"] = Yil

plt.figure(figsize=(10,8))


sns.scatterplot(x="p1", y="p2" , hue="target", data=fa_data,palette=['orange','blue'], legend='full')


plt.title("Factor Analysis with two components p1 vs p2 with MinMax scaling ")

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.decomposition import FactorAnalysis

from sklearn.preprocessing import MinMaxScaler

from mpl_toolkits.mplot3d import Axes3D

import pandas as pd


# Assuming Xil and Yil are defined earlier in your code
```

```
scaler = MinMaxScaler()
x_scaled = scaler.fit_transform(Xil)

fa = FactorAnalysis(n_components=3, random_state=42)  # Set components to 3 for 3D
fa.fit(x_scaled)
X_reduced_fa = fa.transform(x_scaled)
fa_data = pd.DataFrame(X_reduced_fa, columns=["p1", "p2", "p3"])
fa_data["target"] = Yil

# Creating a 3D plot
fig = plt.figure(figsize=(10,8))
ax = fig.add_subplot(111, projection='3d')

# Scatter plot in 3D
scatter = ax.scatter(fa_data["p1"], fa_data["p2"], fa_data["p3"], c=fa_data["target"], cmap='plasma')

# Adding labels
ax.set_xlabel('p1')
ax.set_ylabel('p2')
ax.set_zlabel('p3')
ax.set_title("Factor Analysis with three components p1, p2, p3 with MinMax scaling")

# Show color bar
plt.colorbar(scatter)

plt.show()
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X_reduced_fa, Yil, test_size = 0.2)

from sklearn.ensemble import RandomForestClassifier
import time
#tic = time()

start = time.time()
clf1 = RandomForestClassifier(criterion='gini', max_depth=10, random_state=123)

#t0=time()
#print("training time:", round(time()-t0, 3), "s")
clf1.fit(X_train, Y_train)
stop = time.time()
print(f"Training time: {stop - start}s")



start1 = time.time()
RF=clf1.predict(X_test)
stop1 = time.time()
print(f"Predicting time: {stop1 - start1}s")
#toc = time()
#t1=time()
#print("Predicting time:", round(time()-t0, 3), "s")
accRF=clf1.score(X_test,Y_test)

print(accRF,'Random Forest')
print(classification_report(Y_test,RF))
```

```
from sklearn.metrics import hamming_loss
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import matthews_corrcoef
from sklearn.metrics import zero_one_loss

from sklearn.metrics import precision_score, recall_score, f1_score

# Assuming Y_test are the true labels and RF are the predicted labels
precision = precision_score(Y_test, RF)
recall = recall_score(Y_test, RF)
f1 = f1_score(Y_test, RF)

# Display the results
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues, fontsize= 40):
 """
 This function prints and plots the confusion matrix.
 Normalization can be applied by setting `normalize=True`.
 """
 if normalize:
     cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
```

```
    print("Normalized confusion matrix")
  else:
    print('Confusion matrix, without normalization')

  print(cm)

  plt.imshow(cm, interpolation='nearest', cmap=cmap)
  plt.title(title)
  plt.colorbar()
  tick_marks = np.arange(len(classes))
  plt.xticks(tick_marks, classes, rotation=0)
  plt.yticks(tick_marks, classes)

  fmt = '.2f' if normalize else 'd'
  thresh = cm.max() / 2.
  for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, format(cm[i, j], fmt),
          horizontalalignment="center",
          color="white" if cm[i, j] > thresh else "black")

  plt.tight_layout()
  plt.ylabel('True label')
  plt.xlabel('Predicted label')
# Compute confusion matrix
from sklearn.metrics import confusion_matrix
cn_matrix = confusion_matrix(Y_test, RF)
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
```

```
plt.figure()
classs=['2-as-Benigen','4-as-Malignant']
plot_confusion_matrix(cn_matrix, classes=classs,normalize=False,
                      title='Confusion matrix,RF-FA-mean imputation',fontsize=40)
plt.show()
classs=['2-as-Benigen','4-as-Malignant']
plot_confusion_matrix(cn_matrix, classes=classs,normalize=True,
                      title='Confusion matrix,RF-FA-mean imputation',fontsize=40)
plt.show()


import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
 """
 This function prints and plots the confusion matrix.
 Normalization can be applied by setting `normalize=True`.
 """
 if normalize:
     cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
     print("Normalized confusion matrix")
 else:
     print('Confusion matrix, without normalization')


 print(cm)


 plt.imshow(cm, interpolation='nearest', cmap=cmap)
 plt.title(title)
```

```
 plt.colorbar()
 tick_marks = np.arange(len(classes))
 plt.xticks(tick_marks, classes, rotation=45)
 plt.yticks(tick_marks, classes)

 fmt = '.2f' if normalize else 'd'
 thresh = cm.max() / 2.
 for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
     plt.text(j, i, format(cm[i, j], fmt),
              horizontalalignment="center",
              color="white" if cm[i, j] > thresh else "black")

 plt.tight_layout()
 plt.ylabel('True label')
 plt.xlabel('Predicted label')



from sklearn.metrics import confusion_matrix
cn_matrix = confusion_matrix(Y_test, RF)
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plt.rcParams.update({'font.size': 14})
plt.figure()

plt.figure()
classs=['2-as-Benigen','4-as-Malignant']
plot_confusion_matrix(cn_matrix, classes=classs,normalize=False,
                      title='Confusion matrix,RF-FA-mean imputation')
```

```
plt.show()
classs=['2-as-Benigen','4-as-Malignant']
plot_confusion_matrix(cn_matrix, classes=classs,normalize=True,
                      title='Confusion matrix,RF-FA-mean imputation')
plt.show()



# # PCA Based results
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, confusion_matrix
#from sklearn.neighbors import KNeighborsClassifier,NeighborhoodComponentsAnalysis,
LocalOutlierFactor
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler
scaler = StandardScaler()
sc = MinMaxScaler()
x_scaled = scaler.fit_transform(Xil)

pca = PCA(n_components=2)
pca.fit(x_scaled)
X_reduced_pca = pca.transform(x_scaled)
pca_data = pd.DataFrame(X_reduced_pca, columns=["p1", "p2"])
pca_data["target"] = Yil
plt.figure(figsize=(10,8))

sns.scatterplot(x="p1", y="p2" , hue="target", data=pca_data,palette=['green','blue'], legend='full')
plt.grid()
plt.title("WBC data PCA=2: PC1 vs PC2 ")
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler
from mpl_toolkits.mplot3d import Axes3D
import pandas as pd

# Assuming Xil and Yil are defined earlier in your code
scaler = MinMaxScaler()
x_scaled = scaler.fit_transform(Xil)

# Apply PCA to reduce to 3 components
pca = PCA(n_components=3, random_state=42)  # Set components to 3 for 3D projection
pca.fit(x_scaled)
X_reduced_pca = pca.transform(x_scaled)  # Transform data into 3D

# Create a DataFrame for the reduced PCA components and add the target labels
pca_data = pd.DataFrame(X_reduced_pca, columns=["p1", "p2", "p3"])
pca_data["target"] = Yil

# Create a 3D scatter plot
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# Scatter plot in 3D with color based on target class
scatter = ax.scatter(pca_data["p1"], pca_data["p2"], pca_data["p3"], c=pca_data["target"],
cmap='viridis')
```

```
# Adding axis labels and title
ax.set_xlabel('p1')
ax.set_ylabel('p2')
ax.set_zlabel('p3')
ax.set_title("PCA with three components p1, p2, p3 with MinMax scaling")

# Show the plot
plt.show()

from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X_reduced_pca, Yil, test_size = 0.2)

from sklearn.ensemble import RandomForestClassifier
import time
#tic = time()

start = time.time()
clf2 = RandomForestClassifier(criterion='gini', max_depth=10, random_state=123)

#t0=time()
#print("training time:", round(time()-t0, 3), "s")
clf2.fit(X_train, Y_train)
stop = time.time()
print(f"Training time: {stop - start}s")


start1 = time.time()
RF2=clf2.predict(X_test)
```

```
stop1 = time.time()
print(f"Predicting time: {stop1 - start1}s")
#toc = time()
#t1=time()
#print("Predicting time:", round(time()-t0, 3), "s")
accRF=clf2.score(X_test,Y_test)

print(accRF,'Random Forest')
print(classification_report(Y_test,RF2))

from sklearn.metrics import precision_score, recall_score, f1_score

# Assuming Y_test are the true labels and RF are the predicted labels
precision = precision_score(Y_test, RF2)
recall = recall_score(Y_test, RF2)
f1 = f1_score(Y_test, RF2)

# Display the results
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")

# In[40]:


def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Greens):
```

```
"""
This function prints and plots the confusion matrix.
Normalization can be applied by setting `normalize=True`.
"""
if normalize:
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    print("Normalized confusion matrix")
else:
    print('Confusion matrix, without normalization')

print(cm)

plt.imshow(cm, interpolation='nearest', cmap=cmap)
plt.title(title)
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=0)
plt.yticks(tick_marks, classes)

fmt = '.2f' if normalize else 'd'
thresh = cm.max() / 2.
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, format(cm[i, j], fmt),
             horizontalalignment="center",
             color="white" if cm[i, j] > thresh else "black")

plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

```
# Compute confusion matrix
from sklearn.metrics import confusion_matrix
cn_matrix = confusion_matrix(Y_test, RF2)
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plt.figure()
classs=['2-as-Benigen','4-as-Malignant']
plot_confusion_matrix(cn_matrix, classes=classs,normalize=False,
                      title='Confusion matrix,RF-PCA-mean imputation')
plt.show()
classs=['2-as-Benigen','4-as-Malignant']
plot_confusion_matrix(cn_matrix, classes=classs,normalize=True,
                      title='Confusion matrix,RF-PCA-mean imputation')
plt.show()

# # Feature Selection WITH OUT Dimensionality Reduction

# # CHI2 FS

# In[42]:



X_n = SelectKBest(chi2, k=6).fit_transform(x_scaled, Yil)
X_n.shape



# In[43]:
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X_n, Yil, test_size = 0.2)



# In[44]:



from sklearn.ensemble import RandomForestClassifier
import time
#tic = time()

start = time.time()
clf3 = RandomForestClassifier(criterion='gini', max_depth=10, random_state=123)

#t0=time()
#print("training time:", round(time()-t0, 3), "s")
clf3.fit(X_train, Y_train)
stop = time.time()
print(f"Training time: {stop - start}s")



start1 = time.time()
RF3=clf3.predict(X_test)
stop1 = time.time()
print(f"Predicting time: {stop1 - start1}s")
#toc = time()
#t1=time()
#print("Predicting time:", round(time()-t0, 3), "s")
```

```
accRF=clf3.score(X_test,Y_test)


print(accRF,'Random Forest')
print(classification_report(Y_test,RF3))


# Assuming Y_test are the true labels and RF are the predicted labels
precision = precision_score(Y_test, RF3)
recall = recall_score(Y_test, RF3)
f1 = f1_score(Y_test, RF3)


# Display the results
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
# In[46]:



def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Oranges):
 """
 This function prints and plots the confusion matrix.
 Normalization can be applied by setting `normalize=True`.
 """
 if normalize:
     cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
     print("Normalized confusion matrix")
 else:
```

```
    print('Confusion matrix, without normalization')

 print(cm)

 plt.imshow(cm, interpolation='nearest', cmap=cmap)
 plt.title(title)
 plt.colorbar()
 tick_marks = np.arange(len(classes))
 plt.xticks(tick_marks, classes, rotation=0)
 plt.yticks(tick_marks, classes)

 fmt = '.2f' if normalize else 'd'
 thresh = cm.max() / 2.
 for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, format(cm[i, j], fmt),
          horizontalalignment="center",
          color="white" if cm[i, j] > thresh else "black")

 plt.tight_layout()
 plt.ylabel('True label')
 plt.xlabel('Predicted label')
# Compute confusion matrix
from sklearn.metrics import confusion_matrix
cn_matrix = confusion_matrix(Y_test, RF3)
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plt.figure()
classs=['2-as-Benigen','4-as-Malignant']
```

```
plot_confusion_matrix(cn_matrix, classes=classs,normalize=False,
                      title='Confusion matrix,RF-CHI2-mean imputation')
plt.show()
classs=['2-as-Benigen','4-as-Malignant']
plot_confusion_matrix(cn_matrix, classes=classs,normalize=True,
                      title='Confusion matrix,RF-CHI2-mean imputation')
plt.show()



# # Recurrsive Feature Elimination

# In[47]:



from sklearn.svm import LinearSVC, SVR
from sklearn.feature_selection import RFE
estimator = SVR(kernel="linear")
selector = RFE(estimator, n_features_to_select=7)
selector = selector.fit(x_scaled, Yil)



# In[48]:



scaler = StandardScaler()
x_scaled = scaler.fit_transform(Xil)  # Make sure x_scaled is 2D (shape: [n_samples, n_features])

# Define the estimator for RFE (Support Vector Regression with a linear kernel)
estimator = SVR(kernel="linear")
```

```
# Initialize the RFE selector with the estimator and number of features to select
selector = RFE(estimator, n_features_to_select=7)  # Adjust the number as needed (e.g., 7 features)

# Fit the RFE selector to the scaled data
selector.fit(x_scaled, Yil)



# In[49]:



# Define the estimator for RFE (Support Vector Regression with a linear kernel)
estimator = SVR(kernel="linear")

# Initialize the RFE selector with the estimator and number of features to select
selector = RFE(estimator, n_features_to_select=7)  # Adjust the number as needed (e.g., 7 features)

# Fit the RFE selector to the scaled data
selector.fit(x_scaled, Yil)

# Get the selected features (True means the feature was selected)
selected_features = selector.support_
print("Selected features:", selected_features)

# Prepare the data with the selected features (extract the selected columns from the original scaled data)
x_selected = x_scaled[:, selected_features]

# Split the data into training and test sets
```

```
X_train, X_test, Y_train, Y_test = train_test_split(x_selected, Yil, test_size=0.2, random_state=42)



from sklearn.ensemble import RandomForestClassifier
import time
#tic = time()

start = time.time()
clf4 = RandomForestClassifier(criterion='gini', max_depth=10, random_state=123)

#t0=time()
#print("training time:", round(time()-t0, 3), "s")
clf4.fit(X_train, Y_train)
stop = time.time()
print(f"Training time: {stop - start}s")



start1 = time.time()
RF4=clf4.predict(X_test)
stop1 = time.time()
print(f"Predicting time: {stop1 - start1}s")
#toc = time()
#t1=time()
#print("Predicting time:", round(time()-t0, 3), "s")
accRF=clf4.score(X_test,Y_test)

print(accRF,'Random Forest')
print(classification_report(Y_test,RF4))
```

```
from sklearn.metrics import precision_score, recall_score, f1_score

# Assuming Y_test are the true labels and RF are the predicted labels
precision = precision_score(Y_test, RF4)
recall = recall_score(Y_test, RF4)
f1 = f1_score(Y_test, RF4)

# Display the results
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")



MCC = matthews_corrcoef(Y_test,RF4)
print('MCC:', MCC)



# In[52]:



def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Purples):
 """
 This function prints and plots the confusion matrix.
 Normalization can be applied by setting `normalize=True`.
 """
 if normalize:
```

```
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    print("Normalized confusion matrix")
  else:
    print('Confusion matrix, without normalization')

  print(cm)

  plt.imshow(cm, interpolation='nearest', cmap=cmap)
  plt.title(title)
  plt.colorbar()
  tick_marks = np.arange(len(classes))
  plt.xticks(tick_marks, classes, rotation=0)
  plt.yticks(tick_marks, classes)

  fmt = '.2f' if normalize else 'd'
  thresh = cm.max() / 2.
  for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, format(cm[i, j], fmt),
          horizontalalignment="center",
          color="white" if cm[i, j] > thresh else "black")

  plt.tight_layout()
  plt.ylabel('True label')
  plt.xlabel('Predicted label')
# Compute confusion matrix
from sklearn.metrics import confusion_matrix
cn_matrix = confusion_matrix(Y_test, RF4)
np.set_printoptions(precision=2)
```

```
# Plot non-normalized confusion matrix
plt.figure()
classs=['2-as-Benigen','4-as-Malignant']
plot_confusion_matrix(cn_matrix, classes=classs,normalize=False,
                      title='Confusion matrix,RF-RFE-mean imputation')
plt.show()
classs=['2-as-Benigen','4-as-Malignant']
plot_confusion_matrix(cn_matrix, classes=classs,normalize=True,
                      title='Confusion matrix,RF-RFE-mean imputation')
plt.show()



# # COMBINATION OF PCA with RFE


# In[53]:



import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.feature_selection import RFE
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
```

```
# Example dataset (replace with your dataset)
# Xil = your feature matrix
# Yil = your target vector

# Step 1: Scale the data using StandardScaler and MinMaxScaler
scaler = StandardScaler()
x_scaled = scaler.fit_transform(Xil)

# Step 2: Apply PCA (reduce the number of components)
pca = PCA(n_components=2)  # You can adjust n_components based on your needs
pca.fit(x_scaled)
X_reduced_pca = pca.transform(x_scaled)
pca_data = pd.DataFrame(X_reduced_pca, columns=["p1", "p2"])
pca_data["target"] = Yil


# Step 3: Apply RFE to the PCA-transformed data
estimator = SVR(kernel="linear")
selector = RFE(estimator, n_features_to_select=7)  # Adjust the number of features you want to select
selector.fit(X_reduced_pca, Yil)


# Prepare the data with the selected features (select only the columns of the selected features)
x_selected = X_reduced_pca[:, selected_features]

# Step 4: Split the data into training and test sets
X_train, X_test, Y_train, Y_test = train_test_split(x_selected, Yil, test_size=0.2, random_state=123)
```

```
# Step 5: Train a Random Forest Classifier on the selected features
rf_classifier = RandomForestClassifier(n_estimators=100,criterion='gini', max_depth=17, random_state=123)
rf_classifier.fit(X_train, Y_train)

# Step 6: Make predictions
Y_pred = rf_classifier.predict(X_test)

# Step 8: Evaluate the model using accuracy score and confusion matrix
accuracy = accuracy_score(Y_test, Y_pred)
print(f"Accuracy of Random Forest Classifier on selected features: {accuracy:.4f}")

# Confusion Matrix
cm = confusion_matrix(Y_test, Y_pred)
print("Confusion Matrix:")
print(cm)


# In[54]:


from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.decomposition import PCA
from sklearn.feature_selection import RFE
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
```

```
import pandas as pd
import numpy as np

# Assuming Xil and Yil are defined

# Step 1: Scale the data using StandardScaler
scaler = StandardScaler()
x_scaled = scaler.fit_transform(Xil)

# Step 2: Apply PCA (increase the number of components)
pca = PCA(n_components=5)  # Increased to 5 components
X_reduced_pca = pca.fit_transform(x_scaled)

# Visualize explained variance ratio (optional)
print("Explained variance ratio:", pca.explained_variance_ratio_)

# Step 3: Apply RFE to the PCA-transformed data
estimator = SVR(kernel="linear")
selector = RFE(estimator, n_features_to_select=5)  # Select 5 features
selector.fit(X_reduced_pca, Yil)

# Get the selected features
selected_features = selector.support_  # This is a boolean array
print("Selected features:", selected_features)

# Prepare the data with the selected features
x_selected = X_reduced_pca[:, selected_features]

# Step 4: Split the data into training and test sets
```

```
X_train, X_test, Y_train, Y_test = train_test_split(x_selected, Yil, test_size=0.2, random_state=123)

# Step 5: Train a Random Forest Classifier on the selected features
# Tune the hyperparameters using GridSearchCV
param_grid = {
  'n_estimators': [100, 200, 300],
  'max_depth': [10, 15, 20, None],
  'min_samples_split': [2, 5, 10],
  'min_samples_leaf': [1, 2, 4]
}

rf_classifier = RandomForestClassifier(random_state=123)
grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)
grid_search.fit(X_train, Y_train)

# Best parameters found from grid search
print(f"Best parameters: {grid_search.best_params_}")

# Use the best estimator from the grid search
best_rf_classifier = grid_search.best_estimator_

# Step 6: Make predictions
Y_pred = best_rf_classifier.predict(X_test)

# Step 7: Evaluate the model using accuracy score and confusion matrix
accuracy = accuracy_score(Y_test, Y_pred)
print(f"Accuracy of Random Forest Classifier on selected features: {accuracy:.4f}")
```

```
# Confusion Matrix
cm = confusion_matrix(Y_test, Y_pred)
print("Confusion Matrix:")
print(cm)


# In[56]:



from sklearn.ensemble import RandomForestClassifier
import time
#tic = time()

start = time.time()
#clf4 = RandomForestClassifier(criterion='gini', max_depth=10, random_state=123)

#t0=time()
#print("training time:", round(time()-t0, 3), "s")
best_rf_classifier.fit(X_train, Y_train)
stop = time.time()
print(f"Training time: {stop - start}s")


start1 = time.time()
RF5=best_rf_classifier.predict(X_test)
stop1 = time.time()
print(f"Predicting time: {stop1 - start1}s")
#toc = time()
```

```
#t1=time()
#print("Predicting time:", round(time()-t0, 3), "s")
accRF=best_rf_classifier.score(X_test,Y_test)

print(accRF,'Random Forest')
print(classification_report(Y_test,RF5))



# In[57]:


from sklearn.metrics import precision_score, recall_score, f1_score

# Assuming Y_test are the true labels and RF are the predicted labels
precision = precision_score(Y_test, RF5)
recall = recall_score(Y_test, RF5)
f1 = f1_score(Y_test, RF5)

# Display the results
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Wistia):
 """
 This function prints and plots the confusion matrix.
 Normalization can be applied by setting `normalize=True`.
```

```
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=0)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
# Compute confusion matrix
from sklearn.metrics import confusion_matrix
cn_matrix = confusion_matrix(Y_test, RF5)
```

```
np.set_printoptions(precision=2)


# Plot non-normalized confusion matrix
plt.figure()
classs=['2-as-Benigen','4-as-Malignant']
plot_confusion_matrix(cn_matrix, classes=classs,normalize=False,
                      title='Confusion matrix,Mean-PCA-RFE-RF')
plt.show()
classs=['2-as-Benigen','4-as-Malignant']
plot_confusion_matrix(cn_matrix, classes=classs,normalize=True,
                      title='Confusion matrix,Mean-PCA-RFE-RF')
plt.show()

##################################################

For the WDBC
# In[1]:



import itertools
#import itertools
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import classification_report,accuracy_score
#from sklearn import preprocessing,cross_validation,neighbors
from sklearn.model_selection import learning_curve
from sklearn.model_selection import ShuffleSplit
from sklearn.feature_selection import SelectKBest
```

```
from sklearn.feature_selection import chi2
from time import time
get_ipython().run_line_magic('matplotlib', 'inline')


# In[18]:


dataset = pd.read_csv('wdbc.csv')


# In[19]:


dataset.head()


# # Converting data set class 'B' and 'M' as 0 and 1

# In[21]:


dataset.drop(['id'],1,inplace=True)
dataset=dataset.replace({'diagnoses':{'B':0, 'M':1}})
dataset.head()


# In[22]:
```

```
X = dataset.iloc[:, 1:32].values
Y = dataset.iloc[:, 0].values

print(X)
print(Y)
X.shape


# In[23]:


import seaborn as sns
sns.heatmap(dataset.corr(), center = 0, annot =True, cmap='GnBu');

classs = dataset['diagnoses']
features = dataset.drop(['diagnoses'], axis = 1)
pd.plotting.scatter_matrix(features, alpha = 0.3, figsize = (30,28), diagonal = 'kde', c='red');


# In[26]:


import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler
```

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Example dataset, replace with your actual dataset
# dataset = pd.read_csv('your_dataset.csv')

X_full = dataset.iloc[:, 1:32].values
Y_full = dataset.iloc[:, 0].values

# Handling missing values by replacing them with the median (for simplicity)
X_full = pd.DataFrame(X_full)
X_full = X_full.apply(pd.to_numeric, errors='coerce')  # Convert to numeric (handle non-numeric values)
# X_full.fillna(X_full.median(), inplace=True)  # Fill missing values with median

# 1. Histograms (distribution of features)
plt.figure(figsize=(15, 15))  # Adjusted figure size for 6x6 grid
num_features = X_full.shape[1]

# Create subplots dynamically
for i, feature in enumerate(X_full.columns):
    plt.subplot(6, 6, i+1)  # 6x6 grid for 31 features
    sns.histplot(X_full[feature], kde=True)
    plt.title(f"Histogram of {feature}")

plt.tight_layout()
plt.show()
```

```
# In[28]:



import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd


# Example dataset, replace with your actual dataset

# dataset = pd.read_csv('your_dataset.csv')


X_full = dataset.iloc[:, 1:32].values

Y_full = dataset.iloc[:, 0].values


# Handling missing values by replacing them with the median (for simplicity)

X_full = pd.DataFrame(X_full)

X_full = X_full.apply(pd.to_numeric, errors='coerce')  # Convert to numeric (handle non-numeric values)

# X_full.fillna(X_full.median(), inplace=True)  # Fill missing values with median


# 1. Boxplots (distribution of features)

plt.figure(figsize=(15, 15))  # Adjusted figure size for 6x6 grid

num_features = X_full.shape[1]


# Create subplots dynamically for boxplots

for i, feature in enumerate(X_full.columns):

    plt.subplot(6, 6, i+1)  # 6x6 grid for 31 features

    sns.boxplot(x=X_full[feature])

    plt.title(f"Boxplot of {feature}")
```

```
plt.tight_layout()
plt.show()



# In[30]:



import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Example dataset, replace with your actual dataset
# dataset = pd.read_csv('your_dataset.csv')

X_full = dataset.iloc[:, 1:32].values
Y_full = dataset.iloc[:, 0].values

# Handling missing values by replacing them with the median (for simplicity)
X_full = pd.DataFrame(X_full)
X_full = X_full.apply(pd.to_numeric, errors='coerce')  # Convert to numeric (handle non-numeric values)
# X_full.fillna(X_full.median(), inplace=True)  # Fill missing values with median

# 2. Box plots (checking outliers by class)
plt.figure(figsize=(15, 15))  # Adjusted figure size for 6x6 grid
num_features = X_full.shape[1]

# Create subplots dynamically for boxplots
```

```
for i, feature in enumerate(X_full.columns):
    plt.subplot(6, 6, i+1)  # 6x6 grid for 31 features
    sns.boxplot(x=Y_full, y=X_full[feature])
    plt.title(f"Boxplot of {feature} by class")

plt.tight_layout()
plt.show()



# In[31]:



import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Example dataset, replace with your actual dataset
# dataset = pd.read_csv('your_dataset.csv')

X_full = dataset.iloc[:, 1:32].values
Y_full = dataset.iloc[:, 0].values

# Handling missing values by replacing them with the median (for simplicity)
X_full = pd.DataFrame(X_full)
X_full = X_full.apply(pd.to_numeric, errors='coerce')  # Convert to numeric (handle non-numeric values)
# X_full.fillna(X_full.median(), inplace=True)  # Fill missing values with median

# 3. Density plots (distribution of features)
```

```
plt.figure(figsize=(15, 15))  # Adjusted figure size for 6x6 grid
num_features = X_full.shape[1]

# Create subplots dynamically for density plots
for i, feature in enumerate(X_full.columns):
    plt.subplot(6, 6, i+1)  # 6x6 grid for 31 features
    sns.kdeplot(X_full[feature], shade=True)
    plt.title(f"Density Plot of {feature}")

plt.tight_layout()
plt.show()



# In[34]:



import matplotlib.pyplot as plt
import numpy as np

# Getting value counts of the target variable
counts = dataset['diagnoses'].value_counts()

# Creating the pie chart with enhanced 3D-like effects
fig, ax = plt.subplots(figsize=(8, 8))

# Data for the pie chart
labels = counts.index
sizes = counts.values
explode = [0.1] * len(labels)  # Exploding the slices for better visuals
```

```
# Plotting the pie chart
wedges, texts, autotexts = ax.pie(sizes, labels=labels, autopct='%1.1f%%', explode=explode,
                                  startangle=90, shadow=True, wedgeprops={'edgecolor': 'black', 'linewidth': 1.5})

# Adding 3D-like effect with a shadow and setting the angle for a dynamic appearance
for wedge in wedges:
    wedge.set_linewidth(1.5)
    wedge.set_edgecolor('black')

# Customizing the appearance of the plot
ax.set_title("Simulated 3D Pie Chart of Class Distribution")
ax.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

# Show the plot
plt.show()

# Feature Scaling
from sklearn.preprocessing import MinMaxScaler
#sc = StandardScaler()
X_norm = MinMaxScaler().fit_transform(X)



# # Chi2 Feature Selection

# In[7]:



X_new = SelectKBest(chi2, k=22).fit_transform(X_norm, Y)
```

```
X_new.shape


# In[9]:


from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X_new, Y, test_size = 0.2)


# # Randm Forest Clf CHI2 clf3

# In[41]:


from sklearn.ensemble import RandomForestClassifier
import time
#tic = time()

start = time.time()
clf3 = RandomForestClassifier(criterion='gini', max_depth=10, random_state=123)

#t0=time()
#print("training time:", round(time()-t0, 3), "s")
clf3.fit(X_train, Y_train)
stop = time.time()
print(f"Training time: {stop - start}s")
```

```
start1 = time.time()
RF3=clf3.predict(X_test)
stop1 = time.time()
print(f"Predicting time: {stop1 - start1}s")
#toc = time()
#t1=time()
#print("Predicting time:", round(time()-t0, 3), "s")
accRF=clf3.score(X_test,Y_test)

print(accRF,'Random Forest')
print(classification_report(Y_test,RF3))



# In[42]:



from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import hamming_loss
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import matthews_corrcoef
from sklearn.metrics import zero_one_loss
from sklearn import metrics

from sklearn.metrics import precision_score, recall_score, f1_score

# Assuming Y_test are the true labels and RF are the predicted labels
precision = precision_score(Y_test, RF3)
recall = recall_score(Y_test, RF3)
```

```
f1 = f1_score(Y_test, RF3)

# Display the results
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")

# In[43]:



def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Oranges):
 """
 This function prints and plots the confusion matrix.
 Normalization can be applied by setting `normalize=True`.
 """
 if normalize:
     cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
     print("Normalized confusion matrix")
 else:
     print('Confusion matrix, without normalization')

 print(cm)

 plt.imshow(cm, interpolation='nearest', cmap=cmap)
 plt.title(title)
 plt.colorbar()
```

```
 tick_marks = np.arange(len(classes))
 plt.xticks(tick_marks, classes, rotation=0)
 plt.yticks(tick_marks, classes)

 fmt = '.2f' if normalize else 'd'
 thresh = cm.max() / 2.
 for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
     plt.text(j, i, format(cm[i, j], fmt),
             horizontalalignment="center",
             color="white" if cm[i, j] > thresh else "black")

 plt.tight_layout()
 plt.ylabel('True label')
 plt.xlabel('Predicted label')
# Compute confusion matrix
from sklearn.metrics import confusion_matrix
cn_matrix = confusion_matrix(Y_test, RF3)
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plt.figure()
classs=['2-as-Benigen','4-as-Malignant']
plot_confusion_matrix(cn_matrix, classes=classs,normalize=False,
                      title='Confusion matrix,RF-CHI2-WDBC')
plt.show()
classs=['2-as-Benigen','4-as-Malignant']
plot_confusion_matrix(cn_matrix, classes=classs,normalize=True,
                      title='Confusion matrix,RF-CHI2-WDBC')
plt.show()
```

RFE Feature selection

In[44]:

```
from sklearn.svm import LinearSVC, SVR
from sklearn.feature_selection import RFE
estimator = SVR(kernel="linear")
selector = RFE(estimator, n_features_to_select=24)
selector = selector.fit(X_norm, Y)
```

In[45]:

```
scaler = StandardScaler()
x_scaled = scaler.fit_transform(X)  # Make sure x_scaled is 2D (shape: [n_samples, n_features])

# Define the estimator for RFE (Support Vector Regression with a linear kernel)
estimator = SVR(kernel="linear")

# Initialize the RFE selector with the estimator and number of features to select
selector = RFE(estimator, n_features_to_select=24)  # Adjust the number as needed (e.g., 7 features)

# Fit the RFE selector to the scaled data
selector.fit(x_scaled, Y)
```

```
# Get the selected features (True means the feature was selected)
selected_features = selector.support_
print("Selected features:", selected_features)

# Prepare the data with the selected features (extract the selected columns from the original scaled data)
x_selected = x_scaled[:, selected_features]

# Split the data into training and test sets
X_train, X_test, Y_train, Y_test = train_test_split(x_selected, Y, test_size=0.2, random_state=42)



# In[51]:



from sklearn.ensemble import RandomForestClassifier
import time
#tic = time()

start = time.time()
clf4 = RandomForestClassifier(criterion='gini', max_depth=6, random_state=42)

#t0=time()
#print("training time:", round(time()-t0, 3), "s")
clf4.fit(X_train, Y_train)
stop = time.time()
```

```
print(f"Training time: {stop - start}s")



start1 = time.time()
RF4=clf4.predict(X_test)
stop1 = time.time()
print(f"Predicting time: {stop1 - start1}s")
#toc = time()
#t1=time()
#print("Predicting time:", round(time()-t0, 3), "s")
accRF=clf4.score(X_test,Y_test)

print(accRF,'Random Forest')
print(classification_report(Y_test,RF4))



# In[52]:


from sklearn.metrics import precision_score, recall_score, f1_score

# Assuming Y_test are the true labels and RF are the predicted labels
precision = precision_score(Y_test, RF4)
recall = recall_score(Y_test, RF4)
f1 = f1_score(Y_test, RF4)

# Display the results
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
```

```
# In[53]:


def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Purples):
 """
 This function prints and plots the confusion matrix.
 Normalization can be applied by setting `normalize=True`.
 """
 if normalize:
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    print("Normalized confusion matrix")
 else:
    print('Confusion matrix, without normalization')

 print(cm)

 plt.imshow(cm, interpolation='nearest', cmap=cmap)
 plt.title(title)
 plt.colorbar()
 tick_marks = np.arange(len(classes))
 plt.xticks(tick_marks, classes, rotation=0)
 plt.yticks(tick_marks, classes)

 fmt = '.2f' if normalize else 'd'
```

```
 thresh = cm.max() / 2.
 for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
     plt.text(j, i, format(cm[i, j], fmt),
              horizontalalignment="center",
              color="white" if cm[i, j] > thresh else "black")

 plt.tight_layout()
 plt.ylabel('True label')
 plt.xlabel('Predicted label')
# Compute confusion matrix
from sklearn.metrics import confusion_matrix
cn_matrix = confusion_matrix(Y_test, RF4)
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plt.figure()
classs=['2-as-Benigen','4-as-Malignant']
plot_confusion_matrix(cn_matrix, classes=classs,normalize=False,
                      title='Confusion matrix,RF-RFE-WDBC')
plt.show()
classs=['2-as-Benigen','4-as-Malignant']
plot_confusion_matrix(cn_matrix, classes=classs,normalize=True,
                      title='Confusion matrix,RF-RFE-WDBC')
plt.show()



# # PCA


# In[54]:
```

```
from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.metrics import accuracy_score, confusion_matrix

#from sklearn.neighbors import KNeighborsClassifier,NeighborhoodComponentsAnalysis,
LocalOutlierFactor

from sklearn.decomposition import PCA

from sklearn.preprocessing import MinMaxScaler

scaler = StandardScaler()

sc = MinMaxScaler()

x_scaled = scaler.fit_transform(X)


pca = PCA(n_components=2)

pca.fit(x_scaled)

X_reduced_pca = pca.transform(x_scaled)

pca_data = pd.DataFrame(X_reduced_pca, columns=["p1", "p2"])

pca_data["target"] = Y

plt.figure(figsize=(10,8))


sns.scatterplot(x="p1", y="p2" , hue="target", data=pca_data,palette=['green','red'], legend='full')

plt.grid()

plt.title("WDBC data PCA=2: PC1 vs PC2 ")



import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.decomposition import PCA

from sklearn.preprocessing import MinMaxScaler
```

```
from mpl_toolkits.mplot3d import Axes3D
import pandas as pd

# Assuming Xil and Yil are defined earlier in your code
scaler = MinMaxScaler()
x_scaled = scaler.fit_transform(X)

# Apply PCA to reduce to 3 components
pca = PCA(n_components=3, random_state=42)  # Set components to 3 for 3D projection
pca.fit(x_scaled)
X_reduced_pca = pca.transform(x_scaled)  # Transform data into 3D

# Create a DataFrame for the reduced PCA components and add the target labels
pca_data = pd.DataFrame(X_reduced_pca, columns=["p1", "p2", "p3"])
pca_data["target"] = Y

# Create a 3D scatter plot
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# Scatter plot in 3D with color based on target class
scatter = ax.scatter(pca_data["p1"], pca_data["p2"], pca_data["p3"], c=pca_data["target"],
cmap='viridis')

# Adding axis labels and title
ax.set_xlabel('p1')
ax.set_ylabel('p2')
ax.set_zlabel('p3')
ax.set_title("PCA with three components p1, p2, p3 with MinMax scaling for WDBC")
```

```
# Show the plot
plt.show()



from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X_reduced_pca, Y, test_size = 0.2)



from sklearn.ensemble import RandomForestClassifier
import time
#tic = time()

start = time.time()
clf2 = RandomForestClassifier(criterion='gini', max_depth=100, random_state=42)

#t0=time()
#print("training time:", round(time()-t0, 3), "s")
clf2.fit(X_train, Y_train)
stop = time.time()
print(f"Training time: {stop - start}s")



start1 = time.time()
RF2=clf2.predict(X_test)
stop1 = time.time()
print(f"Predicting time: {stop1 - start1}s")
#toc = time()
#t1=time()
```

```
#print("Predicting time:", round(time()-t0, 3), "s")
accRF=clf2.score(X_test,Y_test)

print(accRF,'Random Forest')
print(classification_report(Y_test,RF2))



Cohen_kappa = cohen_kappa_score(Y_test,RF2)
print(" Cohen_kappa:", Cohen_kappa)

from sklearn.metrics import precision_score, recall_score, f1_score

# Assuming Y_test are the true labels and RF are the predicted labels
precision = precision_score(Y_test, RF2)
recall = recall_score(Y_test, RF2)
f1 = f1_score(Y_test, RF2)

# Display the results
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")



MCC = matthews_corrcoef(Y_test,RF2)
print('MCC:', MCC)



# In[65]:
```

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Greens):
 """
 This function prints and plots the confusion matrix.
 Normalization can be applied by setting `normalize=True`.
 """
 if normalize:
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    print("Normalized confusion matrix")
 else:
    print('Confusion matrix, without normalization')

 print(cm)

 plt.imshow(cm, interpolation='nearest', cmap=cmap)
 plt.title(title)
 plt.colorbar()
 tick_marks = np.arange(len(classes))
 plt.xticks(tick_marks, classes, rotation=0)
 plt.yticks(tick_marks, classes)

 fmt = '.2f' if normalize else 'd'
 thresh = cm.max() / 2.
 for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, format(cm[i, j], fmt),
             horizontalalignment="center",
```

```
            color="white" if cm[i, j] > thresh else "black")

 plt.tight_layout()
 plt.ylabel('True label')
 plt.xlabel('Predicted label')
# Compute confusion matrix
from sklearn.metrics import confusion_matrix
cn_matrix = confusion_matrix(Y_test, RF2)
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plt.figure()
classs=['2-as-Benigen','4-as-Malignant']
plot_confusion_matrix(cn_matrix, classes=classs,normalize=False,
                      title='Confusion matrix,RF-PCA-WDBC')
plt.show()
classs=['2-as-Benigen','4-as-Malignant']
plot_confusion_matrix(cn_matrix, classes=classs,normalize=True,
                      title='Confusion matrix,RF-PCA-WDBC')
plt.show()



# # FACTOR ANALYSIS

# In[66]:



import matplotlib.pyplot as plt
import seaborn as sns
```

```
from matplotlib.colors import ListedColormap

from sklearn.decomposition import FactorAnalysis

from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.metrics import accuracy_score, confusion_matrix

#from sklearn.neighbors import KNeighborsClassifier,NeighborhoodComponentsAnalysis,
LocalOutlierFactor

from sklearn.decomposition import PCA, TruncatedSVD

from sklearn.preprocessing import MinMaxScaler, StandardScaler

scaler = MinMaxScaler()

x_scaled = scaler.fit_transform(X)


fa = FactorAnalysis(n_components=2, random_state=42)

fa.fit(x_scaled)

X_reduced_fa = fa.transform(x_scaled)

fa_data = pd.DataFrame(X_reduced_fa, columns=["p1", "p2"])

fa_data["target"] = Y

plt.figure(figsize=(10,8))


sns.scatterplot(x="p1", y="p2" , hue="target", data=fa_data,palette=['orange','blue'], legend='full')


plt.title("Factor Analysis with two components p1 vs p2 with MinMax scaling WDBC ")


import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.decomposition import FactorAnalysis

from sklearn.preprocessing import MinMaxScaler

from mpl_toolkits.mplot3d import Axes3D

import pandas as pd
```

```
# Assuming Xil and Yil are defined earlier in your code
scaler = MinMaxScaler()
x_scaled = scaler.fit_transform(X)

fa = FactorAnalysis(n_components=3, random_state=42)  # Set components to 3 for 3D
fa.fit(x_scaled)
X_reduced_fa = fa.transform(x_scaled)
fa_data = pd.DataFrame(X_reduced_fa, columns=["p1", "p2", "p3"])
fa_data["target"] = Y

# Creating a 3D plot
fig = plt.figure(figsize=(10,8))
ax = fig.add_subplot(111, projection='3d')

# Scatter plot in 3D
scatter = ax.scatter(fa_data["p1"], fa_data["p2"], fa_data["p3"], c=fa_data["target"], cmap='plasma')

# Adding labels
ax.set_xlabel('p1')
ax.set_ylabel('p2')
ax.set_zlabel('p3')
ax.set_title("Factor Analysis with three components p1, p2, p3 with MinMax scaling WDBC")

# Show color bar
plt.colorbar(scatter)

plt.show()
```

In[68]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X_reduced_fa, Y, test_size = 0.2)
```

In[74]:

```
from sklearn.ensemble import RandomForestClassifier
import time
#tic = time()

start = time.time()
clf1 = RandomForestClassifier(criterion='gini', max_depth=100, random_state=42)

#t0=time()
#print("training time:", round(time()-t0, 3), "s")
clf1.fit(X_train, Y_train)
stop = time.time()
print(f"Training time: {stop - start}s")


start1 = time.time()
RF=clf1.predict(X_test)
stop1 = time.time()
print(f"Predicting time: {stop1 - start1}s")
```

```
#toc = time()
#t1=time()
#print("Predicting time:", round(time()-t0, 3), "s")
accRF=clf1.score(X_test,Y_test)

print(accRF,'Random Forest')
print(classification_report(Y_test,RF))


# In[75]:


from sklearn.metrics import hamming_loss
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import matthews_corrcoef
from sklearn.metrics import zero_one_loss


Cohen_kappa = cohen_kappa_score(Y_test,RF)
print(" Cohen_kappa:", Cohen_kappa)

from sklearn.metrics import precision_score, recall_score, f1_score

# Assuming Y_test are the true labels and RF are the predicted labels
precision = precision_score(Y_test, RF)
recall = recall_score(Y_test, RF)
f1 = f1_score(Y_test, RF)
```

```
# Display the results
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")


def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues, fontsize= 40):
 """
 This function prints and plots the confusion matrix.
 Normalization can be applied by setting `normalize=True`.
 """
 if normalize:
     cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
     print("Normalized confusion matrix")
 else:
     print('Confusion matrix, without normalization')


 print(cm)


 plt.imshow(cm, interpolation='nearest', cmap=cmap)
 plt.title(title)
 plt.colorbar()
 tick_marks = np.arange(len(classes))
 plt.xticks(tick_marks, classes, rotation=0)
 plt.yticks(tick_marks, classes)
```

```
 fmt = '.2f' if normalize else 'd'
 thresh = cm.max() / 2.
 for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
     plt.text(j, i, format(cm[i, j], fmt),
              horizontalalignment="center",
              color="white" if cm[i, j] > thresh else "black")

 plt.tight_layout()
 plt.ylabel('True label')
 plt.xlabel('Predicted label')
# Compute confusion matrix
from sklearn.metrics import confusion_matrix
cn_matrix = confusion_matrix(Y_test, RF)
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plt.figure()
classs=['2-as-Benigen','4-as-Malignant']
plot_confusion_matrix(cn_matrix, classes=classs,normalize=False,
                      title='Confusion matrix,RF-FA-WDBC',fontsize=40)
plt.show()
classs=['2-as-Benigen','4-as-Malignant']
plot_confusion_matrix(cn_matrix, classes=classs,normalize=True,
                      title='Confusion matrix,RF-FA-WDBC',fontsize=40)
plt.show()



from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, confusion_matrix
```

```
from sklearn.decomposition import PCA
from sklearn.feature_selection import RFE
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np

# Assuming Xil and Yil are defined

# Step 1: Scale the data using StandardScaler
scaler = StandardScaler()
x_scaled = scaler.fit_transform(X)

# Step 2: Apply PCA (increase the number of components)
pca = PCA(n_components=5)  # Increased to 5 components
X_reduced_pca = pca.fit_transform(x_scaled)

# Visualize explained variance ratio (optional)
print("Explained variance ratio:", pca.explained_variance_ratio_)

# Step 3: Apply RFE to the PCA-transformed data
estimator = SVR(kernel="linear")
selector = RFE(estimator, n_features_to_select=5)  # Select 5 features
selector.fit(X_reduced_pca, Y)

# Get the selected features
selected_features = selector.support_  # This is a boolean array
print("Selected features:", selected_features)
```

```
# Prepare the data with the selected features
x_selected = X_reduced_pca[:, selected_features]

# Step 4: Split the data into training and test sets
X_train, X_test, Y_train, Y_test = train_test_split(x_selected, Y, test_size=0.2, random_state=123)

# Step 5: Train a Random Forest Classifier on the selected features
# Tune the hyperparameters using GridSearchCV
param_grid = {
   'n_estimators': [100, 200, 300],
   'max_depth': [10, 15, 20, None],
   'min_samples_split': [2, 5, 10],
   'min_samples_leaf': [1, 2, 4]
}

rf_classifier = RandomForestClassifier(random_state=123)
grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid, cv=5, n_jobs=-1,
verbose=2)
grid_search.fit(X_train, Y_train)

# Best parameters found from grid search
print(f"Best parameters: {grid_search.best_params_}")

# Use the best estimator from the grid search
best_rf_classifier = grid_search.best_estimator_

# Step 6: Make predictions
Y_pred = best_rf_classifier.predict(X_test)
```

```
# Step 7: Evaluate the model using accuracy score and confusion matrix
accuracy = accuracy_score(Y_test, Y_pred)
print(f"Accuracy of Random Forest Classifier on selected features: {accuracy:.4f}")

# Confusion Matrix
cm = confusion_matrix(Y_test, Y_pred)
print("Confusion Matrix:")
print(cm)



# In[80]:



from sklearn.ensemble import RandomForestClassifier
import time
#tic = time()

start = time.time()
#clf4 = RandomForestClassifier(criterion='gini', max_depth=10, random_state=123)

#t0=time()
#print("training time:", round(time()-t0, 3), "s")
best_rf_classifier.fit(X_train, Y_train)
stop = time.time()
print(f"Training time: {stop - start}s")



start1 = time.time()
```

```
RF5=best_rf_classifier.predict(X_test)
stop1 = time.time()
print(f"Predicting time: {stop1 - start1}s")
#toc = time()
#t1=time()
#print("Predicting time:", round(time()-t0, 3), "s")
accRF=best_rf_classifier.score(X_test,Y_test)

print(accRF,'Random Forest')
print(classification_report(Y_test,RF5))



# In[81]:



Cohen_kappa = cohen_kappa_score(Y_test,RF5)
print(" Cohen_kappa:", Cohen_kappa)

from sklearn.metrics import precision_score, recall_score, f1_score

# Assuming Y_test are the true labels and RF are the predicted labels
precision = precision_score(Y_test, RF5)
recall = recall_score(Y_test, RF5)
f1 = f1_score(Y_test, RF5)

# Display the results
print(f"Precision: {precision}")
print(f"Recall: {recall}")
```

```
print(f"F1 Score: {f1}")


# In[86]:


def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.cool):
 """
 This function prints and plots the confusion matrix.
 Normalization can be applied by setting `normalize=True`.
 """
 if normalize:
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    print("Normalized confusion matrix")
 else:
    print('Confusion matrix, without normalization')


 print(cm)


 plt.imshow(cm, interpolation='nearest', cmap=cmap)
 plt.title(title)
 plt.colorbar()
 tick_marks = np.arange(len(classes))
 plt.xticks(tick_marks, classes, rotation=0)
 plt.yticks(tick_marks, classes)


 fmt = '.2f' if normalize else 'd'
 thresh = cm.max() / 2.
```

```
 for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
     plt.text(j, i, format(cm[i, j], fmt),
             horizontalalignment="center",
             color="white" if cm[i, j] > thresh else "black")


 plt.tight_layout()
 plt.ylabel('True label')
 plt.xlabel('Predicted label')
# Compute confusion matrix
from sklearn.metrics import confusion_matrix
cn_matrix = confusion_matrix(Y_test, RF5)
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plt.figure()
classs=['2-as-Benigen','4-as-Malignant']
plot_confusion_matrix(cn_matrix, classes=classs,normalize=False,
                      title='Confusion matrix,Mean-PCA-RFE-RF-WDBC')
plt.show()
classs=['2-as-Benigen','4-as-Malignant']
plot_confusion_matrix(cn_matrix, classes=classs,normalize=True,
                      title='Confusion matrix,Mean-PCA-RFE-RF-WDBC')
plt.show()
```