**BREAST CANCER DETECTION USING MACHINE LEARNING**

**DONE BY :**

Team-4

|  |
| --- |
| Index |
| **Introduction** |
| **Methodology :** |
| **Data cleaning and preprocessing** |
| **Identifying and Dropping unnecessary columns** |
| **Identifying and treating null values** |
| **Identifying the unique values in each column** |
| **Identifying and treating the duplicate rows** |
| **Identifying and handling outliers** |
| **Data visualization** |
| **Data scaling** |
| **Handling with smote** |
| **Implementing Ada-boost classifier** |
| **Hyperparameter tuning** |
| **Selecting the important features** |
| **Retraining the Ada-boost with selected features** |
| **Hyperparameter tuning for the best model** |
| **Results** |
| **Confusion Matrix** |
| **Classification Report** |
| **ROC Curve** |
| **Streamlit application** |
| **Conclusion** |

# 1. Introduction :

## What is Breast Cancer?

Breast cancer involves uncontrolled cell growth within the breast, leading to the formation of a lump or mass, commonly referred to as a tumor. If the tumor is malignant, it has the potential to invade surrounding tissues or spread to other parts of the body. Detecting this disease at an early stage is essential to ensure effective treatment options.

## Types of Breast Cancer

1. Malignant (cancerous): This type of tumor grows aggressively and can spread to other organs if not treated.  
   2. Benign (non-cancerous): These tumors generally do not spread to other tissues but may need monitoring or removal if they grow.

## Importance of Early Detection

Early diagnosis of breast cancer is crucial, as it significantly influences the chances of successful treatment. Early-stage cancers are typically easier to manage, offering a more favorable prognosis. Common methods for early detection include mammograms, ultrasounds, biopsies, and lab tests. Routine self-exams and professional checkups are also highly recommended.

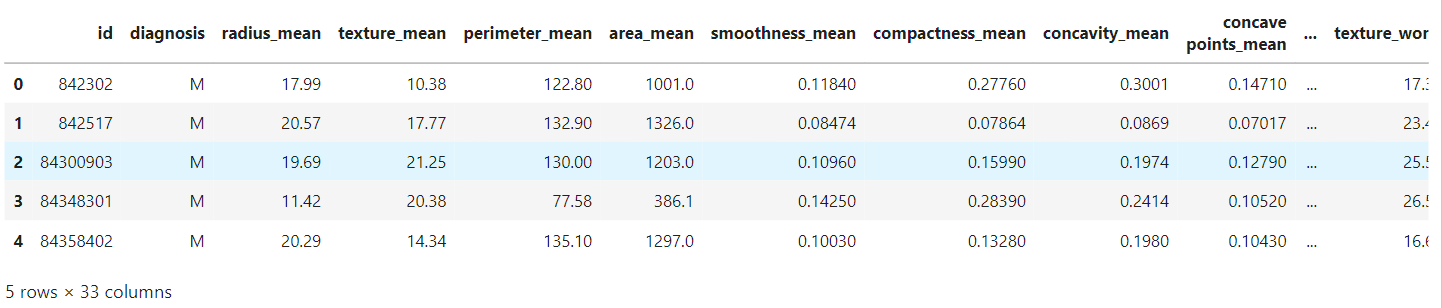
## Machine Learning in Medical Diagnosis

Machine learning applications are revolutionizing healthcare by allowing algorithms to process and analyze vast amounts of medical data with unprecedented speed and accuracy. These advanced systems can identify complex patterns and correlations that might be difficult for humans to detect. For instance, in breast cancer diagnosis, machine learning models assist doctors by examining mammogram images and patient histories to detect subtle indicators of cancer. By integrating this technology, healthcare providers can improve early detection rates, enabling timely interventions and personalized treatment plans. This not only enhances diagnostic precision but also contributes to better patient outcomes

Examples of machine learning models used in breast cancer diagnosis include:  
1. Classification Models: Used to classify tumors as malignant or benign.  
2. Predictive Models: Used to assess the risk of cancer recurrence or predict a patient’s survival likelihood based on specific features.

## Manual examination of data :

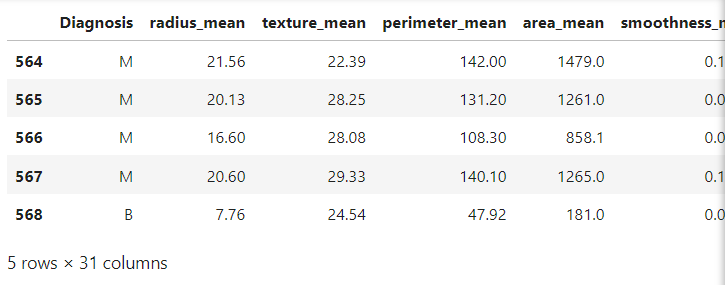
The data consists of 569 rows and 32 columns



# Data cleaning and preprocessing:

## Identifying and Dropping unnecessary columns:

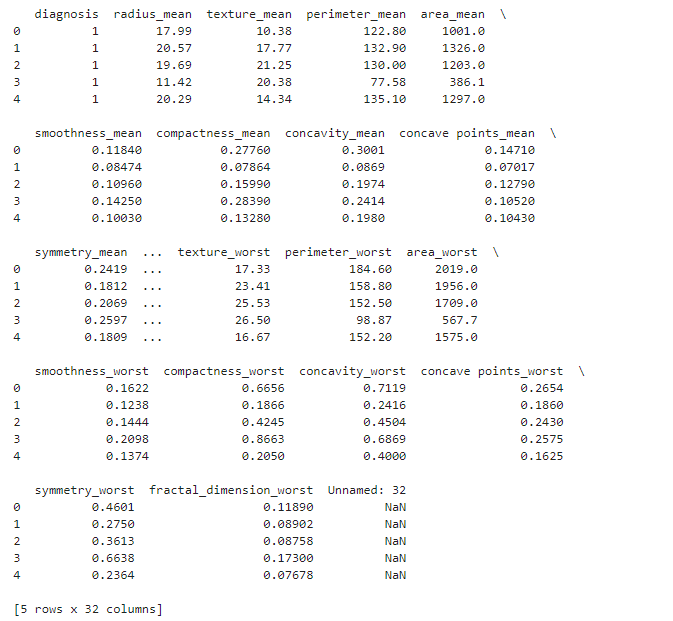
Identifying and dropping unnecessary columns helps focus on the most relevant features for accurate predictions. Columns with irrelevant information, redundant data, or constant values are removed to simplify the dataset and reduce computational complexity. This ensures the model works efficiently and improves its performance.



## Identifying and treating null values:

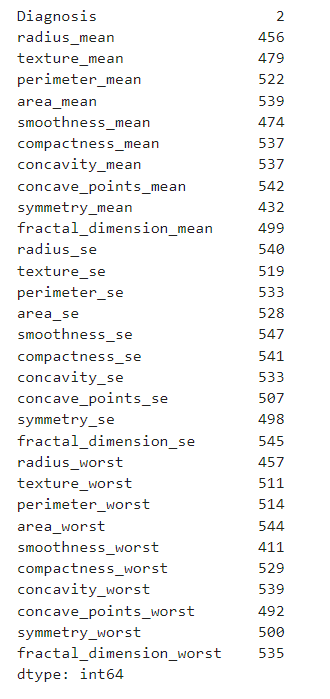
Null values can arise due to missing data in patient records, leading to inaccuracies in model predictions. These values are first identified and analyzed, then treated using techniques such as replacing them with the mean, median, or mode, or by applying more advanced methods like interpolation. Proper handling of null values ensures a clean, reliable dataset for building an accurate and robust detection model.

**The first few rows of the cleaned dataset:**



## Identifying the unique values in each column:

This process involves checking each column for distinct values or categories, which helps in understanding the distribution of data. For instance, in columns like tumor type or diagnosis, identifying unique values (e.g., malignant or benign) ensures that the model can correctly interpret these categories. It also helps in detecting any inconsistencies or errors in the data, such as unexpected values, which can be addressed before further analysis or modeling.



## Identifying and treating the duplicate rows:

Identifying and treating duplicate rows typically involves checking for any repeated data entries that may have been mistakenly included. Even though you mention that there are no duplicate rows in your dataset, this step is important to ensure data integrity. If any duplicates were present, they would need to be removed to prevent bias and over-fitting during model training. However, since there are no duplicates in your case, this step can be skipped, ensuring that the dataset is clean and unique for accurate analysis and predictions.



## Identifying and handling outliers:

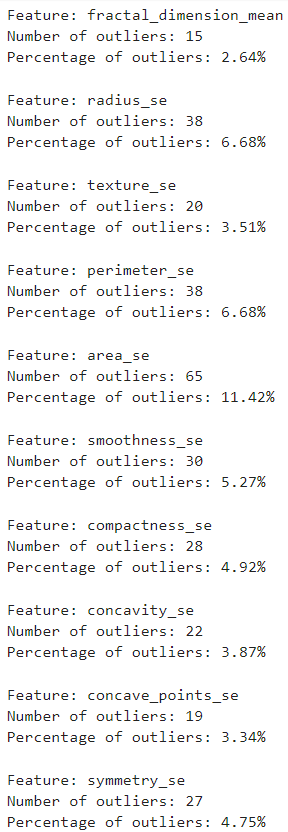
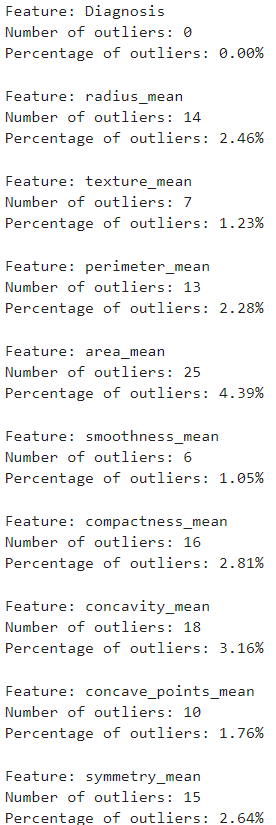
Outliers are data points that deviate significantly from the other values in a dataset, which could skew model results and affect accuracy. To identify outliers, statistical methods such as the Z-score or IQR (Interquartile Range) can be used. Once identified, outliers can be handled in different ways depending on their nature: they can be removed if they are errors, transformed if they are valid but extreme values, or capped to a reasonable range. Proper handling of outliers ensures that the model focuses on representative data, leading to better and more reliable predictions.

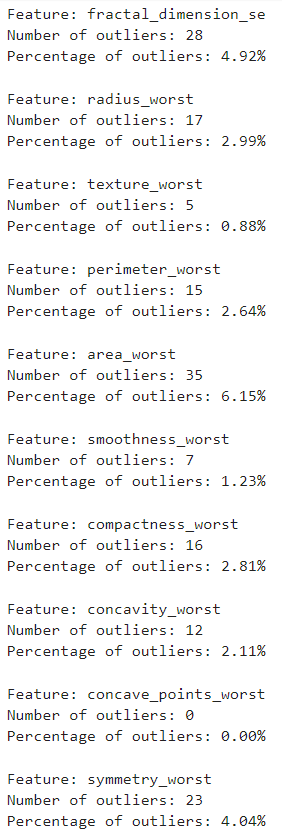
The code is aimed at **identifying and analyzing outliers** in the dataset. Here's what has been done:

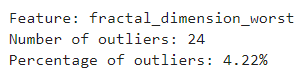
**Objective**: For each feature in the dataset, the code calculates the percentage of outliers based on the Interquartile Range (IQR) method.

**Steps Performed**:

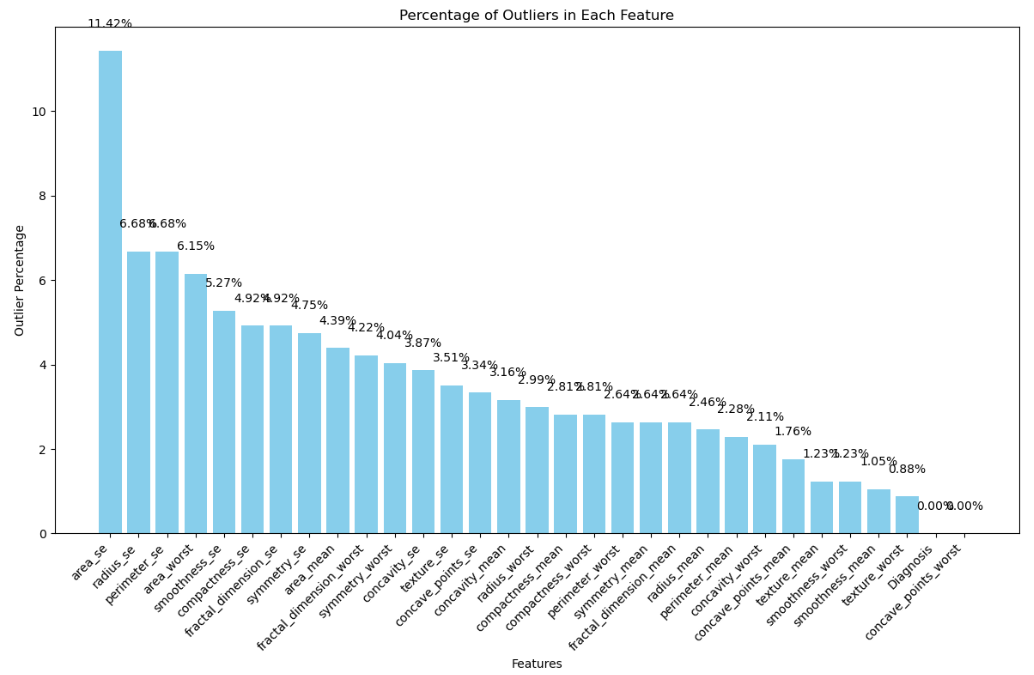
* 1. **IQR Calculation**: The first quartile (Q1) and third quartile (Q3) values are computed for each feature, and the IQR is determined as Q3−Q1Q3 - Q1Q3−Q1.
  2. **Outlier Bounds**: Lower and upper bounds are defined as Q1−1.5×IQRQ1 - 1.5 , IQRQ1−1.5×IQR and Q3+1.5×IQRQ3 + 1.5 ,IQRQ3+1.5×IQR, respectively.
  3. **Outlier Identification**: Data points that fall outside these bounds are identified as outliers.
  4. **Outlier Percentage**: For each feature, the percentage of outliers is calculated relative to the total number of data points.
  5. **Tracking Total Outliers**: The total number of outliers across all features is tracked and printed.







## Graph for outliers for all the features:

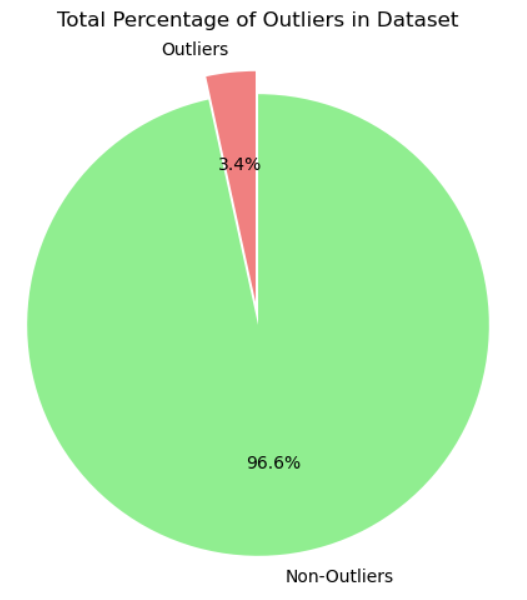


The total percentage of outliers across all features in the dataset, providing an overall measure of the presence of outliers. It uses the previously computed total number of outliers and normalizes it by the total number of data points across all features to get this aggregate percentage.

To better understand the distribution of outliers feature-wise, the outlier percentages for individual features are organized into a DataFrame. This DataFrame is sorted in descending order to highlight the features with the highest proportion of outliers.

For visualization, a bar chart is generated to display the percentage of outliers for each feature. The x-axis represents the features, while the y-axis shows their respective outlier percentages. To enhance readability, the x-axis labels are rotated, and each bar is annotated with its percentage value, making the chart more informative. This visualization provides a clear understanding of how outliers are distributed across different features, helping identify those that may require special attention during data preprocessing.

## Total outlier percentage using piechart:



visualizing the overall proportion of outliers in the dataset using a pie chart. The pie chart represents the dataset as two segments: one for outliers and another for non-outliers. The sizes of these segments are calculated based on the total outlier percentage and the complement percentage for non-outliers.

To make the chart visually engaging, the "Outliers" segment is highlighted by slightly exploding it outward, and distinct colors are used for differentiation—light coral for outliers and light green for non-outliers. The percentages are displayed on the chart for both segments, offering a quick and intuitive understanding of the data composition.

This visualization effectively conveys the extent of outliers in the dataset relative to the total data, helping to assess the impact of outliers on the dataset's quality and the potential need for mitigation.

## Decision on Outlier Handling

After analyzing the dataset, it was observed that the percentage of outliers is minimal, accounting for only 3.4% of the total data. Given this relatively small proportion, the decision was made to neglect the outliers in the dataset instead of applying additional preprocessing or removal steps. This choice simplifies the workflow while ensuring the majority of the data remains unaffected. With this consideration, the model-building process proceeds using the full dataset, focusing on capturing patterns in the larger, representative portion of the data.

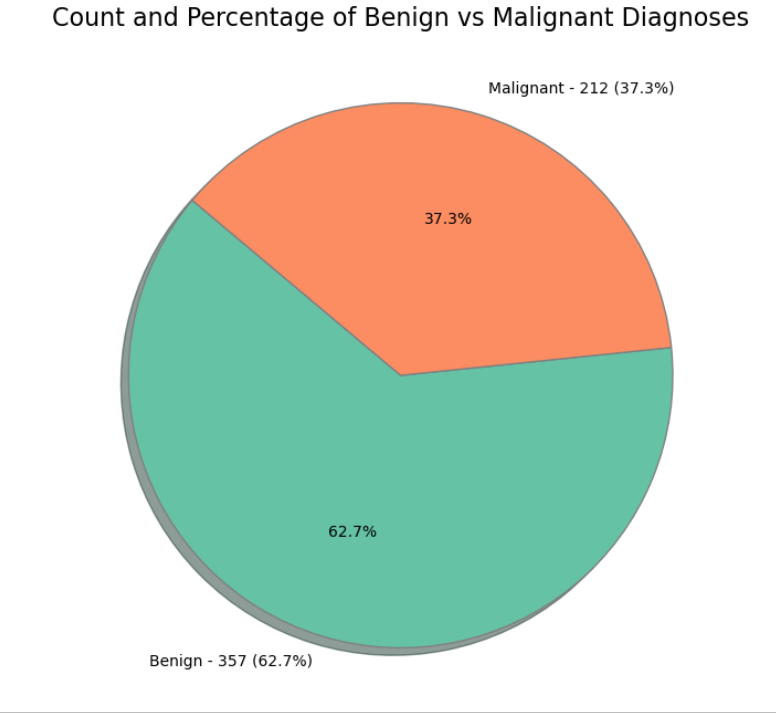
**Data Cleaning and Preprocessing Summary:**

Data cleaning and preprocessing have ensured that the dataset for breast cancer detection is free of irrelevant, missing, and inconsistent data. By addressing null values, removing unnecessary columns, and handling outliers, we’ve prepared a clean and reliable dataset that enhances model accuracy and performance.

# Data visualization:

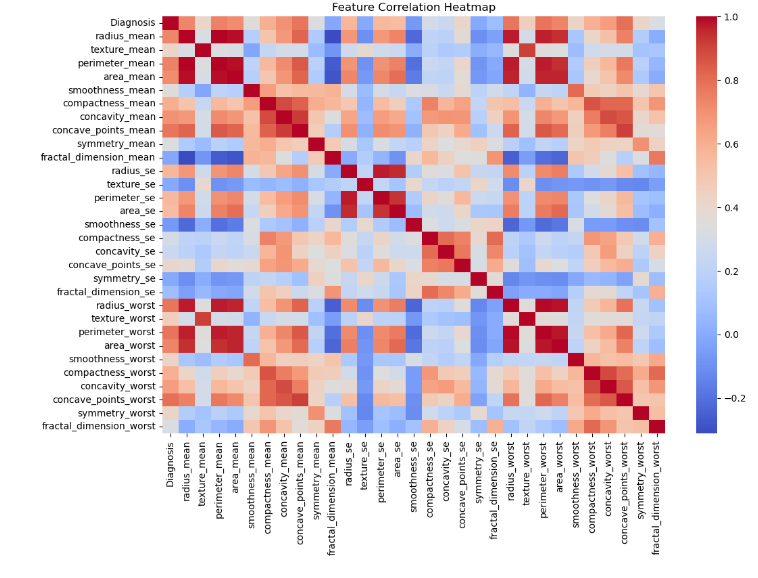
Data visualization helps in understanding patterns and relationships within the breast cancer detection dataset by using charts, graphs, and plots. It makes it easier to identify key insights, such as the distribution of benign vs. malignant cases and feature correlations. This step improves data exploration and effectively communicates findings to stakeholders, ensuring a clear understanding of the data.

The count and percentage of benign vs malignant diagnoses provide an overview of the dataset's distribution. This information helps to understand the balance of cases and can guide the model in addressing any class imbalances.



**Feature Correlation Heatmap:**

A feature correlation heatmap visually represents the relationships between different features in the dataset. It helps identify highly correlated features, which can inform decisions about feature selection and model performance.

****

### Feature Scaling

To ensure uniformity in feature distributions, the original dataset features are scaled using the StandardScaler from the sklearn library. This step standardizes the data by centering it around the mean with unit variance, which is essential for optimizing the performance of machine learning models, especially those sensitive to feature magnitudes, such as AdaBoost.

### Train-Test Split

The dataset is split into training and testing sets using a 70-30 ratio, with 70% of the data used for training the model and 30% reserved for testing its performance. This split ensures that the model is evaluated on unseen data, providing a more accurate assessment of its generalization capabilities. The random\_state parameter is set to 42 for reproducibility of the results.

# Class Imbalance Handling with SMOTE

To address class imbalance in the dataset, the Synthetic Minority Over-sampling Technique (SMOTE) is applied to the training data. SMOTE generates synthetic examples of the minority class by interpolating between existing instances, which helps balance the number of examples in each class. This process improves the model's ability to learn patterns from the minority class, enhancing overall classification performance. By resampling the training data with SMOTE, we ensure that the model is not biased toward the majority class and can generalize better to both classes.

In the code, we initialize the SMOTE technique using the following line:

smote = SMOTE(random\_state=42)

Here, random\_state=42 ensures that the sampling process is reproducible. We then apply SMOTE to the training data:

X\_train\_smote, y\_train\_smote = smote.fit\_resample(X\_train, y\_train)

This step generates a balanced training dataset (X\_train\_smote, y\_train\_smote) by resampling the original training data (X\_train, y\_train). The resampled dataset is now ready for model training, with a more balanced representation of both classes.

# Implementation of Ada-boost classifier:

The code implements an **AdaBoost Classifier** using the SAMME algorithm, which is an ensemble method that combines multiple weak learners to create a strong classifier. The model is trained on the original dataset . AdaBoost works by assigning higher weights to misclassified samples during training, helping the model focus more on difficult cases in subsequent iterations.

After training, the model makes predictions on the test set, and its performance is evaluated using the classification\_report(). This report includes metrics like precision, recall, F1-score, and support for both the benign and malignant classes. These metrics help assess how well the model distinguishes between the two classes and handles imbalances in the dataset, providing valuable insights into its effectiveness for breast cancer detection.

In this step, the AdaBoost classifier is initialized and trained on the resampled data. To create the AdaBoost model, a **DecisionTreeClassifier** with a maximum depth of 1 is used as the **base estimator**. This weak learner is chosen because of its simplicity and effectiveness when combined in an ensemble method like AdaBoost.

The AdaBoost model is initialized with the following code:

base\_estimator = DecisionTreeClassifier(max\_depth=1)

ada\_boost = AdaBoostClassifier(estimator=base\_estimator, random\_state=42)

Here, the random\_state=42 ensures reproducibility of the results. Next, the AdaBoost model is trained on the SMOTE-resampled training data (X\_train\_smote, y\_train\_smote) using the .fit() method:

ada\_boost.fit(X\_train\_smote, y\_train\_smote)

After training, the model is used to predict the target labels on the test set (X\_test) with the .predict() method:

y\_pred = ada\_boost.predict(X\_test)

This step allows the model to make predictions on the unseen test data, which will be evaluated in the next stages of the project.

## Classification Report (Without Hyperparameter Tuning):

After training the AdaBoost classifier, the model's performance is evaluated using the **classification report**. This report provides key metrics such as precision, recall, F1-score, and accuracy for each class in the dataset. These metrics help assess how well the model distinguishes between malignant and benign cases.

The code for generating the classification report is as follows:

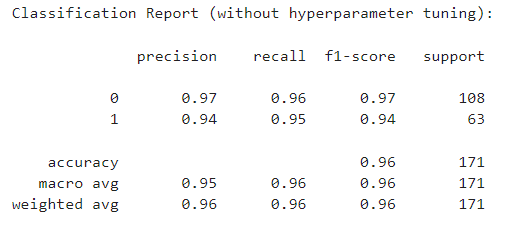
print("Classification Report (without hyperparameter tuning):\n")

print(classification\_report(y\_test, y\_pred))

The classification\_report function from sklearn.metrics compares the true labels (y\_test) with the predicted labels (y\_pred). It computes the precision, recall, F1-score, and support for each class (malignant and benign). This evaluation is crucial for understanding the model's strengths and weaknesses in classifying the test data before any hyperparameter tuning is applied.

Classification report:

The classification report summarizes the model’s precision, recall, F1-score, and support. It allows for an easy assessment of how well the model performs in terms of both identifying true positives and minimizing false positives, which is essential when working with imbalanced datasets.



# Hyperparameter Tuning with GridSearchCV:

To optimize the performance of the AdaBoost classifier, hyperparameter tuning is performed using **GridSearchCV**. This technique systematically evaluates a range of hyperparameters to find the combination that yields the best model performance.

The hyperparameters being tuned in this case are:

* n\_estimators: The number of weak learners (decision trees) to include in the AdaBoost ensemble, with values of 50, 100, and 150.
* learning\_rate: The rate at which the model learns, with values of 0.1, 0.5, and 1.0.

Here, param\_grid defines the grid of hyperparameters to search. Next, the GridSearchCV object is initialized with the AdaBoost model and the parameter grid, along with the following specifications:

* cv=5: 5-fold cross-validation to assess the model’s performance across different splits of the data.
* scoring='recall': The evaluation metric is set to recall, which is particularly useful for imbalanced datasets.
* n\_jobs=-1: This ensures that all available CPU cores are used to perform the cross-validation efficiently.

GridSearchCV is then fitted to the resampled training data:

grid\_search.fit(X\_train\_smote, y\_train\_smote)

This step searches for the best combination of hyperparameters and trains the AdaBoost model using the selected configuration. The results are evaluated using recall to prioritize correctly identifying positive cases, particularly in an imbalanced dataset.

## Evaluating the Best Model from Hyperparameter Tuning:

After performing hyperparameter tuning using **GridSearchCV**, the best model with the optimal hyperparameters is selected and evaluated on the test set. The best model is extracted from the GridSearchCV results using the following code:

best\_adaboost = grid\_search.best\_estimator\_

This model represents the AdaBoost classifier with the hyperparameters that achieved the highest recall during cross-validation. The best hyperparameters found during the search are printed with:

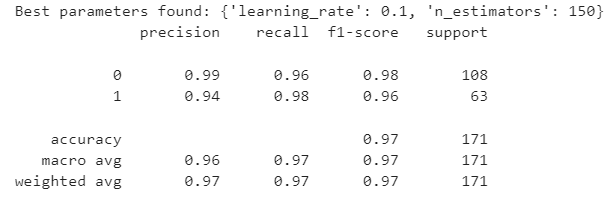
print(f"Best parameters found: {grid\_search.best\_params\_}")

After selecting the best model, it is used to make predictions on the test set (X\_test) using the .predict() method:

y\_pred = best\_adaboost.predict(X\_test)

This step allows the newly tuned AdaBoost model to generate predictions, which can be further evaluated for its performance. The improvements in classification accuracy and recall after hyperparameter tuning will be assessed in the following evaluation stages.

## Classification Report (With Hyperparameter Tuning):



# Model Evaluation - ROC Curve for Best AdaBoost Model

To further evaluate the performance of the best AdaBoost model, the **Receiver Operating Characteristic (ROC) curve** is plotted. The ROC curve is a valuable tool for assessing the tradeoff between the **true positive rate (sensitivity)** and **false positive rate (1 - specificity)** at various thresholds, helping to understand the model's classification performance across different decision boundaries.

The first step in the process is to predict the probabilities for the test set using the .predict\_proba() method:

y\_pred\_proba = best\_adaboost.predict\_proba(X\_test)

This returns the predicted probabilities for both classes (benign and malignant). The probabilities for the positive class (malignant) are used to compute the ROC curve.

The **ROC curve** is then generated by calculating the false positive rate (fpr) and true positive rate (tpr) using the roc\_curve function:

fpr, tpr, \_ = roc\_curve(y\_test, y\_pred\_proba[:, 1])

The area under the ROC curve (AUC) is calculated with the auc() function:

roc\_auc = auc(fpr, tpr)

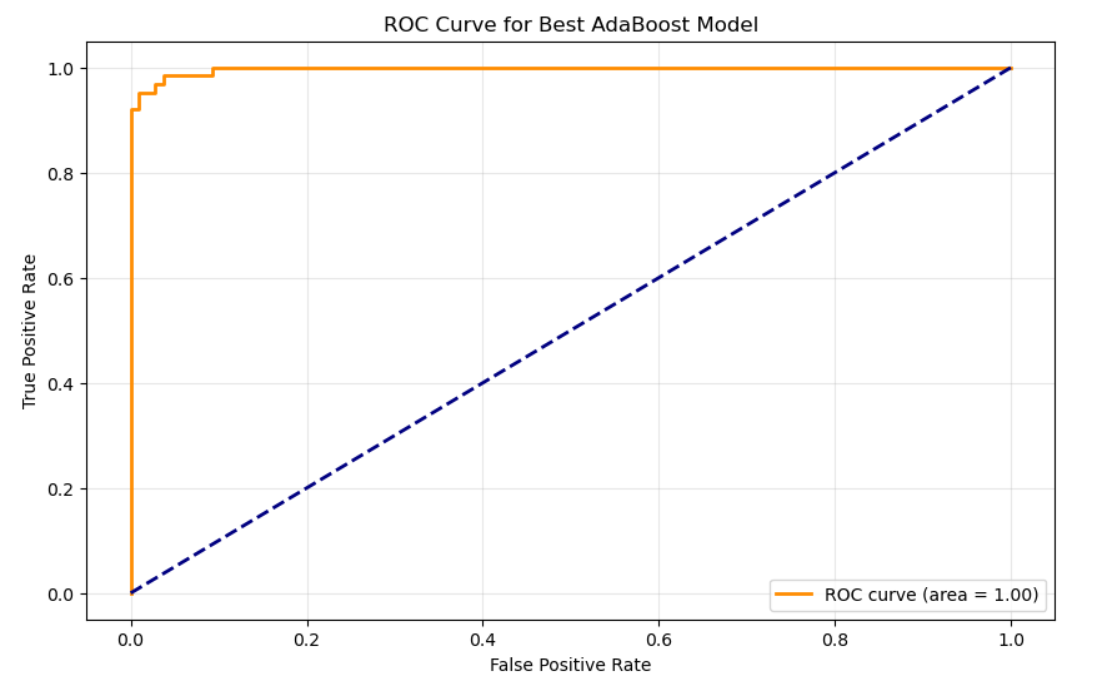
The AUC provides a summary measure of the model’s ability to distinguish between the classes, with values closer to 1 indicating better performance.

The ROC curve is plotted with:

plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc\_auc:.2f})')

A diagonal line representing a random classifier (no discrimination) is also added for reference:

Finally, the plot is displayed with labeled axes, a title, and a legend, providing a clear visual of the model's classification performance.



# **Feature** Importance Analysis:

To gain insights into which features are most influential in predicting the target variable, **feature importance** is calculated. This step is crucial because it allows us to identify the variables that contribute most to the model’s decision-making process, which can provide valuable insights into the underlying patterns of the data.

In AdaBoost, feature importance is determined by how much each feature improves the model's performance across all iterations. Features with higher importance scores are more influential in predicting the target variable, while those with lower scores contribute less to the final decision. Understanding feature importance helps in improving model interpretability and identifying the most relevant features for prediction.

In the code, the feature importances are extracted from the trained AdaBoost model using the .feature\_importances\_ attribute:

importances = best\_adaboost.feature\_importances\_

This returns an array containing the importance scores of each feature in the dataset. These scores are then combined with the feature names into a DataFrame for better clarity:

The resulting feature\_importance\_df DataFrame allows us to see both the feature names and their corresponding importance scores. This analysis not only helps in identifying the most impactful features but also provides guidance on feature selection and model refinement, ultimately contributing to better model performance and a more efficient prediction process.

## Visualizing Feature Importance for Fine-Tuned AdaBoost Model:

After calculating the feature importances, we proceed to visualize them in a **horizontal bar chart**. This visualization allows us to clearly observe which features have the greatest impact on the predictions of the AdaBoost model, particularly after hyperparameter tuning. It helps in determining the relative significance of each feature, guiding potential decisions for feature selection or further refinement of the model.

The feature importance DataFrame is first sorted in descending order to prioritize the most influential features:

feature\_importance\_df = feature\_importance\_df.sort\_values(by='Importance', ascending=False)

Then, the feature importances are plotted using a horizontal bar chart:

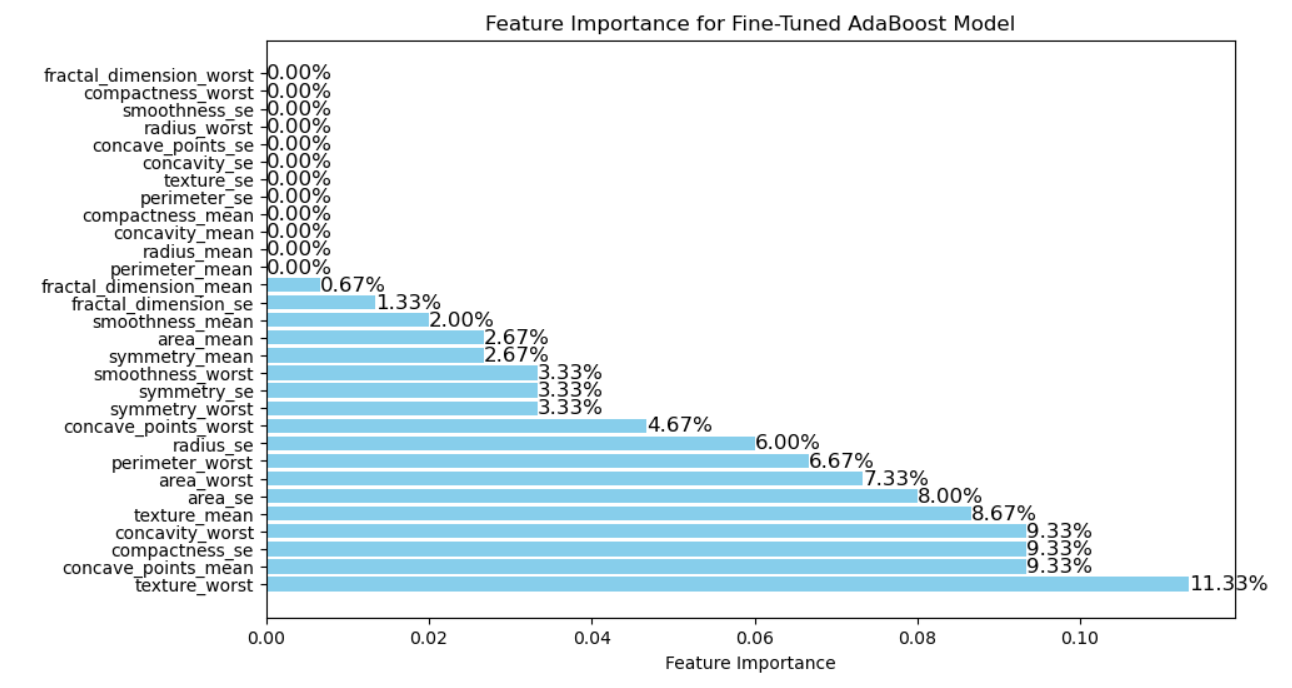
plt.barh(feature\_importance\_df['Feature'], feature\_importance\_df['Importance'], color='skyblue')

Each bar represents a feature, and its length indicates the importance score. The x-axis shows the importance values, while the y-axis lists the feature names.

To enhance readability, the importance values are annotated on the bars as percentages:

plt.text(value, index, f'{value\*100:.2f}%', va='center', color='black', fontsize=12)

This step allows us to directly see the percentage importance of each feature, making the plot more informative. Finally, the plot is displayed, providing a clear and concise visual of feature importance in the fine-tuned AdaBoost model. This helps in understanding how each feature contributes to the model’s performance and provides insights into the most relevant features for breast cancer prediction.



## Selecting Important Features for Fine-Tuned AdaBoost Model

To focus on the most relevant features for building the AdaBoost model, we filter out features with an importance score below a specified threshold. In this case, only features with an importance greater than **3%** (or 0.03 in decimal) are selected for further consideration. This helps eliminate less impactful features, streamlining the model and potentially improving its performance.

The process starts by filtering the features based on their importance:

selected\_features = feature\_importance\_df[feature\_importance\_df['Importance'] >= 0.03]

Then, the selected features are sorted in descending order based on their importance scores to prioritize the most significant ones:

A bar chart is generated to visualize these selected features:

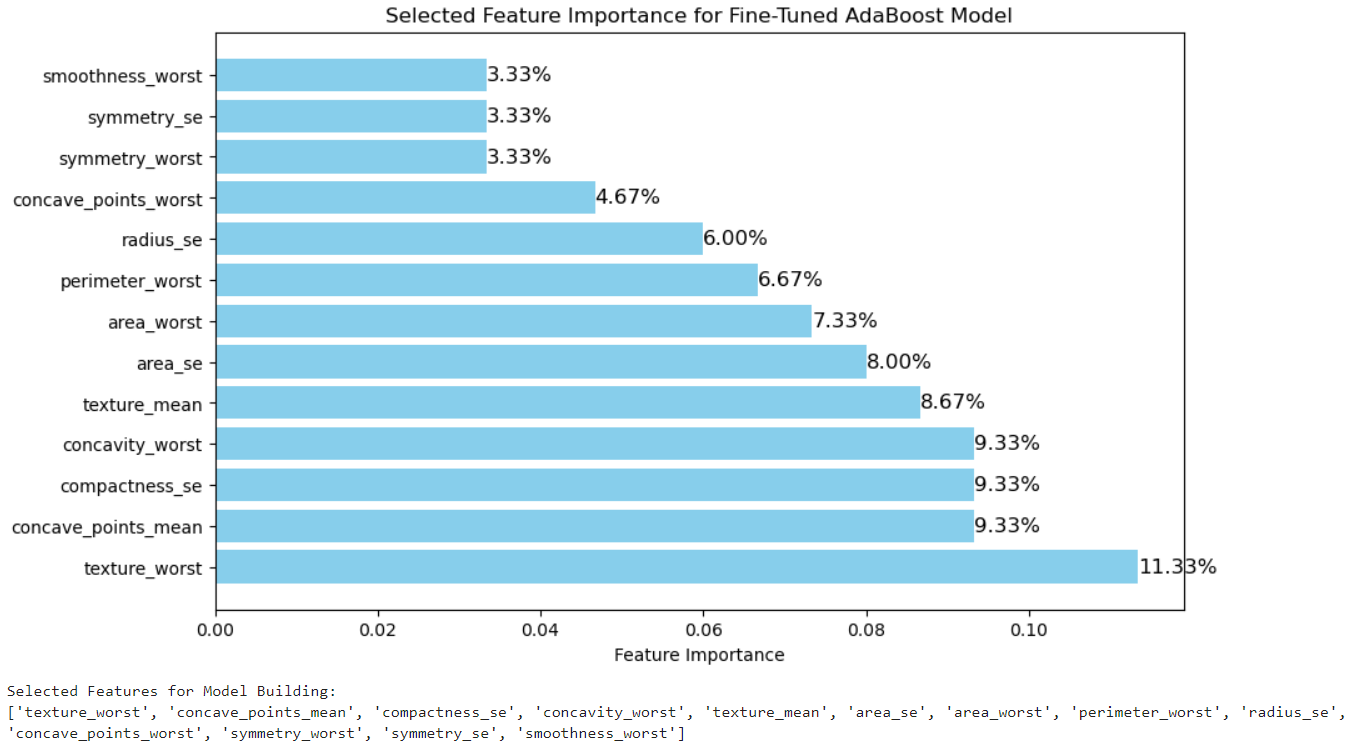
plt.barh(selected\_features['Feature'], selected\_features['Importance'], color='skyblue')

The chart displays the features on the y-axis and their corresponding importance scores on the x-axis. To make the plot more informative, each bar is annotated with the feature's importance as a percentage:

plt.text(value, index, f'{value\*100:.2f}%', va='center', color='black', fontsize=12)

This visualization provides a clear picture of the key features driving the AdaBoost model's predictions. Afterward, the selected features are printed to further confirm which variables were retained for model building:

This step allows for a more focused and efficient model development process by emphasizing the most influential features.



# Re-training AdaBoost with Selected Features:

After selecting the most important features based on their calculated importance scores, the AdaBoost model is retrained using only these selected features. This step is crucial for improving the model's efficiency and potentially enhancing its performance. By focusing on the most relevant features, we reduce the complexity of the model and prevent overfitting, which can occur when too many irrelevant features are included.

Re-training the AdaBoost model with only the selected features allows the model to learn from the data that contributes the most to predicting the target variable (in this case, breast cancer detection). This streamlined model is likely to generalize better, as it focuses on the key predictors rather than being distracted by less important variables.

The process of retraining with selected features typically leads to:

1. **Faster Training**: With fewer features, the model trains more quickly.
2. **Reduced Overfitting**: By eliminating less important features, the model avoids overfitting to noise or irrelevant data.
3. **Improved Generalization**: The model is more likely to perform better on unseen data, as it is trained on the most significant features.

This step is important because it enhances the model's interpretability, reduces computational costs, and helps improve the model’s ability to generalize, all of which contribute to a more robust and accurate classifier.

## Feature Scaling for Selected Features

After selecting the relevant features for model training, the next step is to **scale** the features using **StandardScaler**. Scaling is crucial because it standardizes the data, ensuring that all features have the same scale. Many machine learning algorithms, including AdaBoost, perform better when the features are on a similar scale. Without scaling, features with larger numerical ranges could dominate the learning process, leading to suboptimal model performance.

In the code, the selected features (x\_select) are scaled using the StandardScaler:

scaler = StandardScaler()

x\_selected\_scaled = scaler.fit\_transform(x\_select)

The StandardScaler transforms each feature by subtracting the mean and dividing by the standard deviation, ensuring that the resulting features have a mean of 0 and a standard deviation of 1. This transformation ensures that the model treats all features equally, preventing any one feature from disproportionately influencing the model's decisions.

### Importance of Feature Scaling:

Feature scaling is particularly important in algorithms like AdaBoost that use weighted voting of weak learners, as it ensures that each feature is treated equally during the training process. By standardizing the features, the model can converge more efficiently and achieve better accuracy, making the learning process faster and more stable.

## Train-Test Split for Selected Features

Once the selected features have been scaled, the dataset is split into training and testing sets. This is a crucial step in evaluating the model’s performance, as it ensures that the model is tested on data it has not seen during training, allowing for a fair assessment of its generalization ability.

In this case, the training set will contain 70% of the data, and the testing set will contain the remaining 30%. The split is performed using the train\_test\_split function from sklearn.model\_selection:

X\_train\_sel, X\_test\_sel, y\_train\_sel, y\_test\_sel = train\_test\_split(x\_select, y\_select, test\_size=0.3, random\_state=42)

Here:

* X\_train\_sel and X\_test\_sel represent the training and testing feature sets, respectively.
* y\_train\_sel and y\_test\_sel are the corresponding labels (target variable, Diagnosis) for the training and testing sets.
* test\_size=0.3 specifies that 30% of the data will be used for testing, and the remaining 70% will be used for training.
* random\_state=42 ensures that the split is reproducible, meaning that every time the code runs, the same split will be made.

### Importance of Train-Test Split:

This step is essential for:

1. **Preventing Overfitting**: By training the model on one subset of the data and testing it on another, we ensure that the model does not memorize the training data and can generalize to new, unseen data.
2. **Evaluating Model Performance**: The test set provides an unbiased evaluation of the model’s accuracy and performance, allowing us to gauge its effectiveness in making predictions.
3. **Reproducibility**: By using a fixed random seed (random\_state), we ensure that the results are consistent across different runs of the model, aiding in reproducibility of the analysis.

### Heading: Class Imbalance Handling with SMOTE for Selected Features

To address class imbalance in the training dataset, **SMOTE (Synthetic Minority Over-sampling Technique)** is applied. SMOTE generates synthetic samples of the minority class to balance the number of examples between the classes. This helps prevent the model from being biased toward the majority class and ensures that the model learns from both classes equally.

The SMOTE technique is applied to the selected and scaled training data (X\_train\_sel, y\_train\_sel) to create a balanced dataset:

smote = SMOTE(random\_state=42)

X\_train\_smote\_sel, y\_train\_smote\_sel = smote.fit\_resample(X\_train\_sel, y\_train\_sel)

* random\_state=42 ensures reproducibility of the synthetic sample generation.
* X\_train\_smote\_sel and y\_train\_smote\_sel are the resampled training features and labels, respectively, where the minority class has been artificially increased to match the majority class.

SMOTE is critical in imbalanced datasets because it:

1. **Balances the Classes**: By generating synthetic samples for the minority class, SMOTE ensures that the model has a balanced representation of both classes during training, which helps improve the model's ability to detect the minority class.
2. **Improves Model Performance**: Class imbalance can lead to models that predict the majority class most of the time. SMOTE helps mitigate this by providing the model with a more even distribution of classes.
3. **Reduces Bias**: Without techniques like SMOTE, models trained on imbalanced datasets often fail to recognize the minority class, leading to biased predictions. SMOTE ensures the model is better equipped to classify both classes correctly.

## Training and Prediction with AdaBoost Model

After addressing class imbalance with **SMOTE**, the **AdaBoost classifier** is trained on the resampled training data (X\_train\_smote\_sel, y\_train\_smote\_sel). The model is then used to make predictions on the test data (X\_test\_sel).

The training process is carried out with the following code:

ada\_boost.fit(X\_train\_smote\_sel, y\_train\_smote\_sel)

Here, the AdaBoost model is trained using the balanced dataset, where the features (X\_train\_smote\_sel) are paired with the corresponding labels (y\_train\_smote\_sel). The SMOTE technique has ensured that the training set now has a more balanced distribution of both the malignant and benign classes.

Once the model is trained, it is used to predict the labels of the test set:

y\_pred = ada\_boost.predict(X\_test\_sel)

This step generates predictions for the test set (X\_test\_sel), which can then be evaluated against the true labels (y\_test\_sel) to assess the model's performance.

### Importance of Training and Prediction:

1. **Model Training**: By training the AdaBoost model on the balanced dataset, the classifier is exposed to an equal distribution of both classes, improving its ability to detect the minority class.
2. **Prediction**: Making predictions on the test set allows us to assess the performance of the trained model on new, unseen data. This helps us determine how well the model generalizes and ensures that it is not overfitting to the training data.
3. **Class Imbalance Mitigation**: The application of SMOTE ensures that the model is trained on a balanced dataset, addressing the potential issue of class imbalance, which could otherwise lead to biased predictions.

# Model Evaluation - Classification Report for Selected Features (Without Hyperparameter Tuning)

After training the AdaBoost classifier on the resampled dataset using the selected features, the model's performance is evaluated using a **classification report**. The classification report provides key metrics, including precision, recall, F1-score, and accuracy, for each class (benign and malignant). These metrics are crucial for understanding the model's ability to distinguish between the two classes, especially when dealing with an imbalanced dataset.

The classification report is generated with the following code:

print("Classification Report for selected features (without hyperparameter tuning):\n")

print(classification\_report(y\_test, y\_pred))

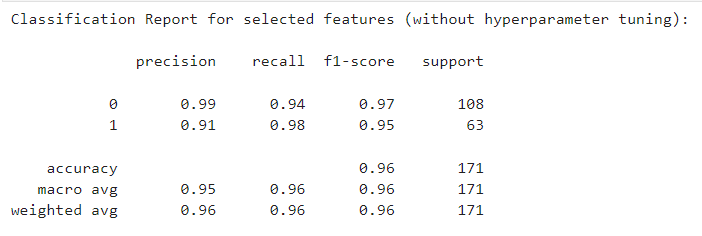
Here, y\_test represents the true labels for the test set, and y\_pred contains the predicted labels generated by the AdaBoost model. The classification report will summarize the performance of the model by computing:

* **Precision**: The proportion of true positive predictions out of all positive predictions.
* **Recall**: The proportion of true positives out of all actual positives in the data.
* **F1-score**: The harmonic mean of precision and recall, providing a balance between the two.
* **Accuracy**: The overall proportion of correct predictions.

### Importance of Classification Report:

The classification report is essential for:

1. **Evaluating Model Performance**: It provides a comprehensive view of how well the model is performing across different metrics, particularly in distinguishing between malignant and benign cases.
2. **Identifying Strengths and Weaknesses**: By looking at precision, recall, and F1-score, we can assess whether the model is biased towards one class or if it is successfully detecting both classes.
3. **Guiding Model Improvement**: The classification report helps identify areas where the model may need improvement, such as adjusting thresholds or applying further tuning techniques like hyperparameter optimization.



# Hyperparameter Tuning with GridSearchCV for Selected Features

To further optimize the AdaBoost model's performance, hyperparameter tuning is performed using **GridSearchCV**. This technique searches through a predefined set of hyperparameters to find the combination that results in the best model performance. The focus is on improving recall, which is crucial in imbalanced datasets, where detecting the minority class (malignant cases) is more important.

In this step, the **GridSearchCV** is applied to the AdaBoost model, using the previously defined hyperparameter grid (param\_grid), which includes options for n\_estimators (number of weak learners) and learning\_rate (rate of model adjustment). The process is carried out with the following code:

grid\_search\_Sel = GridSearchCV(estimator=ada\_boost, param\_grid=param\_grid, cv=5, scoring='recall', n\_jobs=-1)

grid\_search\_Sel.fit(X\_train\_smote\_sel, y\_train\_smote\_sel)

Here:

* param\_grid defines the hyperparameters to search: different values for n\_estimators and learning\_rate.
* cv=5 specifies 5-fold cross-validation, which ensures the model's performance is evaluated across multiple splits of the training data.
* scoring='recall' ensures that the grid search optimizes for recall, as it is the most relevant metric when dealing with imbalanced classes.
* n\_jobs=-1 ensures the grid search uses all available CPU cores for faster execution.

### Importance of Hyperparameter Tuning:

Hyperparameter tuning is critical because:

1. **Improves Model Performance**: By systematically searching through different hyperparameter combinations, GridSearchCV ensures that the AdaBoost model is trained with the best possible configuration, leading to improved accuracy and recall.
2. **Optimizes Recall**: Since recall is prioritized, the grid search specifically aims to improve the model’s ability to correctly identify the minority class (malignant cases), which is crucial in breast cancer detection.
3. **Boosts Generalization**: Proper hyperparameter tuning helps the model generalize better to new, unseen data, reducing the risk of overfitting and ensuring better real-world performance.

# **RESULTS:**

# Confusion Matrix and Classification Report for Best AdaBoost Model

After hyperparameter tuning using **GridSearchCV**, the best AdaBoost model is evaluated on the test set. This evaluation involves both a **classification report** and a **confusion matrix**, providing a comprehensive assessment of the model’s performance.

First, the best model is retrieved from the GridSearchCV results and evaluated on the test data:

best\_adaboost\_sel = grid\_search\_Sel.best\_estimator\_

The **classification report** is then printed, which includes key metrics such as precision, recall, F1-score, and accuracy for both classes:

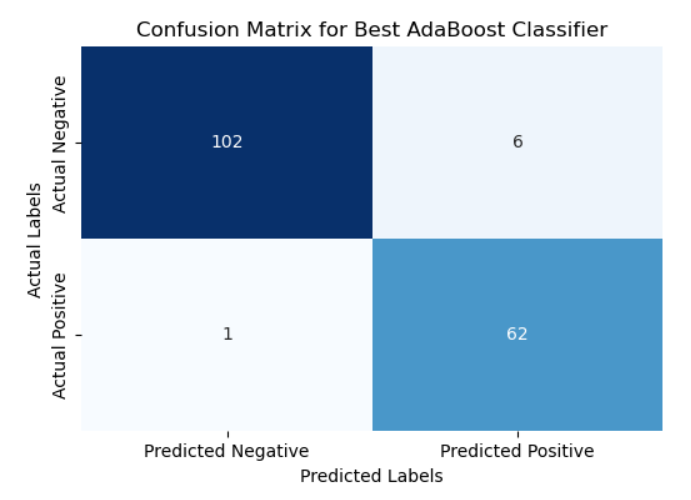
This report helps assess the overall performance of the model, with particular focus on recall due to the class imbalance.

Finally, the **confusion matrix** is plotted to visually represent the true positives, false positives, true negatives, and false negatives:

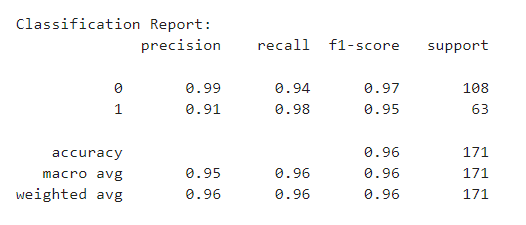
This matrix helps to better understand how well the model is performing in terms of correctly predicting both classes. The heatmap highlights the number of predictions made for each combination of actual and predicted labels, providing a clear visual of the model's strengths and weaknesses.

### Importance of the Confusion Matrix and Classification Report:

**Confusion Matrix**: This matrix provides a detailed view of the model’s performance, showing how well it distinguishes between true positive and negative cases, as well as false positives and false negatives. This is crucial in tasks like cancer detection, where false negatives (failing to identify malignant cases) are particularly problematic.



**Classification Report**: The classification report summarizes the model’s precision, recall, F1-score, and support. It allows for an easy assessment of how well the model performs in terms of both identifying true positives and minimizing false positives, which is essential when working with imbalanced datasets.



# Model Evaluation - ROC Curve for Best AdaBoost Model

To assess the performance of the best AdaBoost model after hyperparameter tuning, the **Receiver Operating Characteristic (ROC) curve** is plotted. The ROC curve is a graphical representation of a model's ability to distinguish between the positive and negative classes across various thresholds. It provides valuable insight into the tradeoff between **True Positive Rate (TPR)** and **False Positive Rate (FPR)**, which is crucial for evaluating classifier performance, especially in imbalanced datasets.

The **predict\_proba** method is used to obtain the predicted probabilities for the test set:

y\_pred\_proba\_sel = best\_adaboost\_sel.predict\_proba(X\_test\_sel)

These probabilities are used to calculate the ROC curve. For binary classification, we focus on the probabilities of the positive class (malignant):

fpr, tpr, \_ = roc\_curve(y\_test\_sel, y\_pred\_proba\_sel[:, 1])

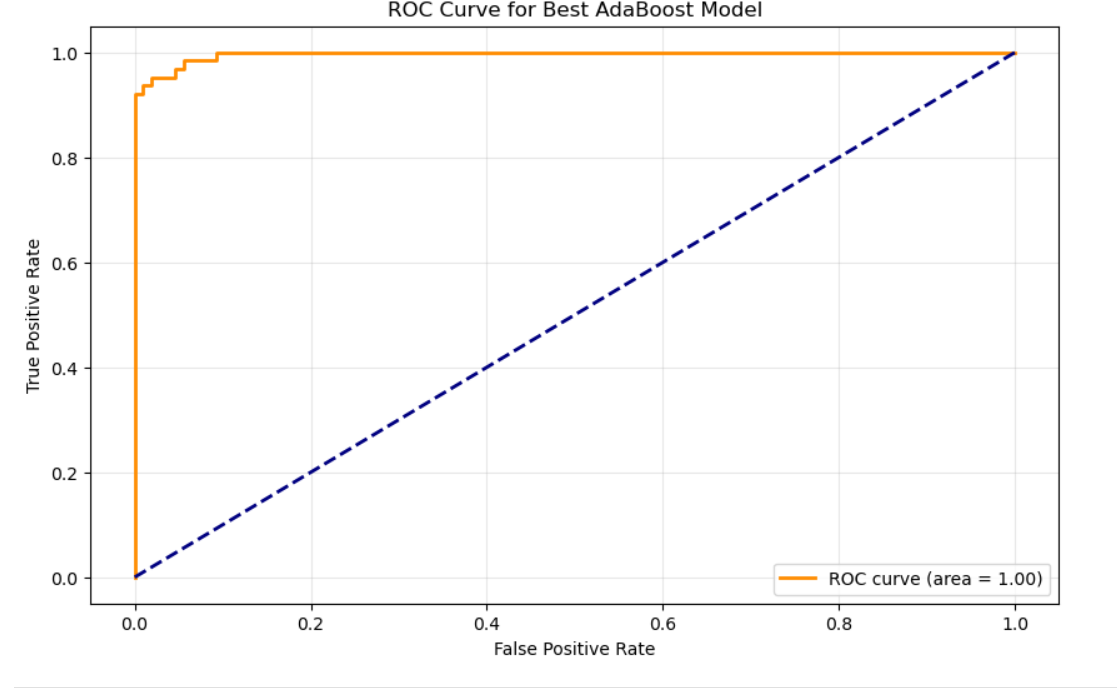
This plot shows the performance of the AdaBoost model at various classification thresholds. The **diagonal dashed line** represents a random classifier, and the area under the curve (AUC) quantifies the model’s performance—values closer to 1 indicate better discrimination.

### Importance of the ROC Curve:

**Evaluating Model Performance**: The ROC curve provides a clear visual of how well the model distinguishes between the classes. A curve that is closer to the top-left corner indicates better performance.

**AUC (Area Under the Curve)**: The AUC gives an aggregate measure of the model’s ability to discriminate between the positive and negative classes. A higher AUC value suggests better performance.

**Threshold Evaluation**: The ROC curve helps identify the optimal threshold for classification, balancing the tradeoff between true positives and false positives, which is particularly important in healthcare applications where false negatives can have serious consequences.



## Saving the Trained Model and Metrics

After training the AdaBoost model and evaluating its performance, the next step is to save the trained model, the scaler, and key evaluation metrics for future use. This process is essential for deploying the model or using it in subsequent applications without needing to retrain it.

A dictionary (model\_data) is created to store the following:

* The **scaler** used to preprocess the data.
* The **AdaBoost model** itself (best\_adaboost\_sel).
* A set of **metrics** including accuracy, precision, and F1-score.

The dictionary is then saved to a file using **pickle**, which serializes the model and its associated objects:

The file is saved as best\_model.sav, and the message confirms that the model, scaler, and metrics have been successfully stored.

### Importance of Saving the Model:

1. **Reusability**: Saving the trained model allows for its reuse in different applications or production environments without the need to retrain it, saving time and computational resources.
2. **Scalability**: The scaler and model can be loaded in new environments to make predictions on new data, ensuring consistent preprocessing and model behavior.
3. **Evaluation and Monitoring**: Saving the metrics alongside the model allows for future comparisons and monitoring of the model’s performance over time, enabling easier updates or improvements when needed.

# Breast Cancer Detection Web App Using AdaBoost Classifier

This code implements a **Streamlit** web application for breast cancer detection using a **trained AdaBoost classifier**. The app provides a user-friendly interface to input features, visualize dataset information, and display prediction results based on user inputs.

## Key Features:

**Model Loading**: The model, scaler, and evaluation metrics are loaded from a saved file (best\_model.sav), ensuring that the trained model and its performance metrics are preserved for future use.

**User Input**: The sidebar allows the user to input feature values for a patient. It ensures that the inputs are numerical and provides a warning if the input is invalid.

**Dataset Information & Visualization**: The sidebar displays information about the dataset, including the number of records and features, and a pie chart visualizes the composition of the dataset (Benign vs Malignant).

**Prediction and Report**: After the user inputs the feature values and clicks the 'Predict' button, the model makes a prediction, and the result is displayed in a styled report format. The report includes metrics like **accuracy**, **precision**, and **F1-score**, which were saved during model training.

**Styling and Layout**: The app is styled with a clean, medical-themed design and a gradient background, ensuring an intuitive and appealing user interface.

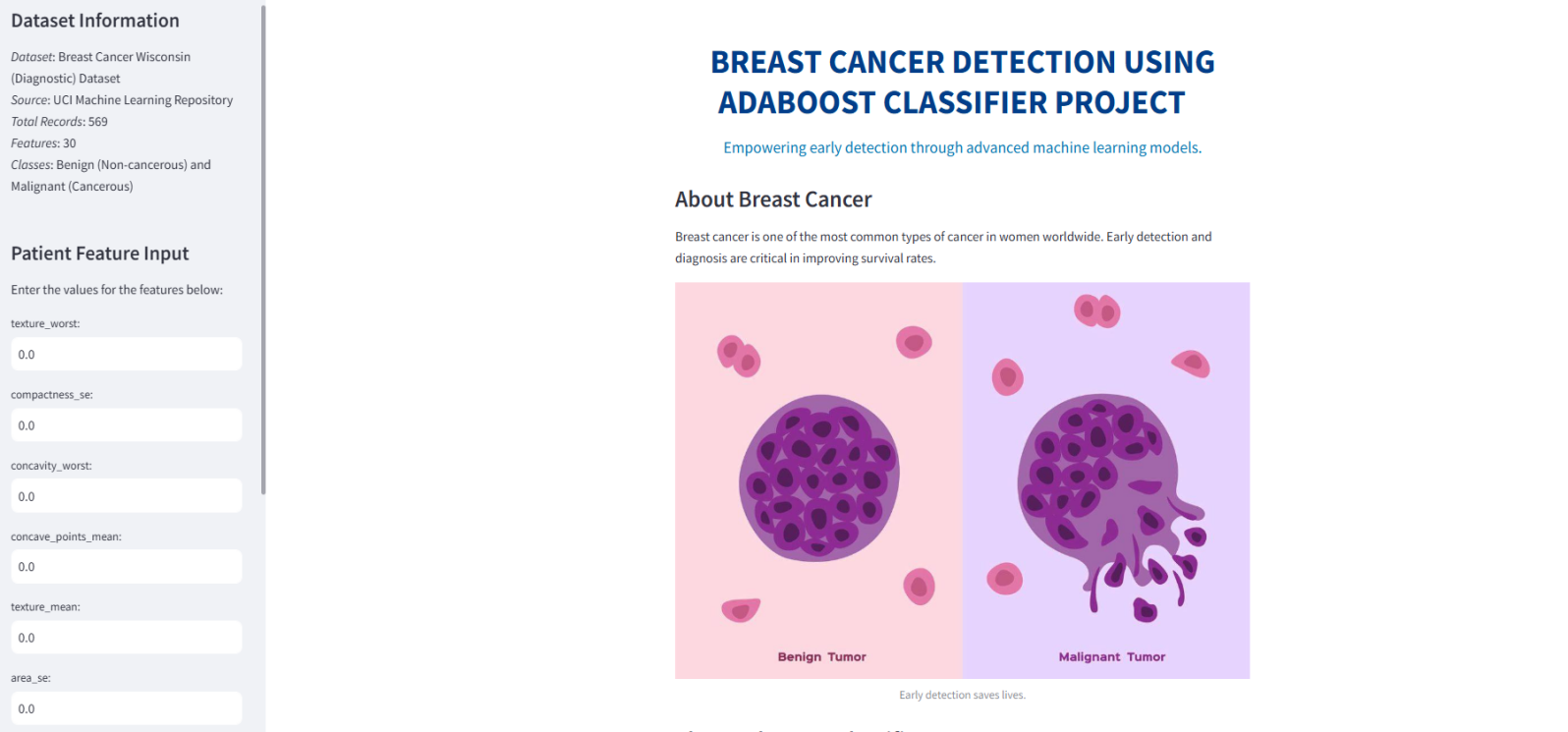
## **Im**portance of this Application:

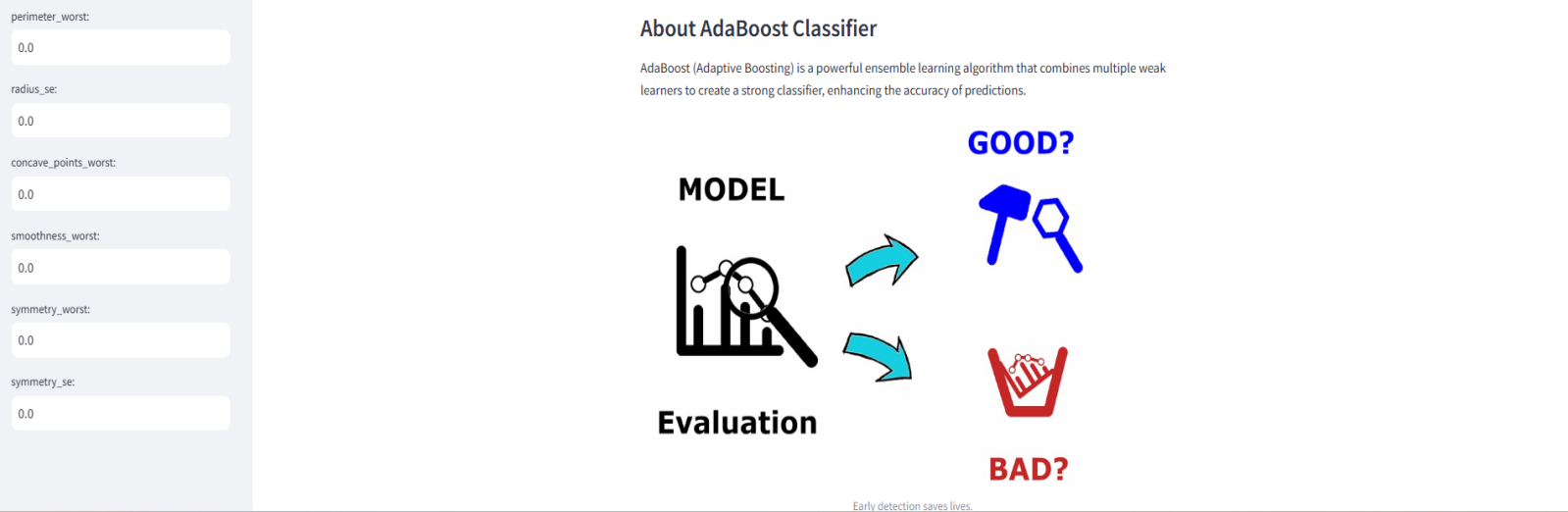
* **User-Friendly Interface**: The app provides a straightforward way for users to input data and receive predictions, making it accessible to both medical professionals and researchers.
* **Model Reusability**: By saving the trained model and associated metrics, the application ensures that the model can be easily deployed and reused without needing retraining.
* **Early Detection**: The model uses features from the Breast Cancer dataset to predict whether a patient has malignant or benign cancer, providing critical information for early diagnosis and treatment planning.

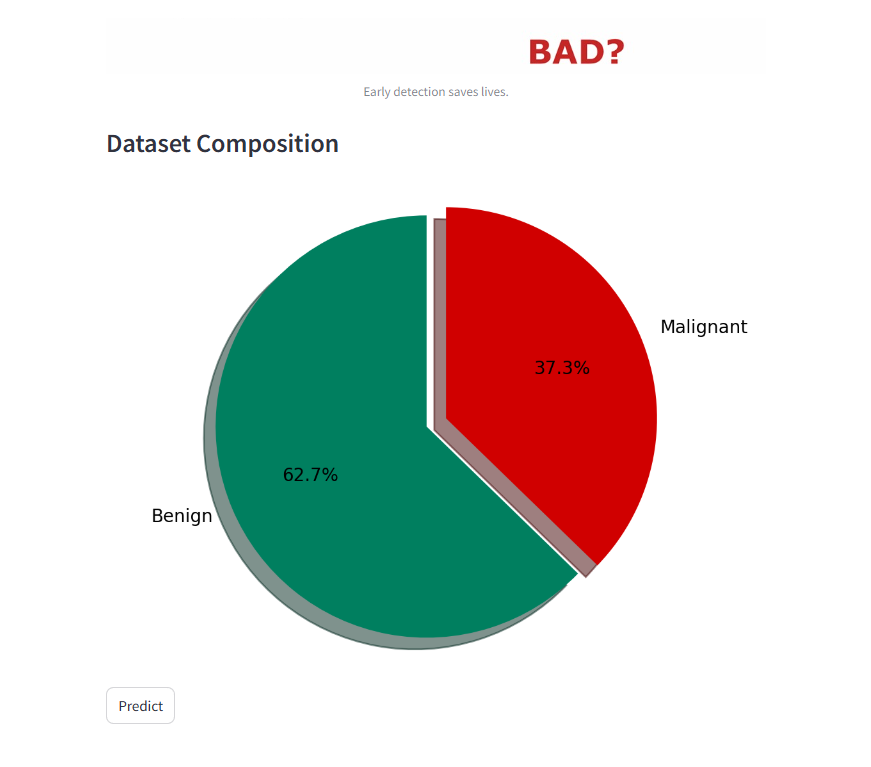
This app is an example of how machine learning can assist in medical diagnostics, helping to make faster and more accurate decisions.

# Streamlit Application Interface:

The following images showcase the user interface of our **Breast Cancer Detection Streamlit Application**. The app provides an intuitive and interactive platform for users to input patient data, view dataset information, and receive predictions from the AdaBoost model. The interface includes a sidebar for entering feature values, a section displaying the dataset composition, and a results area that presents the prediction along with performance metrics such as accuracy, precision, and F1-score. These visualizations and functionalities enable users to easily interact with the model, making it accessible for real-time breast cancer detection and analysis.









## Advantages of the Breast Cancer Detection Streamlit Application

1. **Early Detection**: Predicts malignant or benign tumors, aiding early-stage diagnosis.
2. **Ease of Use**: Simple interface for quick interaction, even for non-technical users.
3. **Real-time Predictions**: Provides instant results to support quick clinical decisions.
4. **Data-Driven Decisions**: Reduces human error by leveraging machine learning for accurate predictions.
5. **Customizable and Scalable**: Can be tailored for different healthcare environments and scaled for broader use.
6. **Comprehensive Reporting**: Displays key metrics like accuracy and precision to evaluate model performance.

## Applications of the Breast Cancer Detection Streamlit Application

1. **Clinical Use**: Provides quick tumor analysis for faster clinical decision-making.
2. **Supplementary Diagnosis**: Acts as a supportive tool for clinicians by offering data-backed predictions.
3. **Telemedicine**: Facilitates remote breast cancer detection, especially in rural or underserved areas.
4. **Training and Education**: A useful platform for medical professionals to learn about machine learning in diagnostics.

# Conclusion

The Breast Cancer Detection Streamlit Application represents a significant advancement in the early diagnosis and detection of breast cancer. By utilizing the AdaBoost classifier—a powerful machine learning algorithm that combines multiple weak learners to form a strong predictor—the application provides real-time, data-driven predictions for distinguishing between malignant and benign tumors. This capability is crucial, as early detection of breast cancer greatly enhances the chances of successful treatment and improves survival rates.

The application’s user-friendly interface ensures that it is accessible to a wide range of users, including medical professionals who may not have a background in machine learning or data science. The intuitive design allows users to easily input patient data and receive immediate predictions, making it a valuable tool in clinical settings. The inclusion of performance metrics such as accuracy, precision, and F1-score further ensures that users can trust the predictions made by the model, providing a solid foundation for clinical decision-making.

Moreover, the Streamlit app’s ability to provide real-time predictions based on feature inputs directly addresses the challenges of speed and efficiency in medical diagnostics. The app is also highly customizable, which means it can be tailored to meet the specific needs of different healthcare environments, whether in urban hospitals, remote clinics, or telemedicine settings. By automating part of the diagnostic process, it reduces reliance on expensive diagnostic tools and extensive manual labor, making it a more cost-effective solution for healthcare providers.

Additionally, the app’s ability to visualize key metrics and dataset composition through pie charts and confusion matrices helps clinicians and medical professionals interpret the model’s outputs more easily, allowing them to make well-informed decisions. This feature can be particularly helpful in patient consultations, where clear visual aids enhance the communication of the diagnosis.

Looking forward, there are numerous opportunities for further development and refinement of the application. Future enhancements could include incorporating more diverse datasets to improve the model's robustness across different populations, integrating more advanced machine learning techniques to refine accuracy, or adding more predictive features that could offer deeper insights into breast cancer risks. Additionally, increasing the model’s interpretability, perhaps by providing feature importance scores in more detail, could help clinicians better understand the decision-making process behind the model's predictions.

Another potential area for improvement is the integration of this application with other diagnostic tools used in breast cancer screening, such as mammograms or ultrasounds, to create a comprehensive diagnostic system that combines the strengths of both machine learning and traditional imaging methods. Furthermore, expanding the application to include continuous learning, where the model could be updated with new data over time, would ensure that the app remains relevant as medical knowledge and datasets evolve.

In conclusion, the Breast Cancer Detection Streamlit Application not only showcases the power of machine learning in healthcare but also provides a practical tool for improving breast cancer detection, offering clinicians a valuable resource for decision support. By continuing to refine and expand its capabilities, this application has the potential to become an indispensable part of the breast cancer diagnostic process, empowering healthcare professionals to make faster, more accurate decisions and ultimately improving patient outcomes.