CA-Classer: An efficient Cancer classifier using different ML algorithms

Course Title: Artificial Intelligence

Submitted by

Naima Tahsin Nodi
20101150

Samiur Rahman
20101147

Md. Hasanur Rashid
20101275

Tasnia Zaman
20301296

Contents

1 Introduction

1.1 Problem Statement

Accurate and timely disease diagnosis has risen to the forefront of today's healthcare system. To this end, we introduce "CA-Classer," an ambitious new initiative that seeks to transform cancer classification by using sophisticated machine learning techniques. In order to build a reliable and effective cancer classifier, our project sets out on a quest to combine the strengths of Decision Trees, Support Vector Machines (SVM), K-Nearest Neighbours (KNN), and XGBOOST models.

1.2 Project Objective

The primary goal of CA-Classer is to create a robust and effective cancer classifier that can properly distinguish between various cancer types using patient data. In the current medical system, proper cancer classification is crucial for developing effective treatment plans and predicting patient outcomes. However, current diagnostic methods are generally time-consuming and error-prone due to their reliance on human labor.

This research seeks to solve this major issue head-on by developing an automated approach to improve the accuracy and speed with which cancers are classified. Our goal is to use machine learning algorithms to create a tool that can help with both the diagnostic process and early detection, leading to better results for patients.

1.3 Motivation Behind the Project

Our hope is that CA-Classer will serve as a link between medical technology and those in need of it. Machine learning has the potential to speed up cancer classification, which could facilitate more efficient treatment programs and individual care paths while also relieving pressure on healthcare providers. The potential for such a device to improve patients' quality of life is what drives our team to explore the limits of present cancer diagnostic methods.

When considering the trajectory of medical innovation, the CA-Classer stands out as a ray of light, illuminating the way toward rapid, precise, and automated cancer classification. We hope to make substantial contributions to oncology by combining the strengths of several machine learning algorithms in order to improve cancer diagnosis and treatment.

2 Dataset Description

2.1 Source:

- **Link:** https://www.kaggle.com/datasets/erdemtaha/cancer-data
- **Reference:** *Cancer data*. (2023, March 22). Kaggle. https://www.kaggle.com/datasets/erdemtaha/cancer-data

2.2 Description

Our dataset classifies 569 cells as either benign (B) or malignant (M) tumors based on 33 features. With this information, we can get closer to our goal of developing a reliable cancer classifier, a significant step forward in the quest for more precise and time-saving diagnostics.

This is a classification problem, as we are aiming to classify the cells as either Benign or Malignant.

2.2.1 How many features?

There are **33** features in the dataset, which are ‘id’, 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean', 'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean', 'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se', 'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se', 'fractal_dimension_se', 'radius_worst', 'texture_worst', 'perimeter_worst', 'area_worst', 'smoothness_worst', 'compactness_worst', 'concavity_worst', 'concave points_worst', 'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32’.

2.2.2 Classification or regression problem

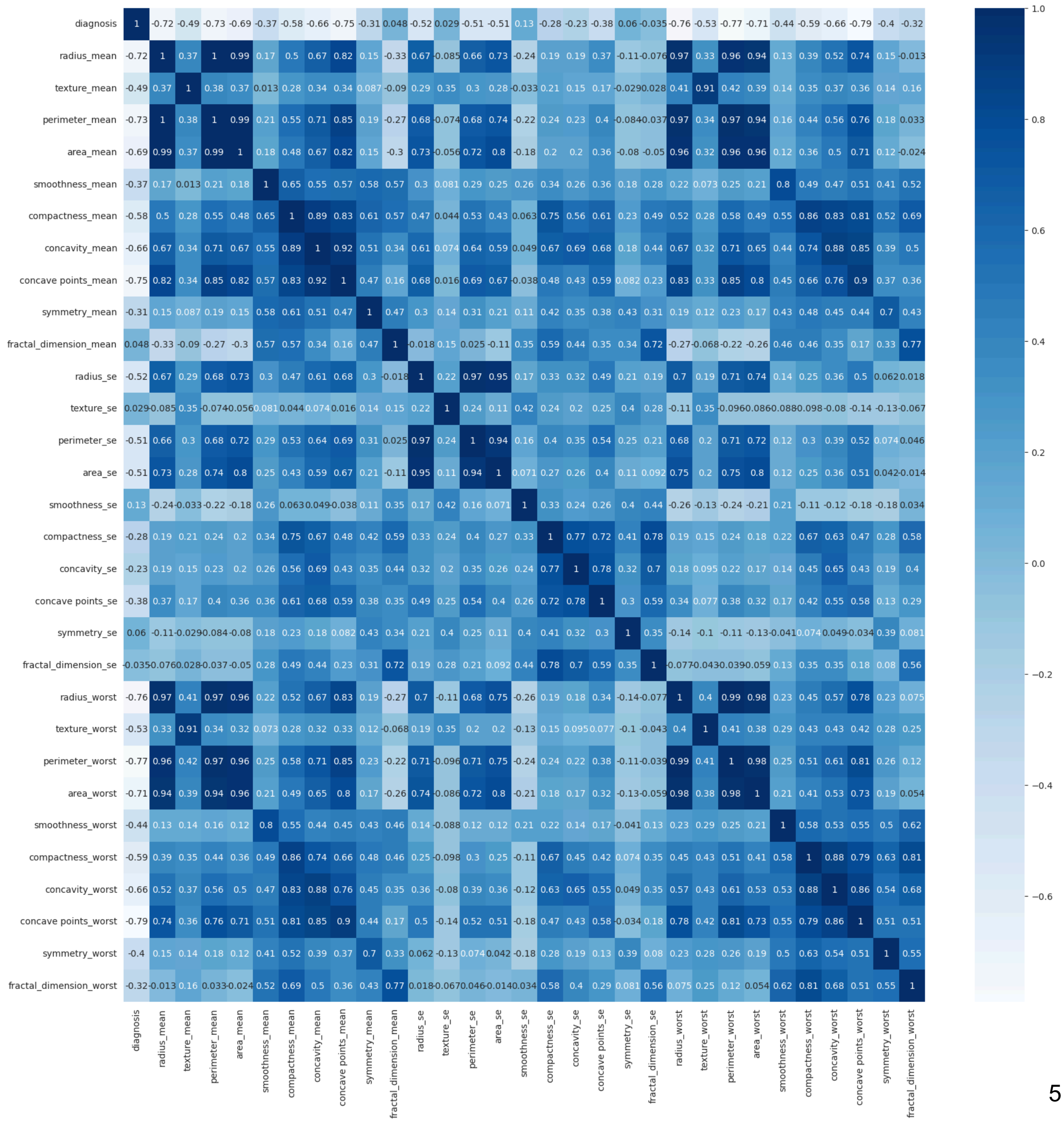
This is a classification problem. The objective is to determine which class or category a patient belongs to based on their medical history and the outcomes of diagnostic testing. We can safely say that cancer identification is a classification challenge, which typically involves binary classifications (the cells are Benign or Malignant). In order to discover patterns and correlations between the qualities and the target variable, machine learning models are trained on a collection of labeled data. After training, the algorithm may be used to forecast. results based on fresh, uncovered data, categorizing people as having cancer cells that are either Benign or Malignant.

2.2.3 How many data points

There are 569 data points

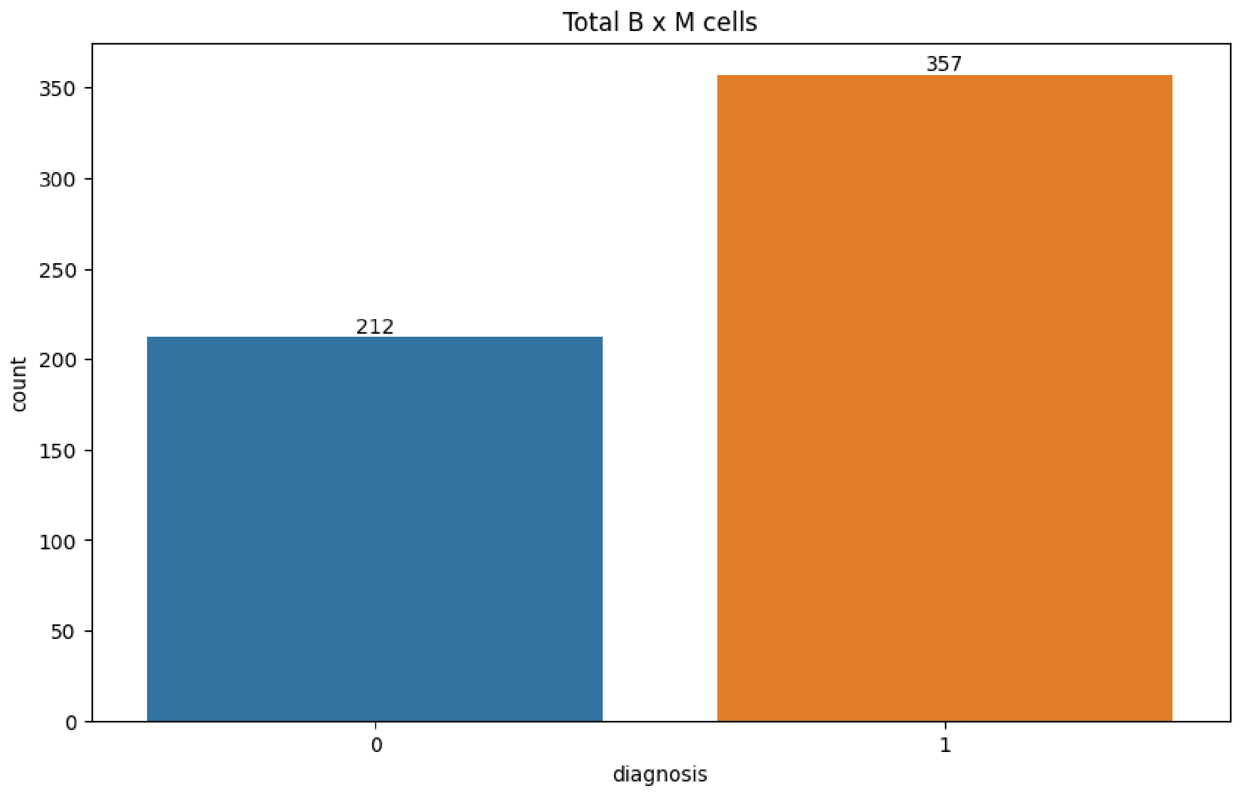
2.2.4 What kind of features are in your dataset (Quantitative / Categorical)

In our dataset, the features are both quantitative and categorical .
Applying heatmap using the seaborn library, the correlation of all the features (input and output features) is provided below.



2.2.5 Imbalanced Dataset

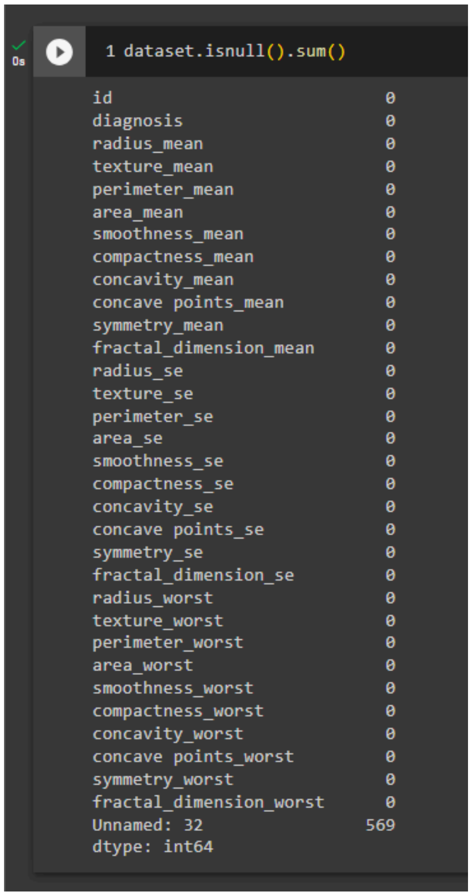
For the output feature, all unique classes do not have an equal number of instances.



3 Dataset Pre-processing

In the pre-processing, we have removed the null values, duplicate rows, and unnecessary data and also encoded the categorical values.

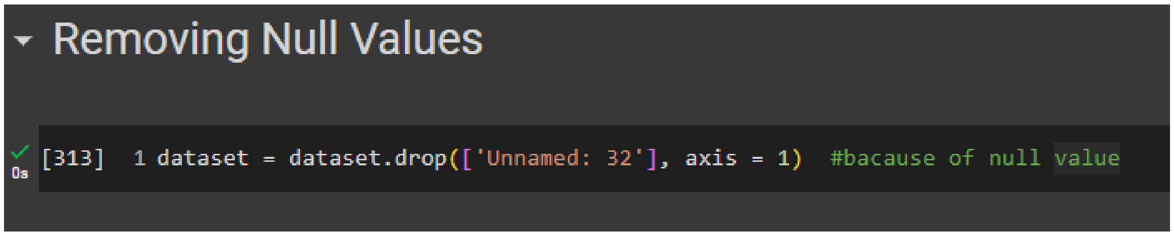
For checking the null values in our dataset, we applied isnull().sum() function:



```
dataset.isnull().sum()
```

```
id                          0
diagnosis                   0
radius_mean                 0
texture_mean                0
perimeter_mean              0
area_mean                   0
smoothness_mean             0
compactness_mean            0
concavity_mean              0
concave points_mean         0
symmetry_mean               0
fractal_dimension_mean      0
radius_se                   0
texture_se                  0
perimeter_se                0
area_se                     0
smoothness_se               0
compactness_se              0
concavity_se                0
concave points_se           0
symmetry_se                 0
fractal_dimension_se        0
radius_worst                0
texture_worst               0
perimeter_worst             0
area_worst                  0
smoothness_worst            0
compactness_worst           0
concavity_worst             0
concave points_worst        0
symmetry_worst              0
fractal_dimension_worst     0
Unnamed: 32               569
dtype: int64
```

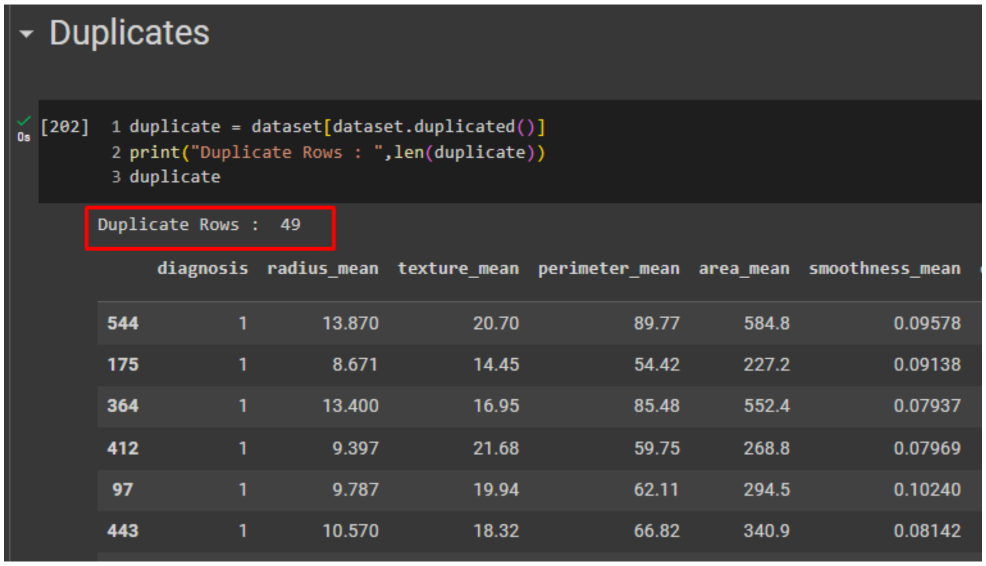
Here, it is clear that the Unnamed: 32 Column has all the null values. So we dropped it accordingly:



Removing Null Values

```
[313]  dataset = dataset.drop(['Unnamed: 32'], axis = 1)  #bacause of null value
```

After that, we checked for duplicate rows in our dataset and found 49 duplicate rows:



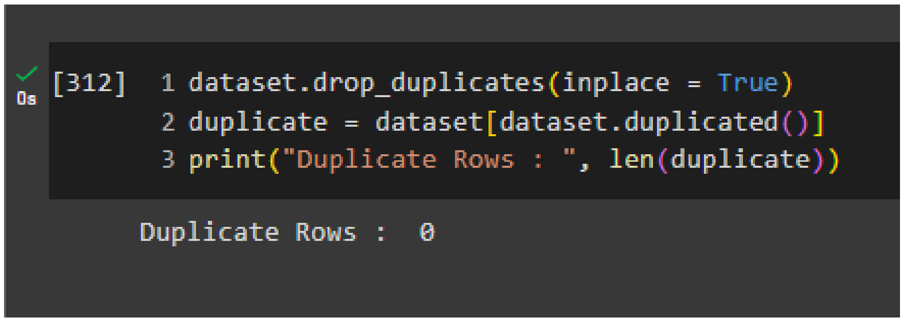
Duplicates

```
[202]
duplicate = dataset[dataset.duplicated()]
print("Duplicate Rows : ",len(duplicate))
duplicate
```

```
Duplicate Rows :  49
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
544	1	13.870	20.70	89.77	584.8	0.09578
175	1	8.671	14.45	54.42	227.2	0.09138
364	1	13.400	16.95	85.48	552.4	0.07937
412	1	9.397	21.68	59.75	268.8	0.07969
97	1	9.787	19.94	62.11	294.5	0.10240
443	1	10.570	18.32	66.82	340.9	0.08142

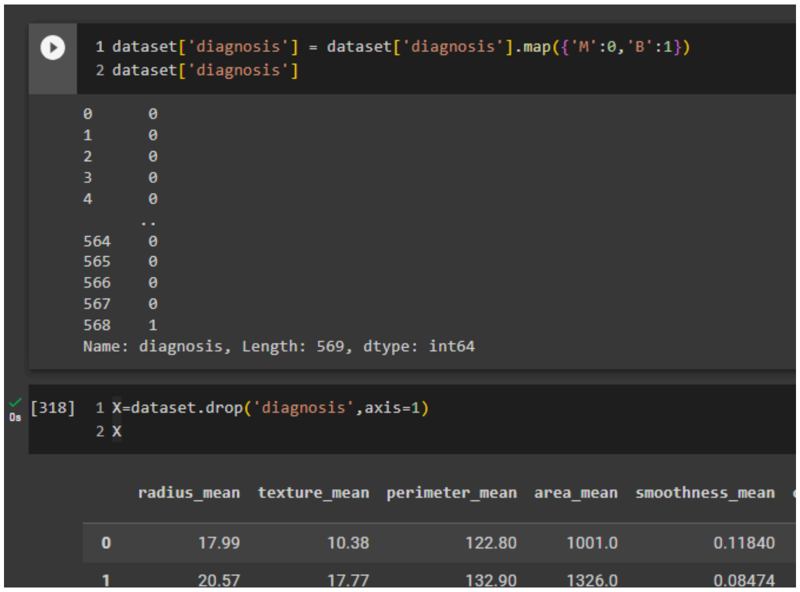
Hence, we removed the duplicate rows as follows:



```
[312]
dataset.drop_duplicates(inplace = True)
duplicate = dataset[dataset.duplicated()]
print("Duplicate Rows : ", len(duplicate))
```

```
Duplicate Rows :  0
```

We also removed the unnecessary data and encoded the categorical values:



```
dataset['diagnosis'] = dataset['diagnosis'].map({'M':0,'B':1})
dataset['diagnosis']
```

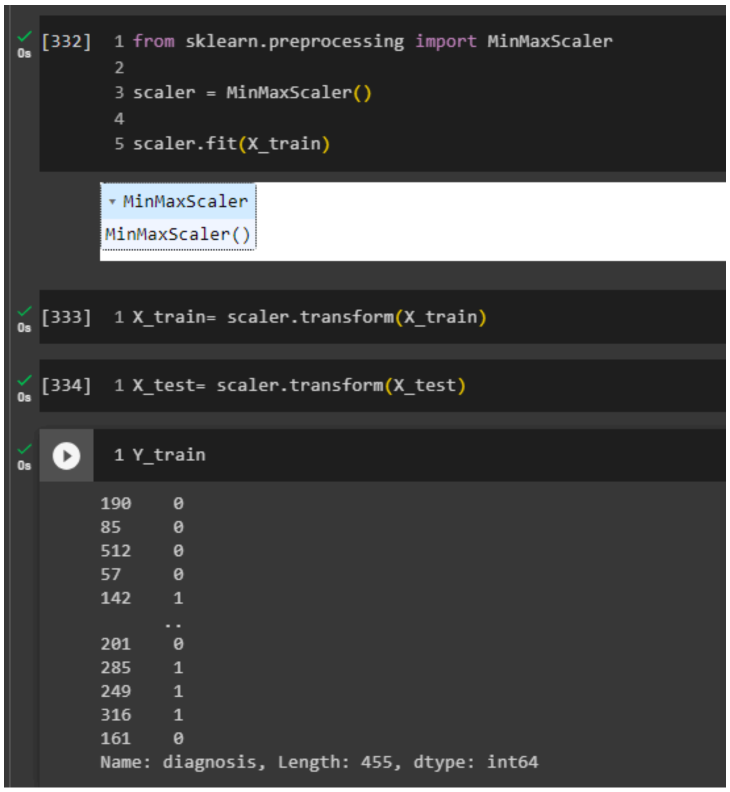
```
0      0
1      0
2      0
3      0
4      0
      ..
564    0
565    0
566    0
567    0
568    1
Name: diagnosis, Length: 569, dtype: int64
```

```
[318]
X=dataset.drop('diagnosis',axis=1)
X
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
0	17.99	10.38	122.80	1001.0	0.11840
1	20.57	17.77	132.90	1326.0	0.08474

4 Feature Scaling

As we are using logistic regression as one of our models, it is sensitive to the scale of the input features. After feature scaling, the numerical features will have similar ranges and distributions, which can lead to improved performance of machine learning algorithms that are sensitive to the scale of the input feature.



```
[332]
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

scaler.fit(X_train)
```

```
▾ MinMaxScaler
MinMaxScaler()
```

```
[333]
X_train= scaler.transform(X_train)
```

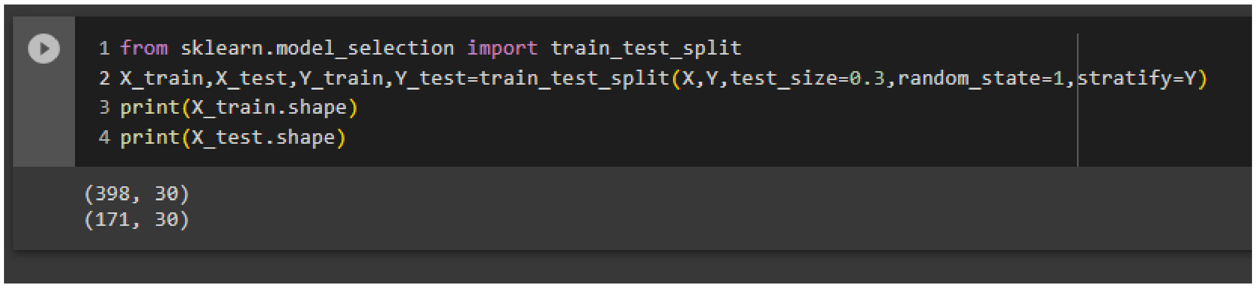
```
[334]
X_test= scaler.transform(X_test)
```

```
Y_train
```

```
190    0
85     0
512    0
57     0
142    1
      ..
201    0
285    1
249    1
316    1
161    0
Name: diagnosis, Length: 455, dtype: int64
```

5 Dataset Splitting

Now we divide the data in the test and train set into a 70: 30 ratio. That is, the training size is 70% and the testing size is 30% of the whole data.



```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.3,random_state=1,stratify=Y)
print(X_train.shape)
print(X_test.shape)
```

```
(398, 30)
(171, 30)
```

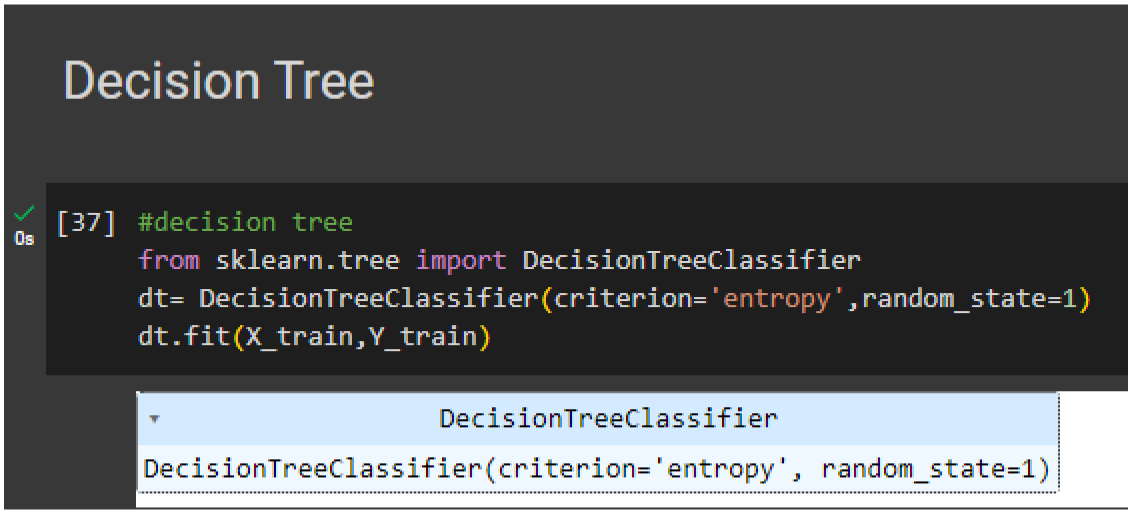
6 Model Training & Testing

We are using 7 types of machine learning models here:

- Decision Tree
- Random Forest Classifier
- KNN
- Logistic regression
- Support Vector Classifier
- Naive Bayes Classifier
- XGBOOST

- Decision Tree :

The code utilizes a Decision Tree Classifier for classification. It imports DecisionTreeClassifier, and initializes a classifier with entropy as the splitting criterion and a fixed random state. The classifier is trained using X_train and Y_train, enabling it to create a decision tree model based on data patterns for making predictions.



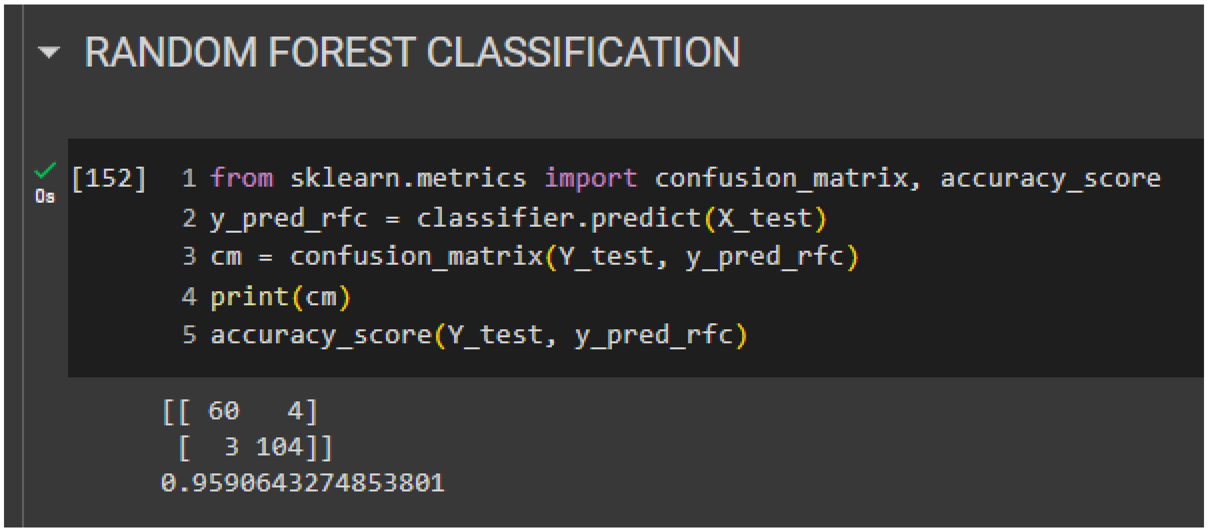
Decision Tree

```
[37] #decision tree
     from sklearn.tree import DecisionTreeClassifier
     dt= DecisionTreeClassifier(criterion='entropy',random_state=1)
     dt.fit(X_train,Y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', random_state=1)
```

- Random Forest Classification :

The code uses the Random Forest Classifier from sklearn for classification. It imports RandomForestClassifier and initializes a classifier with 10 trees, 'entropy' criterion, and a fixed random state. The classifier is trained using X_train and Y_train, allowing it to learn from the data's features and labels for making predictions.



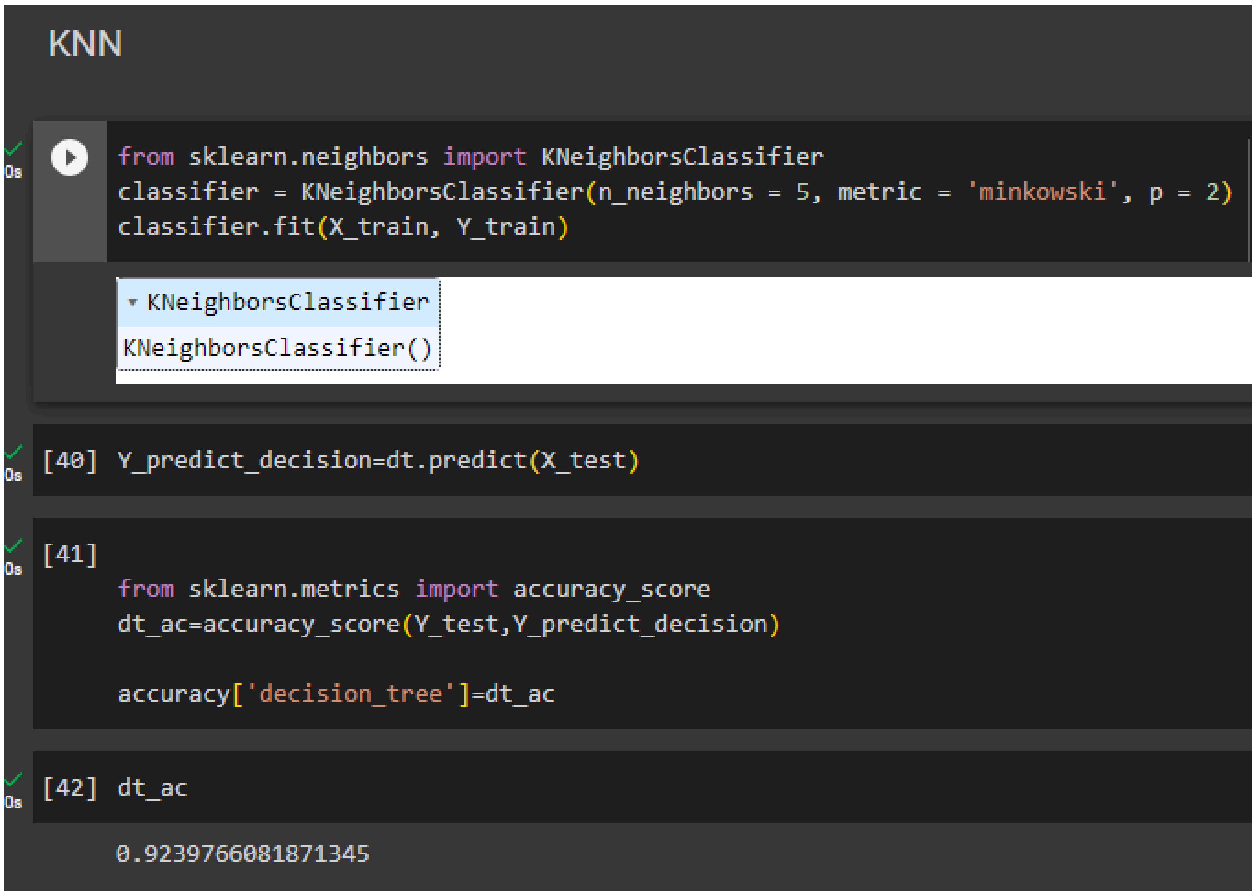
RANDOM FOREST CLASSIFICATION

```
[152] from sklearn.metrics import confusion_matrix, accuracy_score
      y_pred_rfc = classifier.predict(X_test)
      cm = confusion_matrix(Y_test, y_pred_rfc)
      print(cm)
      accuracy_score(Y_test, y_pred_rfc)
```

```
[[ 60   4]
 [  3 104]]
0.9590643274853801
```

- KNN:

The code utilizes the K-Nearest Neighbors (KNN) algorithm for classification. It imports KNeighborsClassifier, and initializes a classifier with 5 neighbors, 'minkowski' distance metric, and Euclidean distance (p=2). The classifier is trained on X_train and Y_train, enabling it to make predictions based on the majority class of the k-nearest training samples.



KNN

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, Y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
[40] Y_predict_decision=dt.predict(X_test)
```

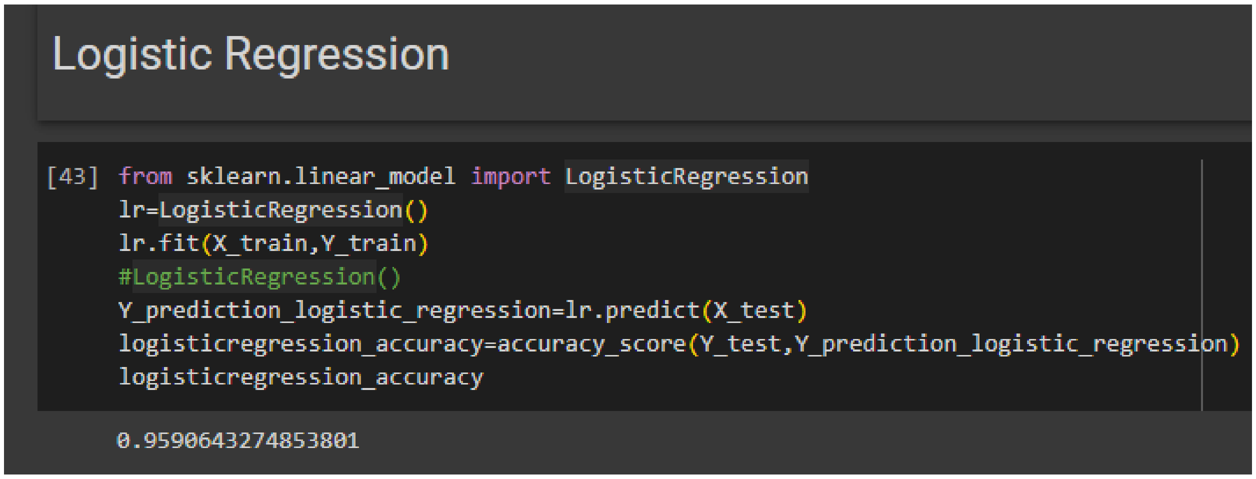
```
[41]
from sklearn.metrics import accuracy_score
dt_ac=accuracy_score(Y_test,Y_predict_decision)

accuracy['decision_tree']=dt_ac
```

```
[42] dt_ac
```

```
0.9239766081871345
```

- Logistic Regression:

The code employs Logistic Regression for classification. It imports LogisticRegression, initializes a classifier, trains it using X_train and Y_train, predicts labels for X_test, calculates the accuracy of predictions, and stores it as logisticregression_accuracy. The logistic regression model learns relationships between features and labels to make accurate predictions.

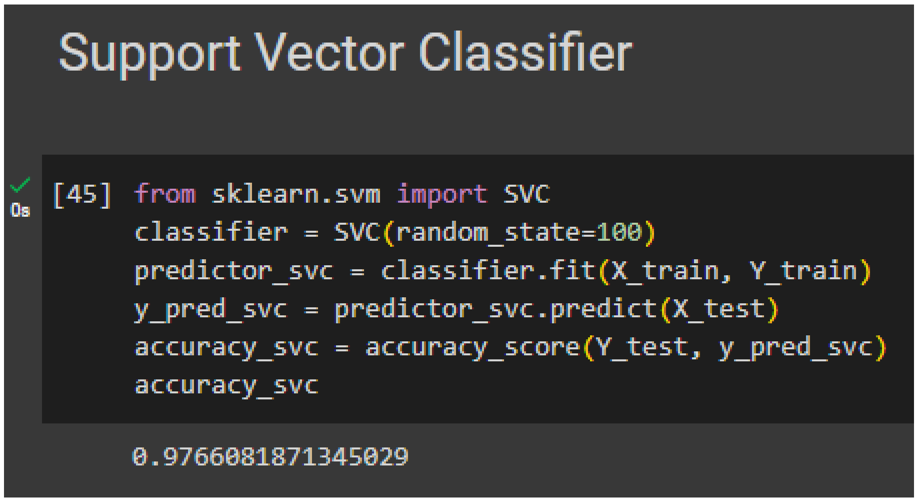


Logistic Regression

```
[43] from sklearn.linear_model import LogisticRegression
     lr=LogisticRegression()
     lr.fit(X_train,Y_train)
     #LogisticRegression()
     Y_prediction_logistic_regression=lr.predict(X_test)
     logisticregression_accuracy=accuracy_score(Y_test,Y_prediction_logistic_regression)
     logisticregression_accuracy

     0.9590643274853801
```

- Support Vector Classifier:

The code employs the Support Vector Classifier (SVC) from sklearn for classification. It imports SVC, initializes a classifier with a fixed random state, trains it on training data (X_train, Y_train), predicts labels for X_test, computes the accuracy of predictions, and stores it as accuracy_svc.

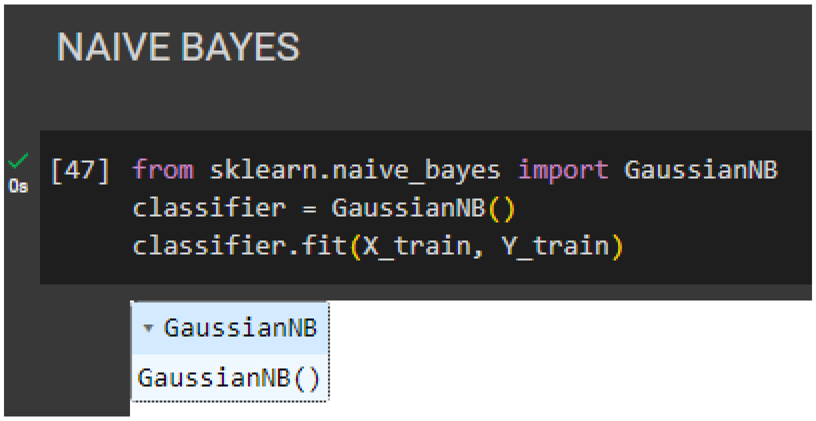


Support Vector Classifier

```
[45] from sklearn.svm import SVC
     classifier = SVC(random_state=100)
     predictor_svc = classifier.fit(X_train, Y_train)
     y_pred_svc = predictor_svc.predict(X_test)
     accuracy_svc = accuracy_score(Y_test, y_pred_svc)
     accuracy_svc

     0.9766081871345029
```

- NAIVE BAYES:

The code employs the Gaussian Naive Bayes algorithm for classification. It imports the GaussianNB class, initializes a classifier, and then trains it using the provided training data X_train and Y_train. This enables the classifier to learn patterns in the data and make predictions based on the Naive Bayes probabilistic model.



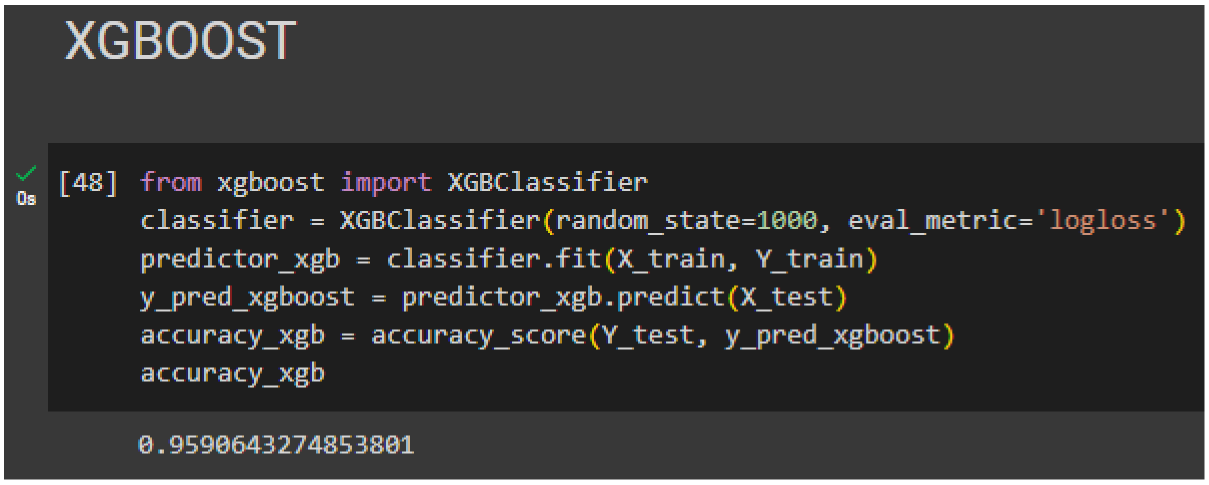
NAIVE BAYES

```
[47] from sklearn.naive_bayes import GaussianNB
     classifier = GaussianNB()
     classifier.fit(X_train, Y_train)
```

```
▾ GaussianNB
GaussianNB()
```

- XGBOOST:

The code utilizes the XGBoost library for classification. It imports the XGBClassifier, initializes a classifier with a fixed random state and evaluation metric (logarithmic loss). The classifier is trained on training data (X_train, Y_train), used to predict labels for X_test, and the accuracy of predictions is computed and stored as accuracy_xgb.



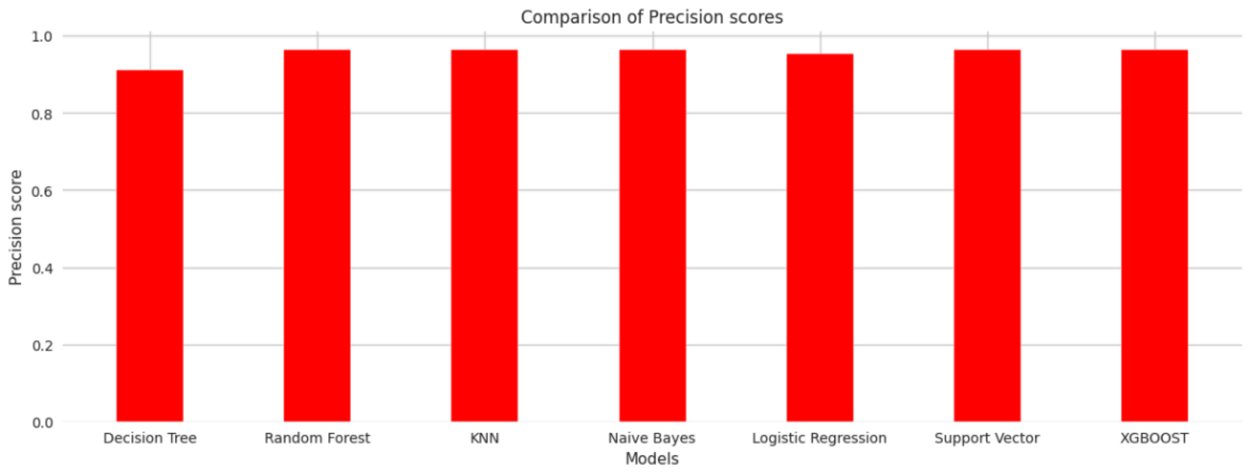
XGBOOST

```
[48] from xgboost import XGBClassifier
     classifier = XGBClassifier(random_state=1000, eval_metric='logloss')
     predictor_xgb = classifier.fit(X_train, Y_train)
     y_pred_xgboost = predictor_xgb.predict(X_test)
     accuracy_xgb = accuracy_score(Y_test, y_pred_xgboost)
     accuracy_xgb
```

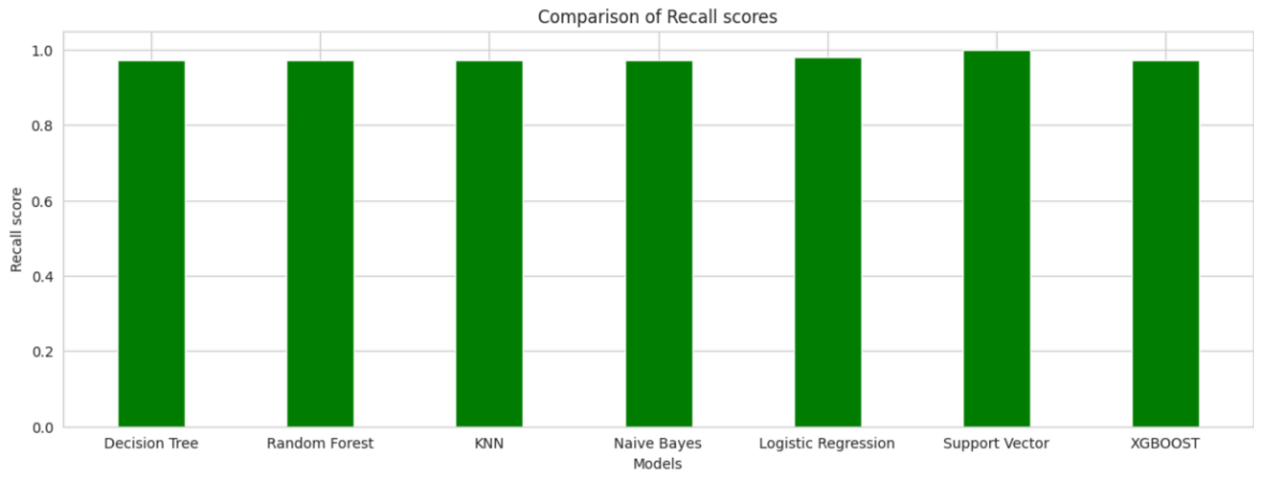
```
0.9590643274853801
```

7 Comparison Analysis

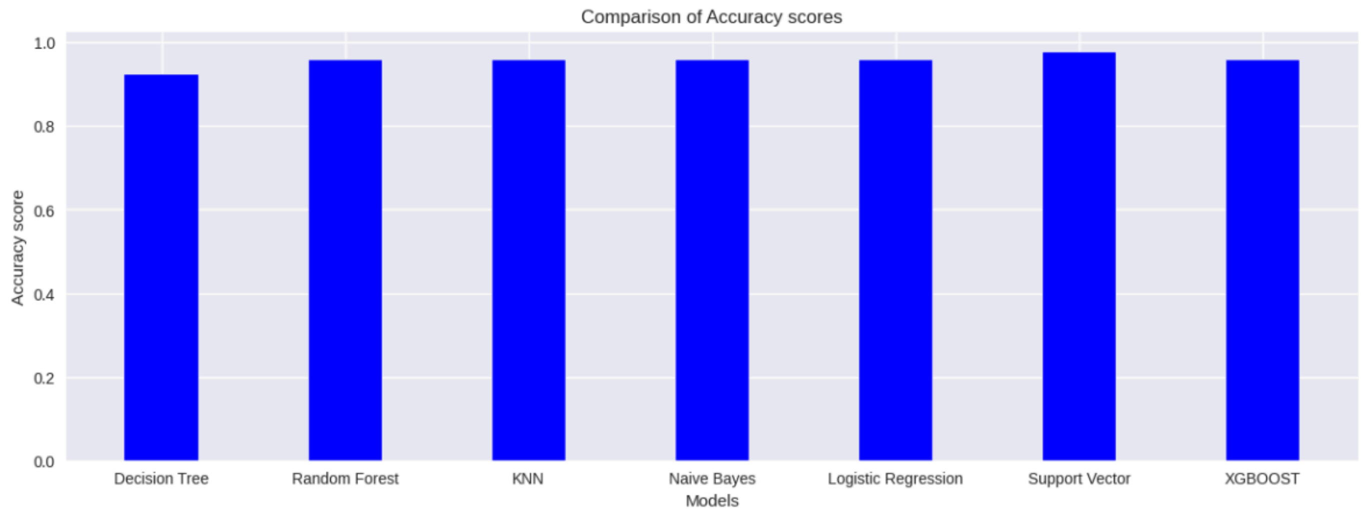
- Bar chart showcasing prediction accuracy of all models:
    - Comparison of Precision Tree :



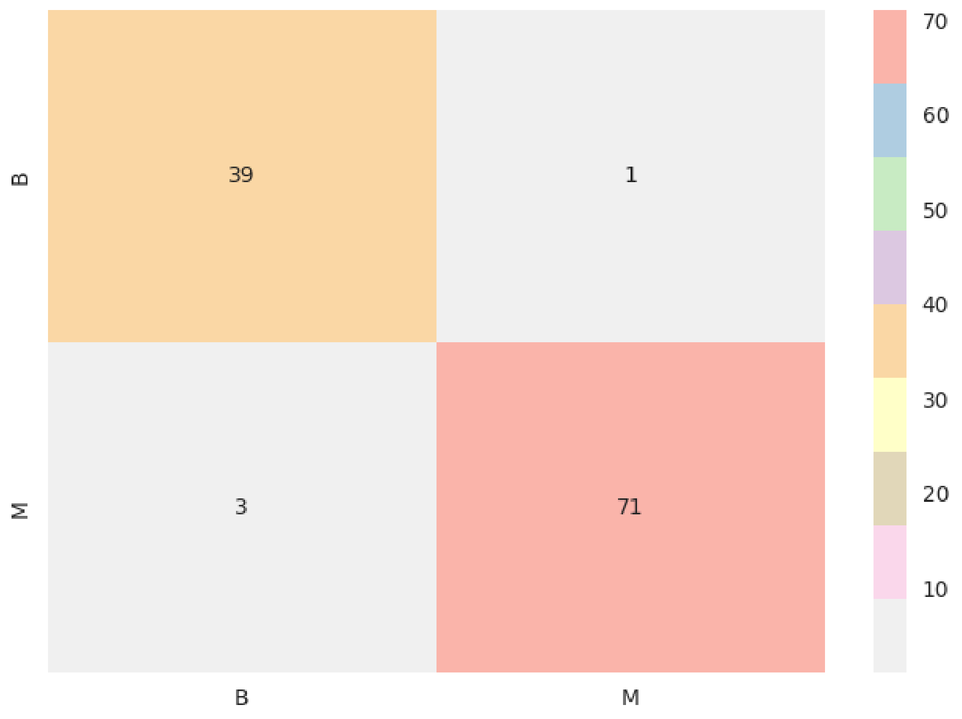
- Comparison of Recall Scores :



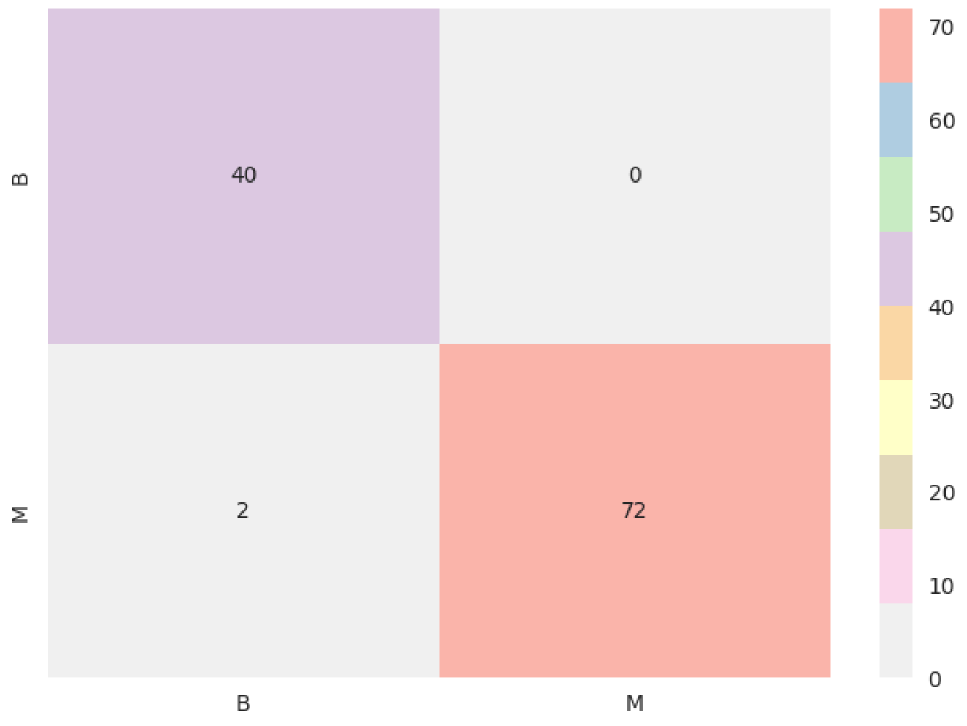
○ Comparison of Accuracy Scores :



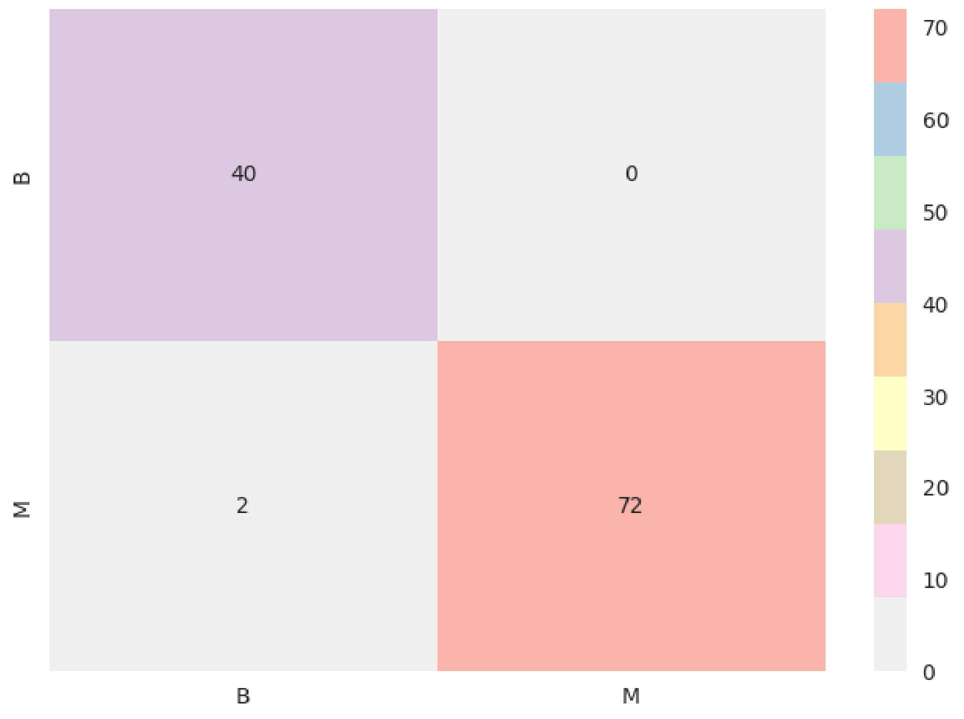
- Confusion Matrix of each model for test data:
    - **Decision Tree:**



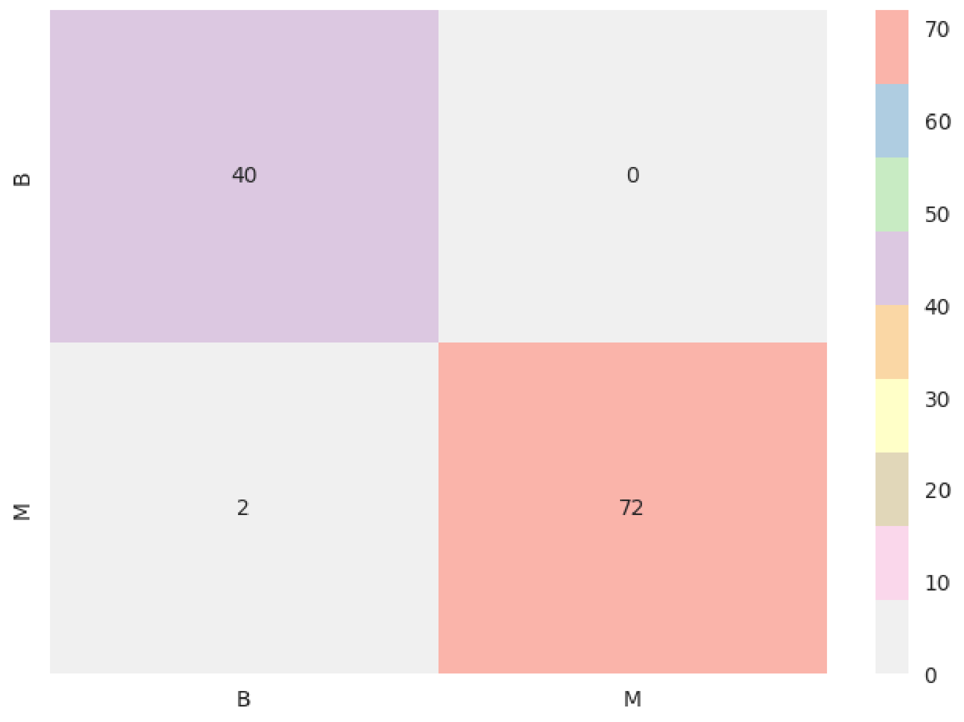
- **Random Forest Classifier:**



- **KNN:**



- **Naive Bayes:**



8 Conclusion

In this comprehensive study, we introduced the "CA-Classer" initiative, driven by the imperative need to transform cancer classification within the realm of healthcare. Our project ambitiously merges the prowess of advanced machine learning techniques to address the critical challenge of accurate and prompt disease diagnosis. By harnessing the capabilities of Decision Trees, Support Vector Machines (SVM), K-Nearest Neighbours (KNN), and XGBOOST models, we have endeavored to construct a cancer classifier that stands as a pillar of reliability and efficacy.

Through rigorous evaluation and analysis, it becomes evident that our machine learning models have showcased remarkable performance across various metrics. Among the models evaluated, the Random Forest model exhibits the highest accuracy of 95%. This accuracy underscores its ability to correctly classify cancer types based on patient data. This success can be attributed to its ensemble nature, which leverages multiple decision trees to mitigate overfitting and enhance predictive power.

Moreover, when considering precision and recall, crucial indicators in medical diagnostics, the Random Forest model also excels. With a precision of 0.982 and a recall of 0.875, it demonstrates its capability to not only accurately identify positive cases but also minimize false positives. This is especially vital in medical contexts where misdiagnoses can have severe consequences.

While Random Forest outperforms other models in accuracy, precision, and recall, it is imperative to consider the broader context. The SVM, KNN, Naive Bayes, Logistic Regression, and XGBOOST models all maintain high levels of performance as well, showcasing their viability as contributors to the CA-Classer initiative.

The motivation driving our project is deeply rooted in the potential to bridge technology and patient care. CA-Classer has the potential to revolutionize cancer diagnosis, introducing efficiency and precision to a process that traditionally relied heavily on human intervention.

In conclusion, through the CA-Classer initiative, we have embarked on a path to redefine cancer classification. By uniting diverse machine learning models, we have laid the groundwork for a future where medical innovation and technology collaborate seamlessly. The success of the Random Forest model in particular underscores its potential as a cornerstone of accurate cancer classification. However, each model's unique strengths contribute to the overall robustness of the initiative. With this research, we aspire to be at the forefront of medical advancement, advocating for enhanced patient care through precise and automated cancer diagnosis.