REPORT CHRONIC KIDNEY DISEASE

Members:

Rabeb Sdiri
Wajdi Hajji
Mohamed Salah Chafrouda
Abdelkerim Dassi
Ayoub Yahyaoui

We have conducted this project using the CRISP DM methodology which is composed of 3 parts : Business understanding, Data understanding and Data preparation.

1- Business Understanding:

Chronic kidney disease (CKD) is a type of kidney disease in which there is gradual loss of kidney function over a period of time ; from months to years. That's why it has received much attention. In fact, it is among the top 10 leading causes of death in the world. There are two leading causes of CKD, i.e. diabetes and hypertension. Initially, there are generally no symptoms but later, symptoms may include leg swelling, feeling tired, vomiting, loss of appetite, and confusion. Complications include an increased risk of heart disease, high blood pressure, bone disease, and anemia. CKD is associated with a decrease in kidney function related to age and is accelerated in hypertension, diabetes, obesity, and primary kidney disorders. CKD is a global health problem with a high morbidity and mortality rate, and it induces other diseases. As there are no obvious symptoms during the early stages of CKD, patients often do not notice the disease, this being the main feature, eventually leading to a complete loss of kidney function. Early detection of CKD allows patients to receive timely treatment to improve the progression of this disease. Therefore, an automated and accurate diagnosis method of CKD is necessary to assist medical staff to contribute to the reduction of significant complications in the disease. The main criterion of success for this project, with the help of machine learning, is to enhance the quality of chronic kidney disease classification .

2-Data understanding:

Our CKD dataset was collected from 400 patients from the University of California, Irvine Machine Learning Repository: 250 instances was chronic kidney disease (ckd) and 150 instances was nonchronic kidney disease (notckd). There were 25 attributes, including the class attribute.
First, we will explain each feature :
`Age` : the patient's age .
`Blood Pressure:` The pressure in the arteries when the heart rests between beats. This is the time when the heart fills with blood and gets oxygen. This is what your diastolic blood pressure number means:

- Normal : Lower than 80
- Stage 1 : Hypertension: 80-89
- Stage 2 : Hypertension: 90 or more
- Hypertensive crisis: 120 or more.

`Specific Gravity:` Urine specific gravity is a laboratory test that shows the concentration of all chemical particles in the urine .
`Albumin:` Albumin is a protein found in the blood. The albumin urine test measures the amount of albumin in a urine sample. Albuminuria is a sign of kidney disease and means that you have too much albumin in your urine. A healthy kidney doesn't let albumin pass from the blood

into the urine. A damaged kidney lets some albumin pass into the urine. The less albumin in your urine, the better.
`sugar :` The glucose urine test measures the amount of sugar (glucose) in a urine sample. The presence of glucose in the urine is called glycosuria or glucosuria. Glucose is not usually found in urine. If it is, further testing is needed. Normal glucose range in urine: 0 to 0.8 mmol/l (0 to 15 mg/dL).
`red blood cell count :` Red blood cells are one of the major components of blood, along with white blood cells and platelets. Red blood cells help carry oxygen throughout the body. A high red blood cell count means the number of red blood cells in your bloodstream is higher than normal. Normal red blood cell counts are :

- For men, 4.7 to 6.1 million red blood cells per microliter of blood.
- For women, 4.2 to 5.4 million red blood cells per microliter of blood.
- For children, 4.0 to 5.5 million red blood cells per microliter of blood.

`ur_puscell / ur pus_cell clumps:` Presence of pus cells in urine defined as pyuria is an important accompaniment of bacteriuria which may be asymptomatic or can indicate toward underlying urinary tract infection.

`ur_bacteria :` Presence of bacteria in urine .

`blood glucose random / diabetes:` Random glucose testing is a blood test done at a random moment of the day to check glucose (sugar) levels. Values of 200 mg/dL or above can indicate diabetes.

`blood urea :` A blood urea nitrogen (BUN) test measures the amount of nitrogen in your blood that comes from the waste product urea. Urea is made when protein is broken down in your body. Urea is made in the liver and passed out of your body in the urine. A BUN test is done to see how well your kidneys are working. In general, around 6 to 24 mg/dL (2.1 to 8.5 mmol/L) is considered normal.

`serum creatinine :` Creatinine is a waste product that comes from the normal wear and tear on muscles of the body. Everyone has creatinine in their bloodstream. The normal level of creatinine depends on your age, race, gender, and body size. In general , a normal result is 0.7 to 1.3 mg/dL for men and 0.6 to 1.1 mg/dL for women.

`sodium :` A normal blood sodium level is between 135 and 145 milliequivalents per liter (mEq/L). Hyponatremia occurs when the sodium in your blood falls below 135 mEq/L.

`potassium` **:** Normally, your blood potassium level is 3.6 to 5.2 millimoles per liter (mmol/L). A very low potassium level (less than 2.5 mmol/L) can be life-threatening and requires urgent medical attention.

`hemoglobin / anemia` **:** A hemoglobin test measures the amount of hemoglobin in your blood. Hemoglobin is a protein in your red blood cells that carries oxygen to your body's organs and tissues and transports carbon dioxide from your organs and tissues back to your lungs. An Hb value less than 5.0 g/dL can lead to heart failure and death. The normal range for hemoglobin is:

- For men, 13.5 to 17.5 grams per deciliter.
- For women, 12.0 to 15.5 grams per deciliter.
- Anything below is considered a form of anemia.

`packed cell volume` **:** The packed cell volume (PCV) is a measurement of the proportion of blood that is made up of cells. The value is expressed as a percentage or fraction of cells in blood. For example, a PCV of 40% means that there are 40 milliliters of cells in 100 milliliters of blood. Critical values are <18% and >55% (for adults).

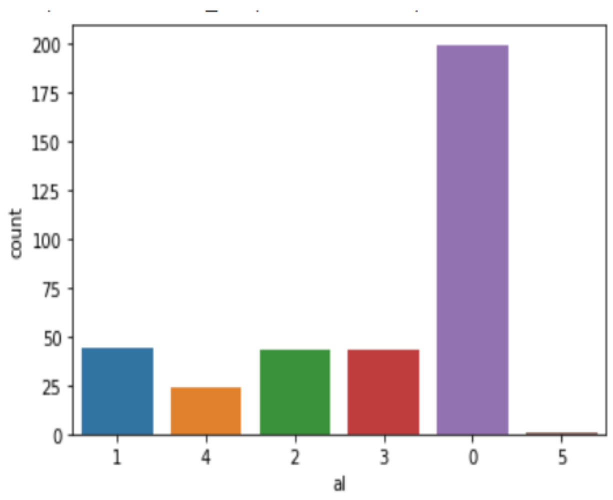
`white blood cell count`: White blood cell count varies from person to person , the normal range is usually between 4,000 and 11,000 white blood cells per microlitre of blood. Anything below 4,000 is typically considered to be a low white blood cell count.

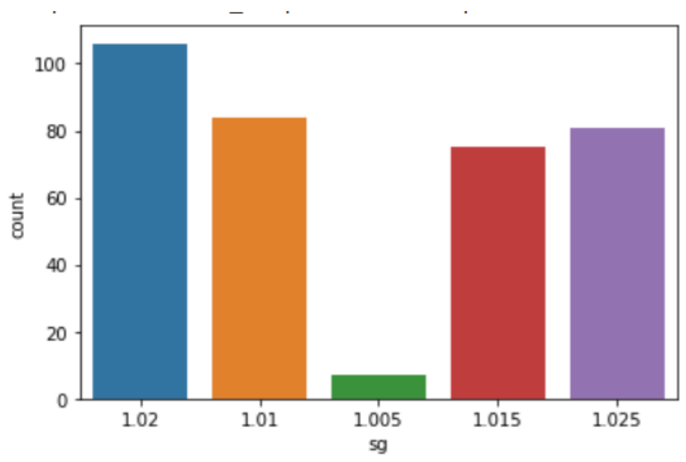
`coronary artery disease` **:** The coronary arteries supply blood, oxygen and nutrients to your heart. Coronary artery disease develops when the coronary arteries become damaged or diseased. The damage may be caused by various factors (Smoking / High blood pressure /High cholesterol /Diabetes /sedentary lifestyle...).

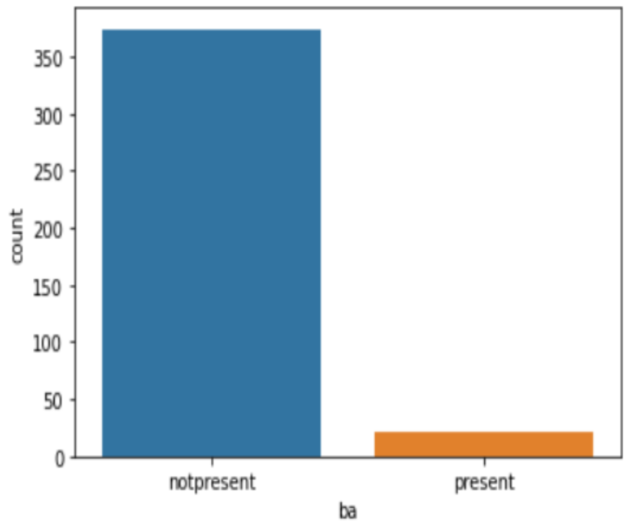
`appetit` **:** Appetite is the desire to eat food, sometimes due to hunger. A poor appetite is when your desire to eat is reduced. The medical term for a loss of appetite is anorexia. Any illness can reduce appetite. If the illness is treatable, the appetite should return when the condition is cured.

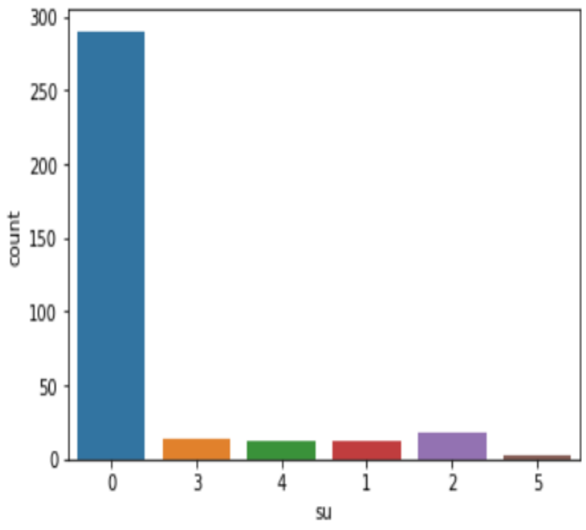
`pedal edema` **:** Pedal edema causes an abnormal accumulation of fluid in the ankles, feet, and lower legs causing swelling of the feet and ankles. Two mechanisms can cause edema of the feet. Two most common causes are being overweight and standing or sitting for long periods.

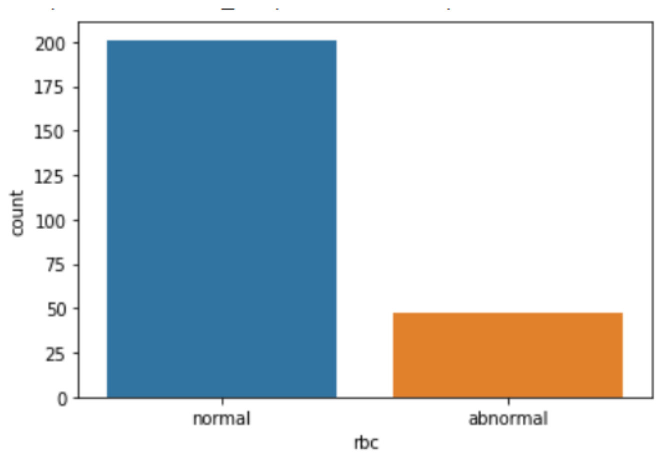
Before starting to process the data set, a set of visualizations is made to help better understand the characteristics of the information being worked with .

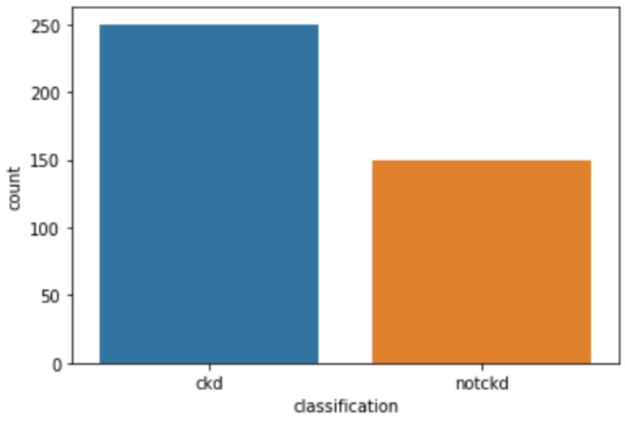


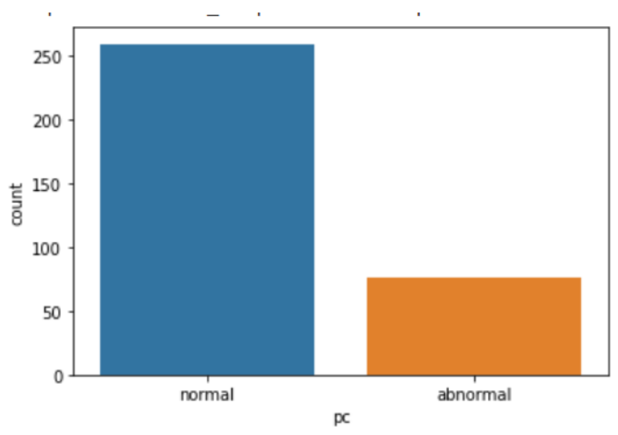


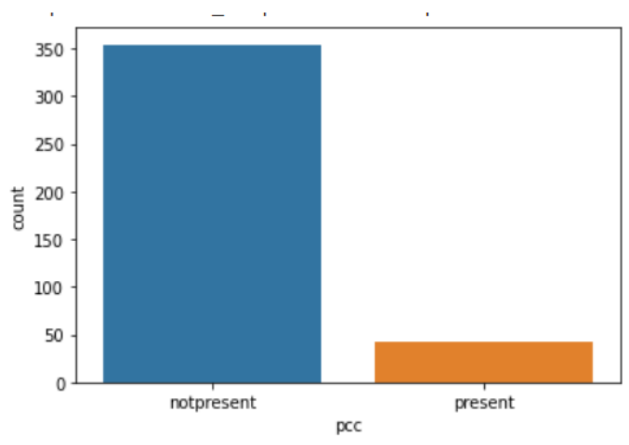


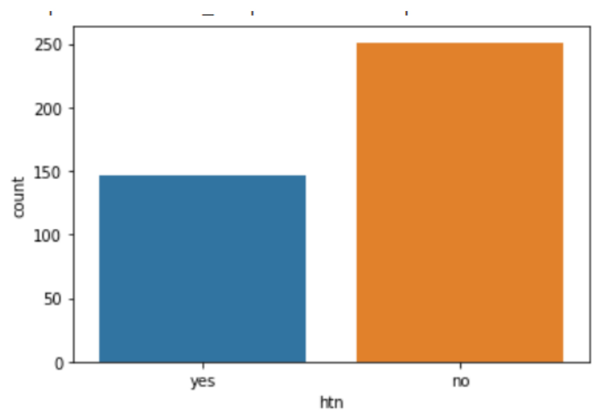


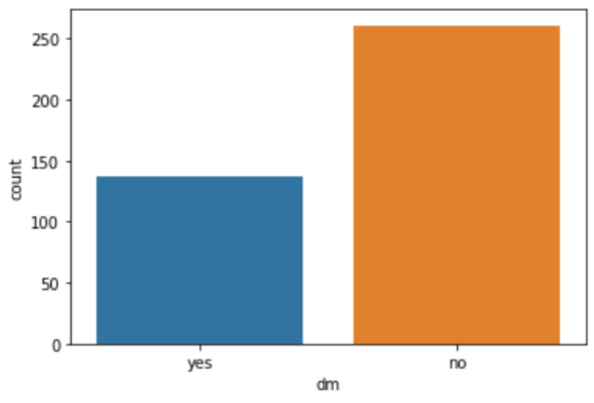


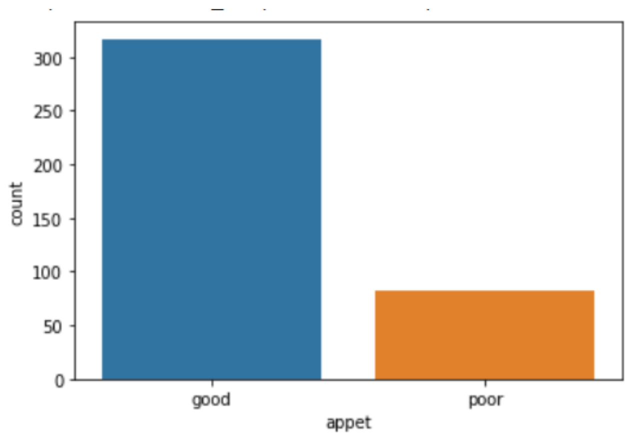


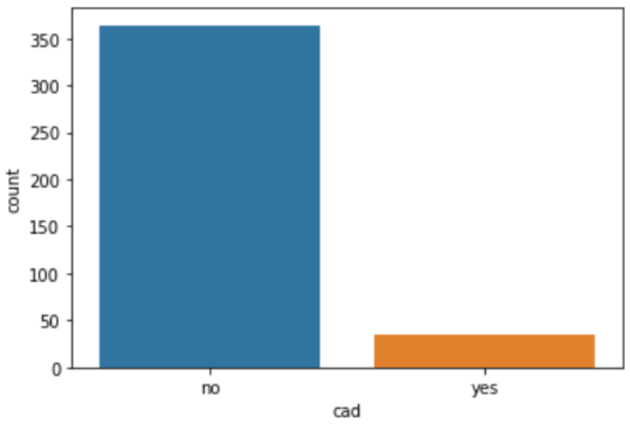

count
0
50
100
150
200
250
normal
abnormal
pc

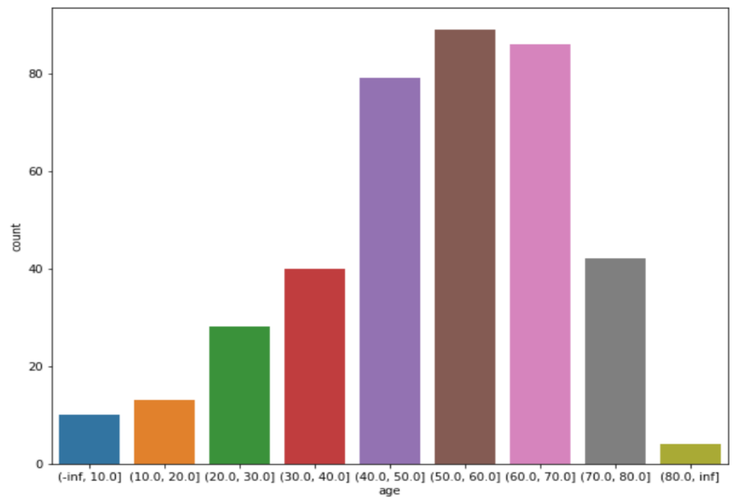

count
0
50
100
150
200
250
300
350
notpresent
present
pcc

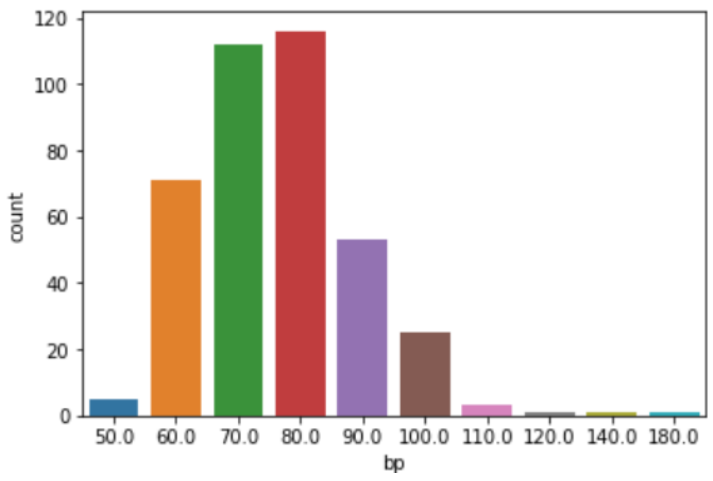

count
0
50
100
150
200
250
yes
no
htn


count
0
50
100
150
200
250
yes
no
dm


count
0
50
100
150
200
250
300
good
poor
appet


count
0
50
100
150
200
250
300
350
no
yes
cad

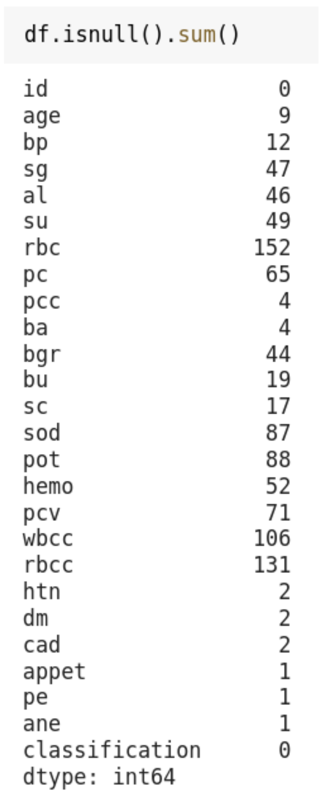




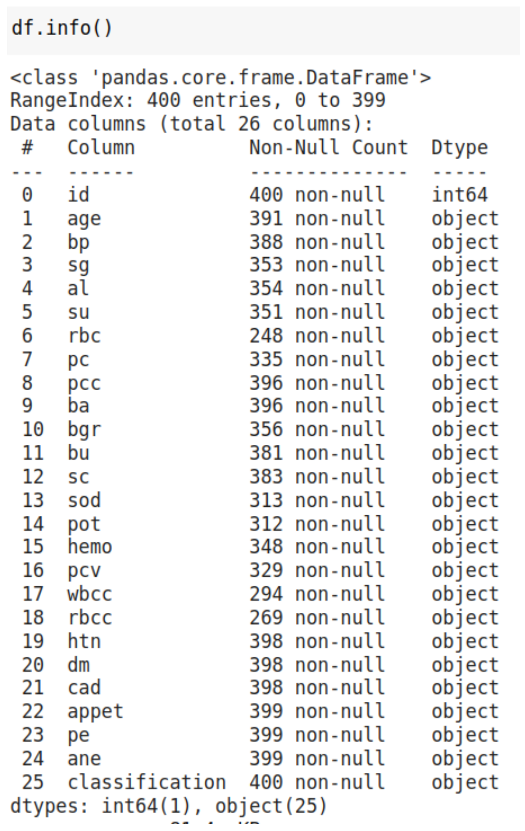
2-Data Preparation :

I. Data Cleaning :

After getting a clear understanding of our data, we explored it for any inconsistencies and missing data. We found that there is a lot of NaNs (missing values) and that supposedly numerical values are of type ‘object’ as seen in the figures below:



```
df.isnull().sum()

id                  0
age                 9
bp                 12
sg                 47
al                 46
su                 49
rbc               152
pc                 65
pcc                 4
ba                  4
bgr                44
bu                 19
sc                 17
sod                87
pot                88
hemo               52
pcv                71
wbcc              106
rbcc              131
htn                 2
dm                  2
cad                 2
appet               1
pe                  1
ane                 1
classification      0
dtype: int64
```



```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 26 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   id              400 non-null    int64
 1   age             391 non-null    object
 2   bp              388 non-null    object
 3   sg              353 non-null    object
 4   al              354 non-null    object
 5   su              351 non-null    object
 6   rbc             248 non-null    object
 7   pc              335 non-null    object
 8   pcc             396 non-null    object
 9   ba              396 non-null    object
 10  bgr             356 non-null    object
 11  bu              381 non-null    object
 12  sc              383 non-null    object
 13  sod             313 non-null    object
 14  pot             312 non-null    object
 15  hemo            348 non-null    object
 16  pcv             329 non-null    object
 17  wbcc            294 non-null    object
 18  rbcc            269 non-null    object
 19  htn             398 non-null    object
 20  dm              398 non-null    object
 21  cad             398 non-null    object
 22  appet           399 non-null    object
 23  pe              399 non-null    object
 24  ane             399 non-null    object
 25  classification  400 non-null    object
dtypes: int64(1), object(25)
```

number of null values per feature

general information concerning our dataset

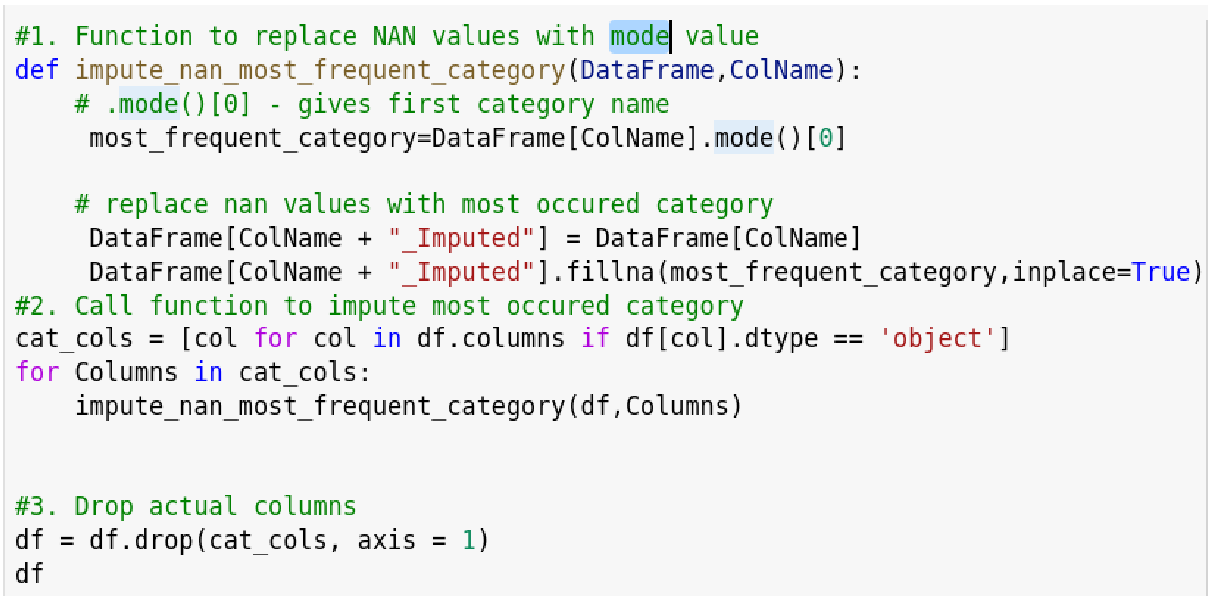
the approach we opted for was to convert the numeric values of type object to floats :

```
num_cols = ['age','bp','rbcc','bgr','bu','sc','sod','pot','hemo','pcv','wbcc']
```

```
df[num_cols]=df[num_cols].apply(pd.to_numeric, errors='ignore')
```

In medical field, numeric data is crucial, treatments and decisions are basically based upon it, for that particular reason we determined that a better approach to treating the NAn values of the numeric data other than ignoring the records, is to replace them with the mean value of their respective feature, as it gives a close estimation of the patients health situation.

As for the categorical data, what we did was basically determine the most frequent state of each particular column and replace the NAn values with it. it serves as a good estimation for our next steps:



```
#1. Function to replace NAN values with mode value
def impute_nan_most_frequent_category(DataFrame,ColName):
    # .mode()[0] - gives first category name
     most_frequent_category=DataFrame[ColName].mode()[0]

    # replace nan values with most occured category
     DataFrame[ColName + "_Imputed"] = DataFrame[ColName]
     DataFrame[ColName + "_Imputed"].fillna(most_frequent_category,inplace=True)
#2. Call function to impute most occured category
cat_cols = [col for col in df.columns if df[col].dtype == 'object']
for Columns in cat_cols:
    impute_nan_most_frequent_category(df,Columns)


#3. Drop actual columns
df = df.drop(cat_cols, axis = 1)
df
```

II. Feature Selection :

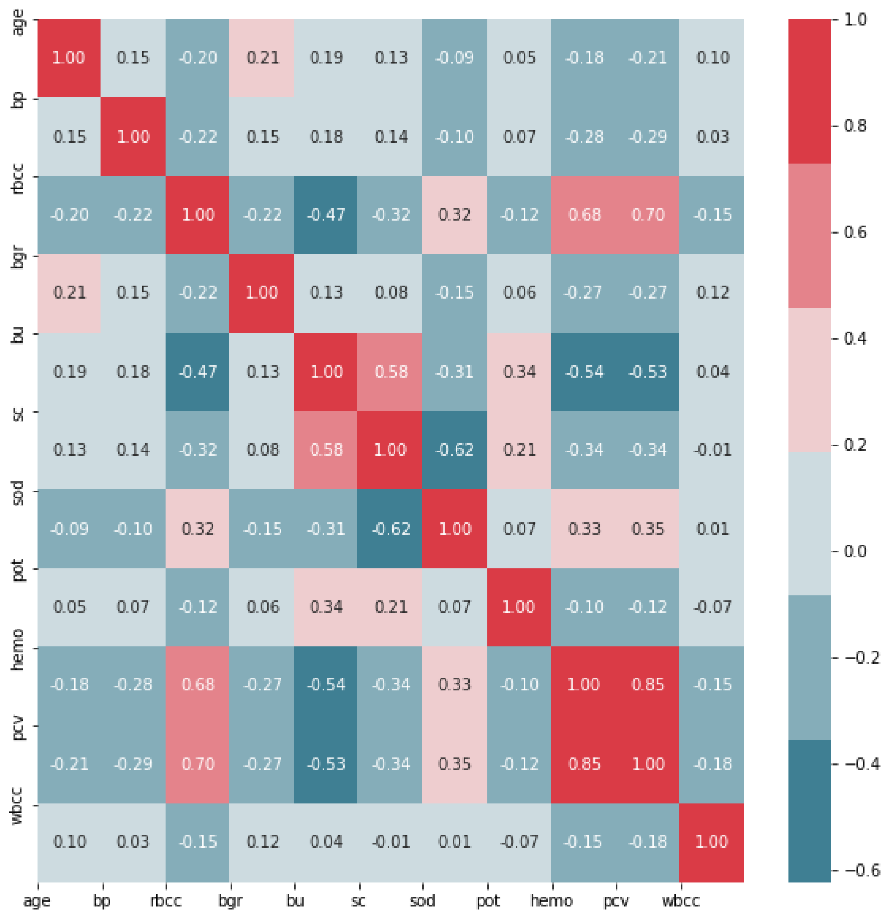
1. LAB1: Recursive feature elimination with cross-validation

After exploring our data we treated the null values with appropriate imputations. Then, we analyzed our variables to understand which features will lead to an efficient prediction model. This art of choosing the minimum number of variables that can create the most accurate predictive model is called feature selection in machine learning.
It is an excellent practice to work with a minimum set of predictive modeling features as they significantly reduce the algorithm's complexity and computational costs.

In this study, we used the RFE algorithm to extract the most important features of a prediction. Technically, The Recursive Feature Elimination (RFE) is a wrapper-style feature selection algorithm that also uses filter-based feature selection internally.
RFE is popular because it is easy to configure and use and because it is effective at selecting those features (columns) in a training dataset that are more or most relevant in predicting the target variable.
There are two important configuration options when using RFE: the choice in the number of features to select and the choice of the algorithm used to help choose features. Both of these hyperparameters can be explored, although the performance of the method is not strongly dependent on these hyperparameters being configured well.
RFE works by searching for a subset of features by starting with all features in the training dataset and successfully removing features until the desired number remains.
This is achieved by fitting the given machine learning algorithm used in the core of the model, ranking features by importance, discarding the least important features, and re-fitting the model. This process is repeated until a specified number of features remains.
The figure1 shows the ranking of the features ordered by priority impact on the target value. It shows that the optimal number of features is 8. The feature we selected are (pcc, sod, hemo, pcv, htn, dm, pe, appet)
Then due to the correlation matrix we noticed that pcv and hemo have a correlation coefficient equal to 0.85 so we can drop one of them . we dropped pcv

```
[13 10 12 11  4  9  5  1  6 14  3  7  1  2  1  1 15 16  1  1 17  1  1  8]
```

Fig1.



	age	bp	rbcc	bgr	bu	sc	sod	pot	hemo	pcv	wbcc
age	1.00	0.15	-0.20	0.21	0.19	0.13	-0.09	0.05	-0.18	-0.21	0.10
bp	0.15	1.00	-0.22	0.15	0.18	0.14	-0.10	0.07	-0.28	-0.29	0.03
rbcc	-0.20	-0.22	1.00	-0.22	-0.47	-0.32	0.32	-0.12	0.68	0.70	-0.15
bgr	0.21	0.15	-0.22	1.00	0.13	0.08	-0.15	0.06	-0.27	-0.27	0.12
bu	0.19	0.18	-0.47	0.13	1.00	0.58	-0.31	0.34	-0.54	-0.53	0.04
sc	0.13	0.14	-0.32	0.08	0.58	1.00	-0.62	0.21	-0.34	-0.34	-0.01
sod	-0.09	-0.10	0.32	-0.15	-0.31	-0.62	1.00	0.07	0.33	0.35	0.01
pot	0.05	0.07	-0.12	0.06	0.34	0.21	0.07	1.00	-0.10	-0.12	-0.07
hemo	-0.18	-0.28	0.68	-0.27	-0.54	-0.34	0.33	-0.10	1.00	0.85	-0.15
pcv	-0.21	-0.29	0.70	-0.27	-0.53	-0.34	0.35	-0.12	0.85	1.00	-0.18
wbcc	0.10	0.03	-0.15	0.12	0.04	-0.01	0.01	-0.07	-0.15	-0.18	1.00

2. Lab2: Correlation-based feature selection

In our analysis, we utilized Correlation-based Feature Selection (CFS) as a method for selecting features. CFS assesses the relationship between each feature and the target variable, and selects the subset of features that are most strongly correlated with the target variable. This approach can help to identify the features that are most relevant for predicting the target variable, and can improve the performance of machine learning models by reducing the dimensionality of the data. Additionally, using CFS can make the model more interpretable, as it reduces the number of features and thus makes it easier to understand. The principle behind it is the calculation of the merit of a subset of features ‘s’ with ‘k’ features to determine the best subset that has the lowest correlation between them and the highest correlation with the target value:

$$Merit_s = \frac{k\overline{r_{cf}}}{\sqrt{k+k(k-1)\overline{r_{ff}}}}.$$

- $\overline{r_{ff}}$: average feature-feature correlation
- $\overline{r_{rf}}$: average feature-class correlation
- k: number of features of that subset

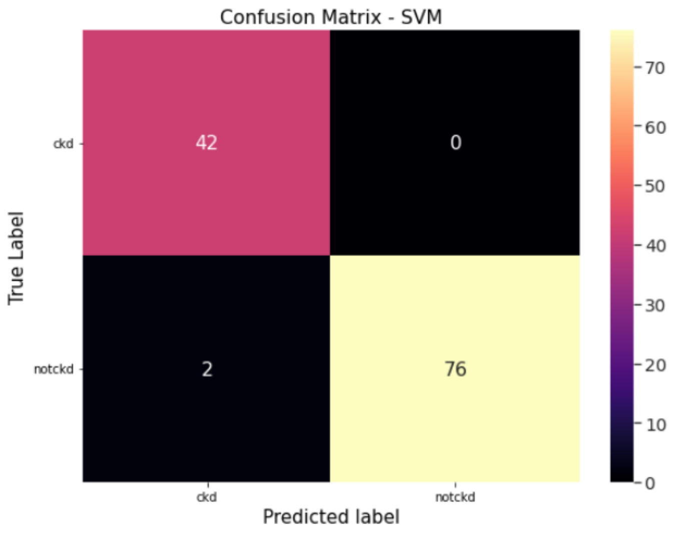
in our study, and with the help of fellow developers we implemented the code for the CFS and managed to extract the following subset of features :

```
htn_Imputed
sg_Imputed
dm_Imputed
al_Imputed
appet_Imputed
rbcc
pc_Imputed
ane_Imputed
pe_Imputed
bp
rbc_Imputed
pcv
cad_Imputed
```

III. Modeling:

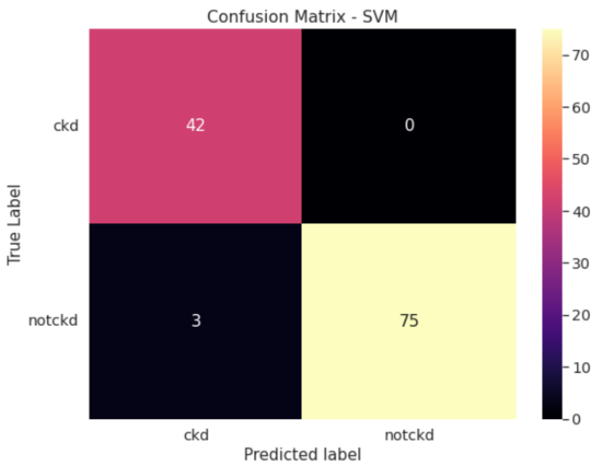
1. Support Vector Machine:

Support Vector Machines (SVMs) are a type of supervised machine learning algorithm that can be used for classification or regression tasks. The SVM algorithm would learn the relationship between the features and the labels of our dataset, and use this information to make predictions on new data. SVMs are often used for this type of task because they can effectively handle high-dimensional data and can make predictions with a high degree of accuracy.
the performance of the SVM model trained on all the features has produced an accuracy score of 98% (the percentage of the rightly predicted values) and precision of 95% with predictions being distributed as seen below in the confusion matrix:



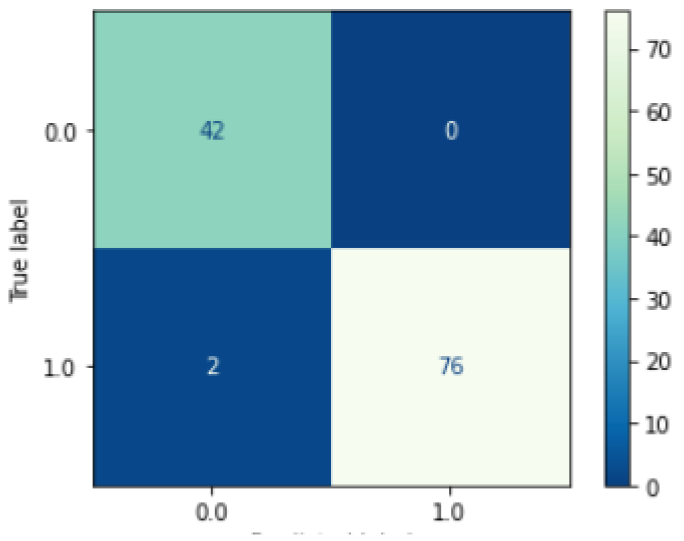
The SVM being so powerful with high dimensional data, it managed to do the job just fine with all the given features, but we will move forward with experimenting with some feature selections and see how the model behaves.

- First we tried **SVM with the RFECV** and got 97% accuracy and a precision of 93% even with **Adaboost** which trains the model on different subsets of the training data, and then combines their predictions to make a final prediction, The accuracy and precision of the predictions stayed the same and distributed as follows:



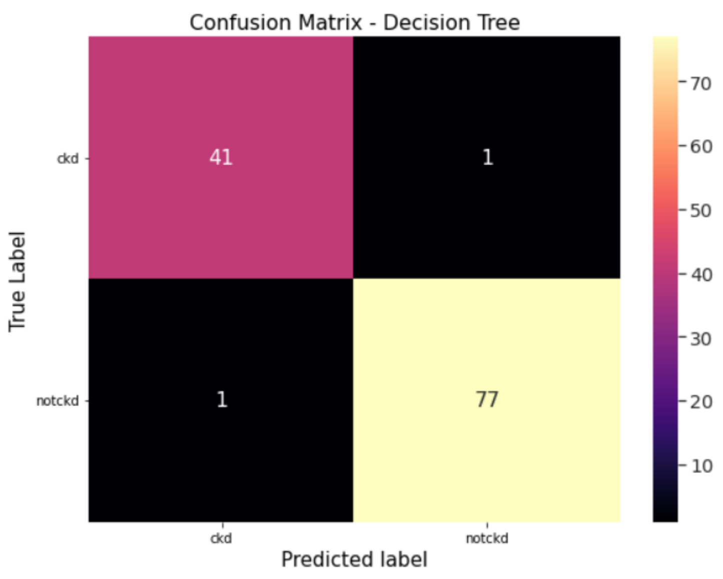
- ➔ There are several possible explanations for the deterioration of the model's performance with feature selection. One possibility is that the feature selection process removed some features that were actually useful for making predictions, which reduced the model's ability to accurately classify the data. Another possibility is that the remaining features were less correlated with the target variable, which made it more difficult for the model to learn the relationship between the features and the target variable.

- Secondly, we tried **SVM with Correlation-based feature selection** and it produced an accuracy of 98% and precision of 95%. By boosting the model with **Adaboost**, its performance barely improved which is understandable because our dataset is very small. below is the confusion matrix of SVM with CFS and Adaboost:

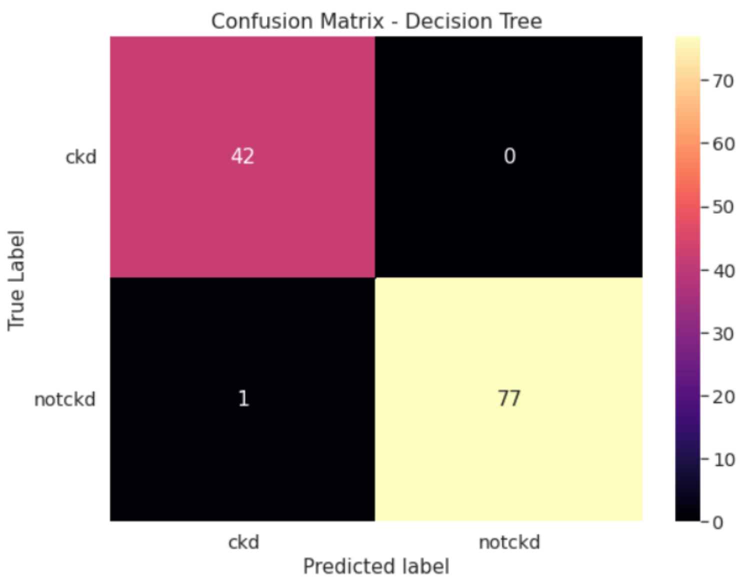


2. Decision Tree:

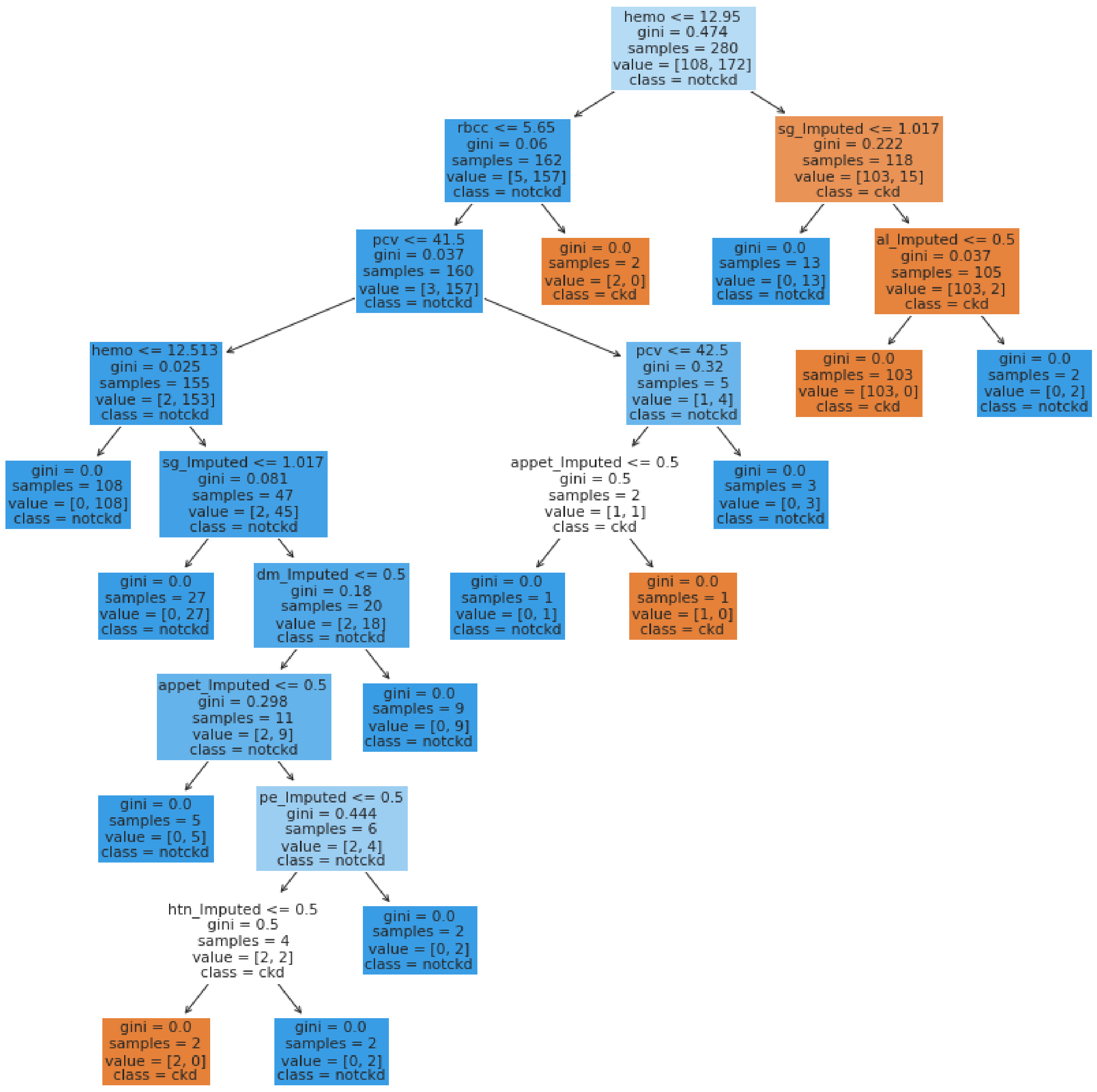
We have conducted a Decision Tree classification on our data.The decision tree classifier uses a series of if-then-else rules to predict the class of a given data point, based on the values of the features in that data point. The classifier is trained on a labeled dataset, where the correct class for each data point is known.
This model can diagnose Chronic Kidney Disease(CKD) at 0.98 accuracy. Below is details about predictions (Confusion Matrix)



Our Model is producing false negatives , which is not ideal in a medical setting. Therefore, it is in our best interest to further augment the accuracy of our model. Thus, we performed a process of feature selection using RFECV which helped improve the accuracy to 0.99 and eliminating the false negatives as shown in the figure below



The corresponding Tree rules to the model are as follows:



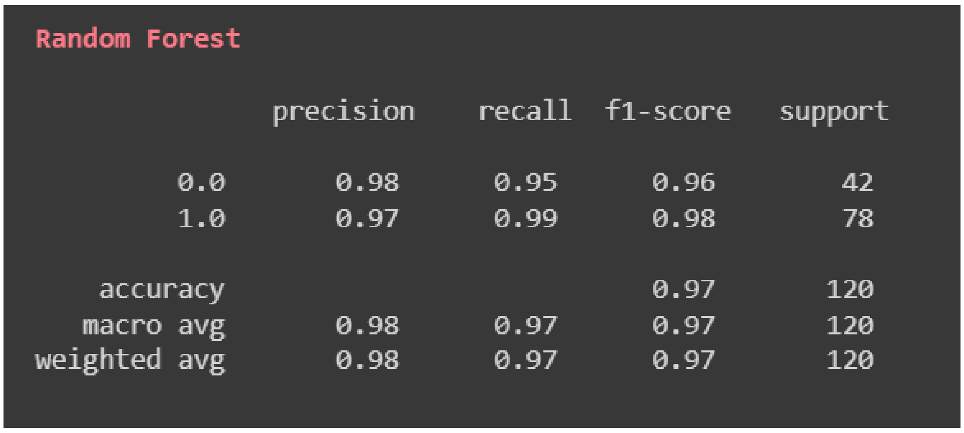
In an attempt to even further increase the accuracy of the model, we used AdaBoost . However, this has not proven to be useful and no fruitful accuracy improvements were recorded.

Random Forest :

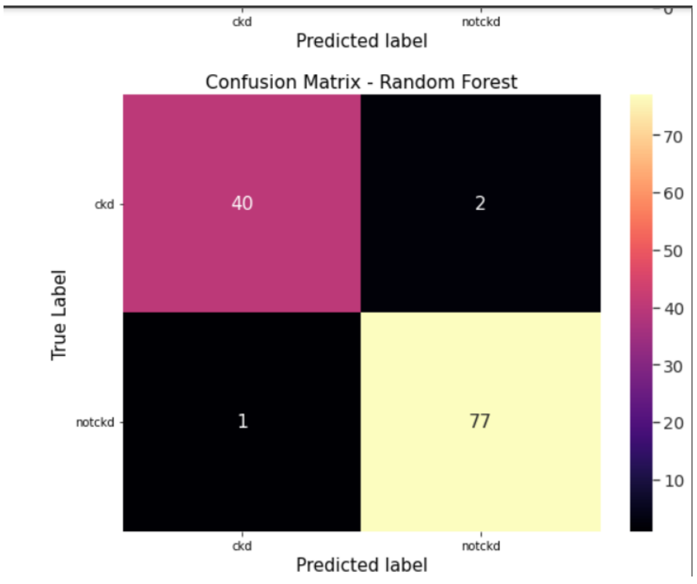
A random forest classifier is a type of machine learning algorithm that can be used for classification tasks. It is called a "random forest" because it uses multiple decision trees to

make predictions, and each tree is trained on a randomly selected subset of the data. By combining the predictions of multiple trees, a random forest classifier can often make more accurate predictions than a single decision tree.
One advantage of using a random forest classifier is that it can handle a large number of features, or variables, in the data. This is because each decision tree in the forest only uses a subset of the available features to make predictions. This means that the algorithm can handle high-dimensional data, which can be challenging for other types of classifiers.

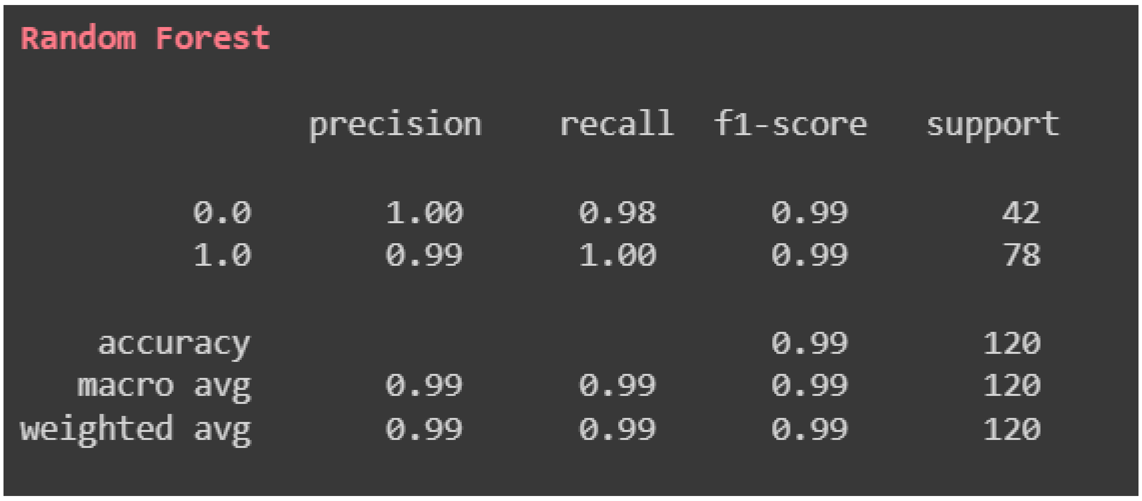
In the first experiment, we used the classifier without performing any feature selection and achieved an accuracy of 97.5%.



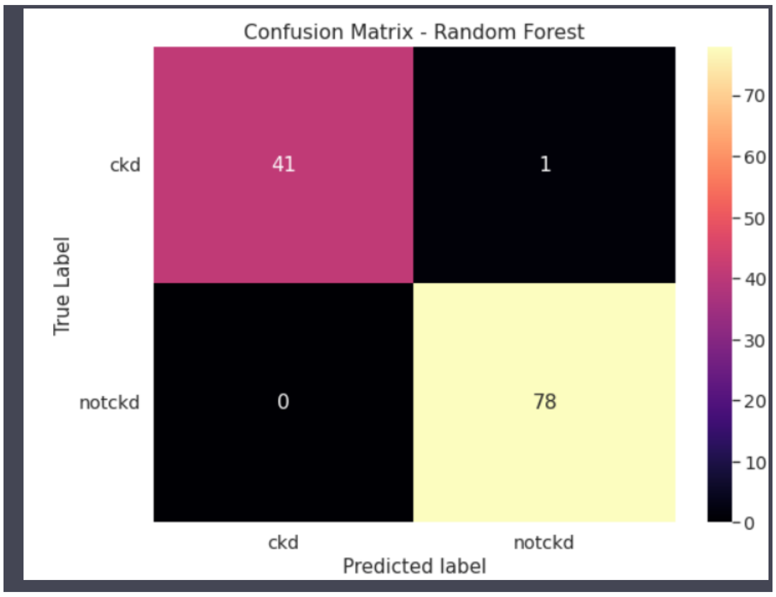
```
Random Forest

              precision    recall  f1-score   support

         0.0       0.98      0.95      0.96        42
         1.0       0.97      0.99      0.98        78

    accuracy                           0.97       120
   macro avg       0.98      0.97      0.97       120
weighted avg       0.98      0.97      0.97       120
```



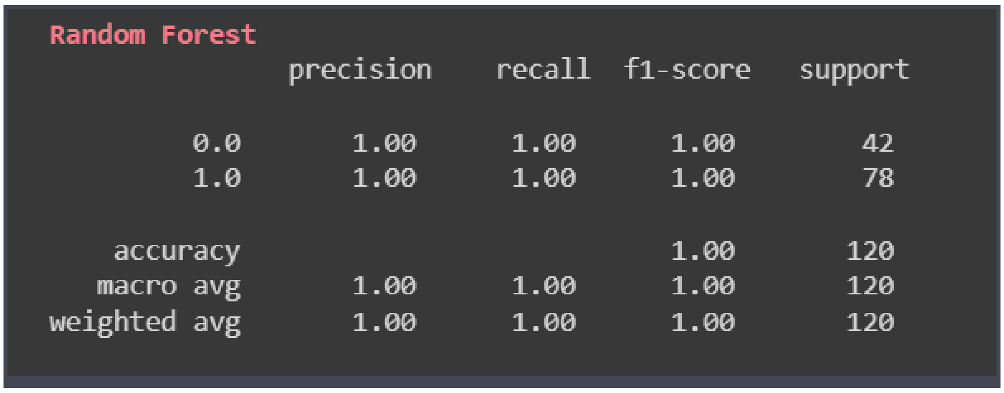
In the second experiment, weused recursive feature elimination with cross-validation (RFECV) to select the 9 most important features, which resulted in an accuracy of 99.17%.



```
Random Forest

              precision    recall  f1-score   support

         0.0       1.00      0.98      0.99        42
         1.0       0.99      1.00      0.99        78

    accuracy                           0.99       120
   macro avg       0.99      0.99      0.99       120
weighted avg       0.99      0.99      0.99       120
```



Finally, we boosted the classifier using the AdaBoost algorithm, which gave us a perfect accuracy of 100%.

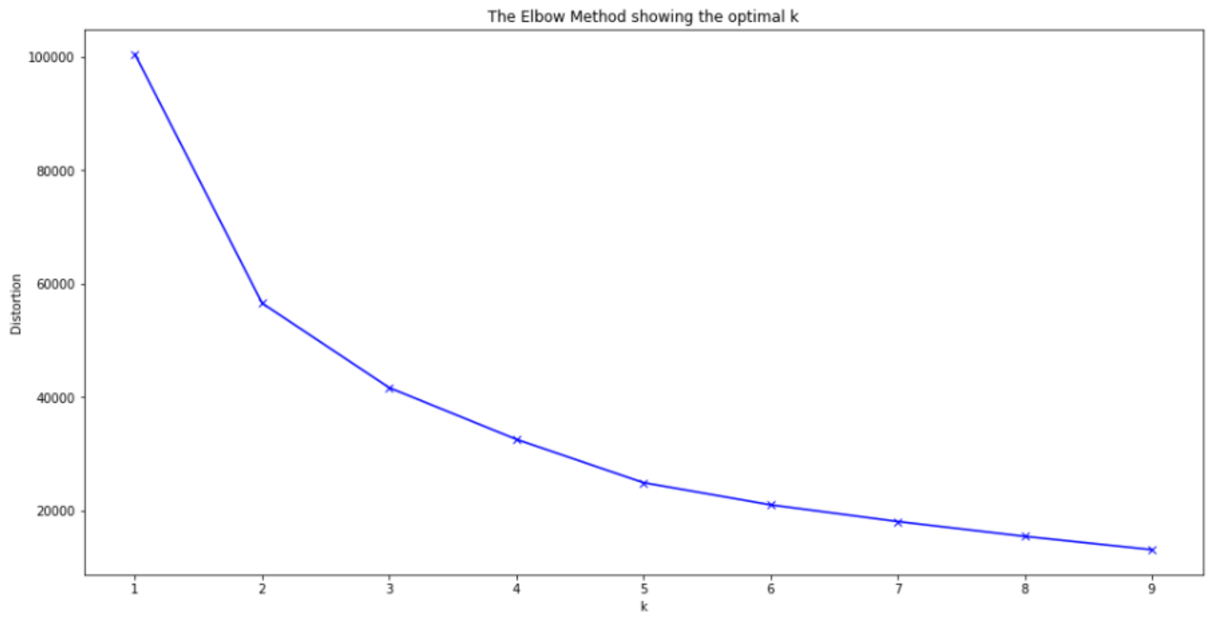


```
Random Forest
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        42
         1.0       1.00      1.00      1.00        78

    accuracy                           1.00       120
   macro avg       1.00      1.00      1.00       120
weighted avg       1.00      1.00      1.00       120
```

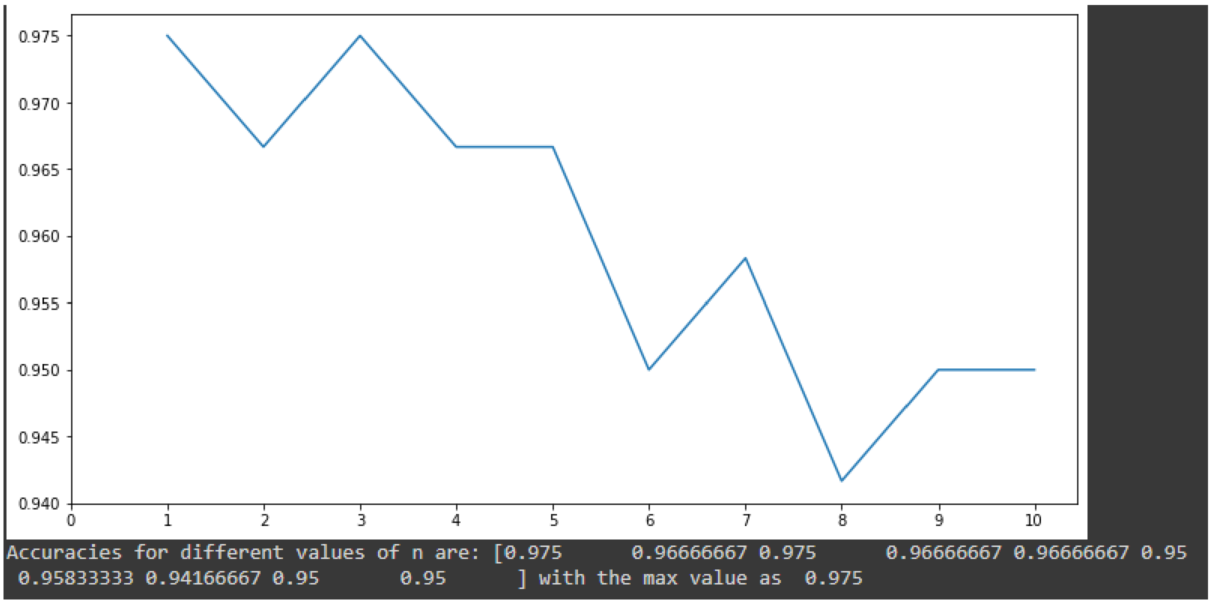
Overall, it seems like our experiments have been successful in improving the accuracy of the classifier. It's worth noting that achieving a perfect accuracy (using Adaboost) on our dataset is a strong indication that our model may be overfitting to the data. In other words, the model may be too closely tied to the specific details of your training data, and may not generalize well to new, unseen data. To avoid overfitting and improve the generalizability of our model, we could try using a larger, more diverse training dataset (Because RandomForest is a strong classifier and our dataset is too small so it easily overfits in case we boosted it).

- **KNN lab CFS :**

K-Nearest Neighbors (KNN) is a simple,easy-to-implement and effective classification algorithm that assigns a new sample to the class that is most common among its k nearest neighbors.we wanted to use this model with CFS feature selection and without .
The value of k is a hyperparameter that must be specified in advance.
One way to determine the value of k is to use the elbow method, which involves plotting the error rate of the KNN model against different values of k and choosing the value of k that results in the lowest error rate.



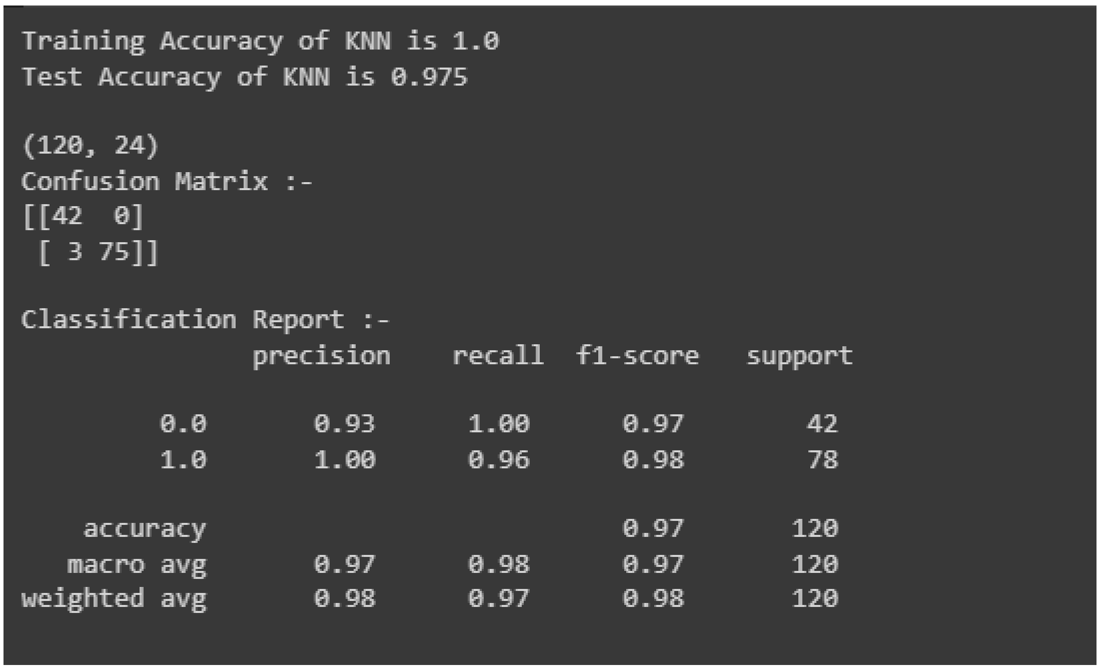
Another way to determine the value of k is to test the accuracy of the KNN model for different values of k. So we did test the model's accuracy for k values ranging from 1 to 10, and choose the value of k that results in the highest accuracy.



```
Accuracies for different values of n are: [0.975      0.96666667 0.975      0.96666667 0.96666667 0.95
 0.95833333 0.94166667 0.95       0.95      ] with the max value as  0.975
```

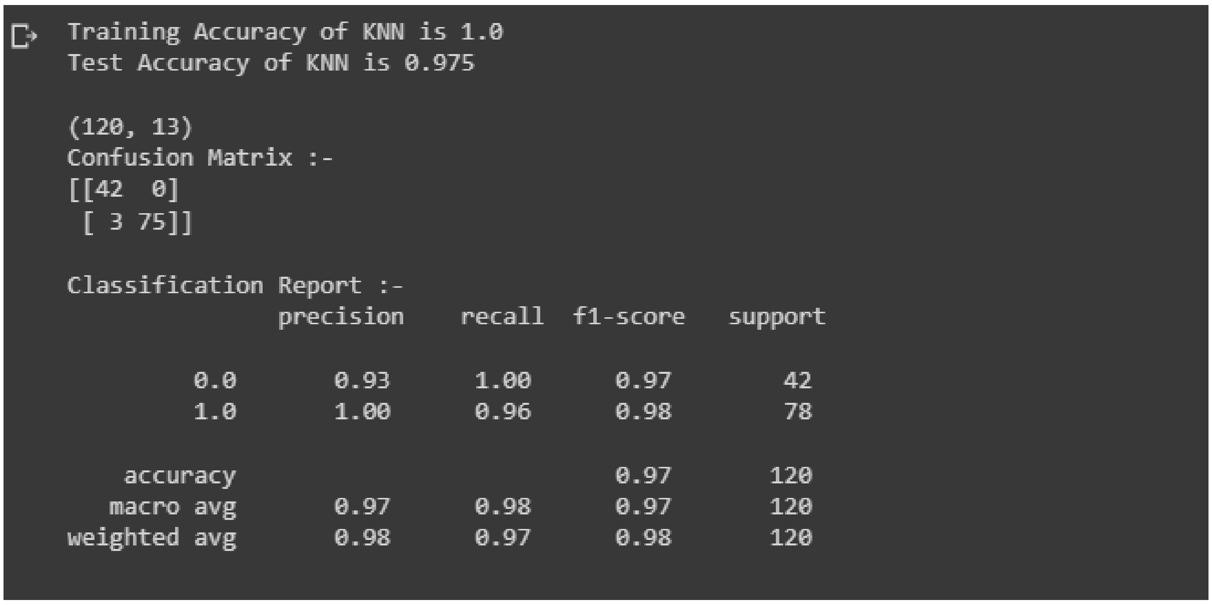
Through our analysis, we concluded that a value of **k=3** yielded the most desirable results for the two steps (with CFS and without).We evaluated the performance of the model using a classification report and a confusion matrix ;a classification report provided us information about the precision, recall, and f1-score of the model for each class and the confusion matrix will show the number of true positive, true negative, false positive, and false negative predictions made by the model.

without using CFS

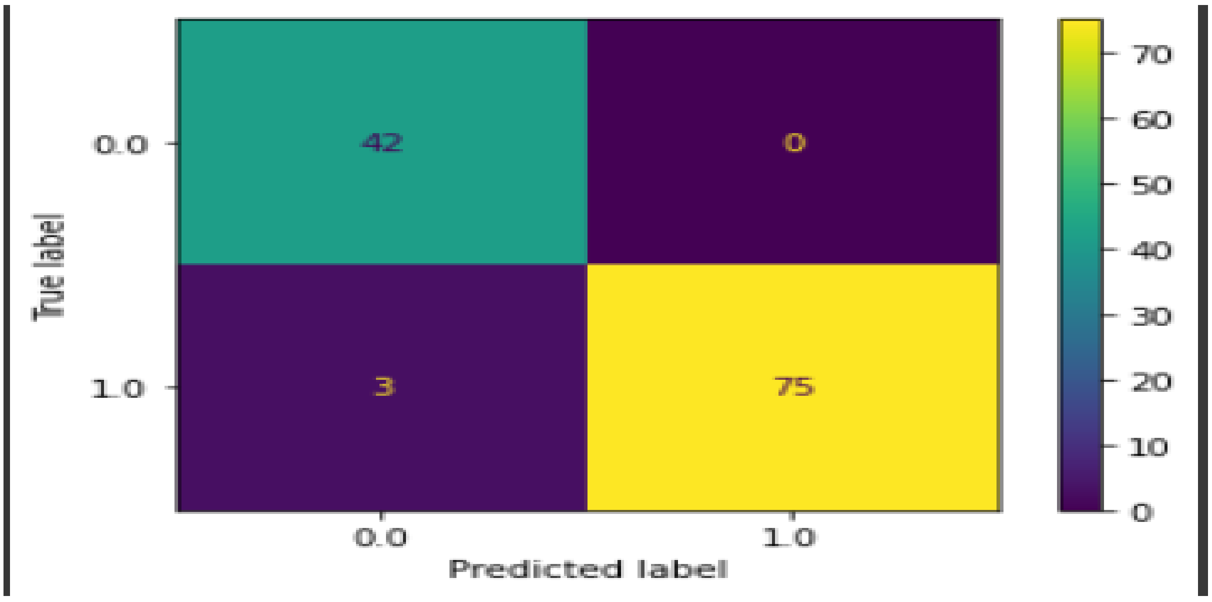


```
Training Accuracy of KNN is 1.0
Test Accuracy of KNN is 0.975

(120, 24)
Confusion Matrix :-
[[42  0]
 [ 3 75]]

Classification Report :-
              precision    recall  f1-score   support

         0.0       0.93      1.00      0.97        42
         1.0       1.00      0.96      0.98        78

    accuracy                           0.97       120
   macro avg       0.97      0.98      0.97       120
weighted avg       0.98      0.97      0.98       120
```

using CFS



```
Training Accuracy of KNN is 1.0
Test Accuracy of KNN is 0.975

(120, 13)
Confusion Matrix :-
[[42  0]
 [ 3 75]]

Classification Report :-
              precision    recall  f1-score   support

         0.0       0.93      1.00      0.97        42
         1.0       1.00      0.96      0.98        78

    accuracy                           0.97       120
   macro avg       0.97      0.98      0.97       120
weighted avg       0.98      0.97      0.98       120
```

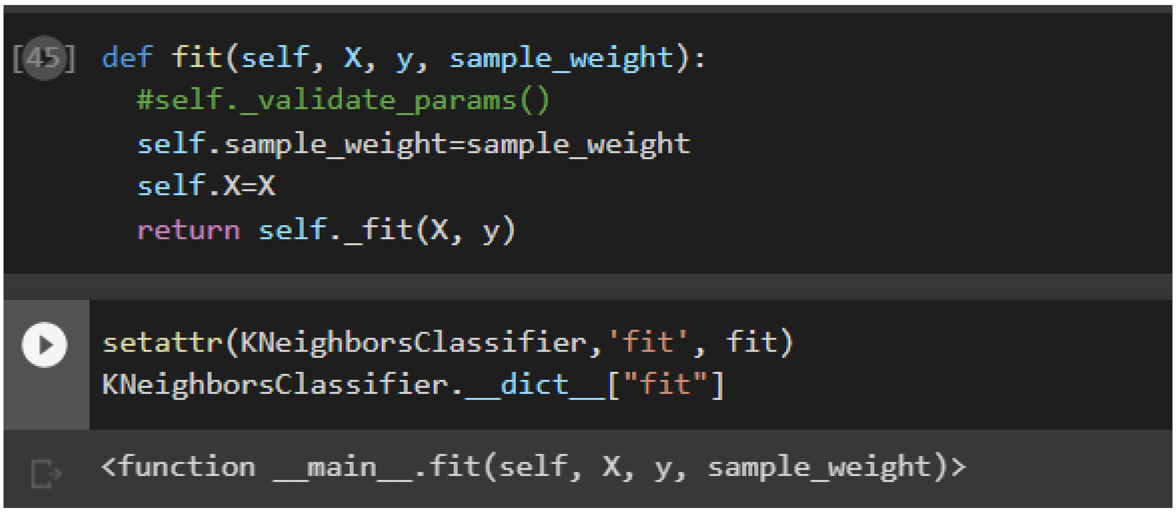


We correctly classified 42 samples as class 0 and 75 samples as class 1, but there were 3 times when the model misclassified samples as class 0. This indicates that the KNN model performed well in your classification task, **but there may be room for improvement.**
One way we could improve the accuracy of our KNN model is by using **Adaboost**.

- **KNN + ADABOOST :**

First we faced a problem to combine both models together ; KNN is a classification algorithm that does not support sample weighting. This means that when using KNN, all samples in the training dataset are given equal importance and the model cannot be adjusted to give more or less emphasis to certain samples.
Adaboost, on the other hand, is an ensemble learning algorithm that does support sample weighting. In Adaboost, each weak learner in the ensemble is trained on a different weighted version of the training dataset, where the weights are adjusted after each iteration to give more emphasis to samples that were misclassified by the previous weak learners.
The lack of support for sample weighting in KNN was a problem when using Adaboost to improve the performance of a KNN model. In order to address this issue,we needed to carefully consider the use of sample weighting in Adaboost and we choose an appropriate strategy for incorporating it into your analysis.
To improve the flexibility of the Sklearn KNN class, we created new versions of the predict, fit, and predict_proba functions that include a sample_weights parameter. This allows us to assign different weights to the samples in the training dataset, which can help the KNN model to better account for imbalanced classes or errors in the training data. We incorporated these modifications into our local version of Sklearn KNN, and used them to make predictions with the KNN model.

- ☐ **fit function :**

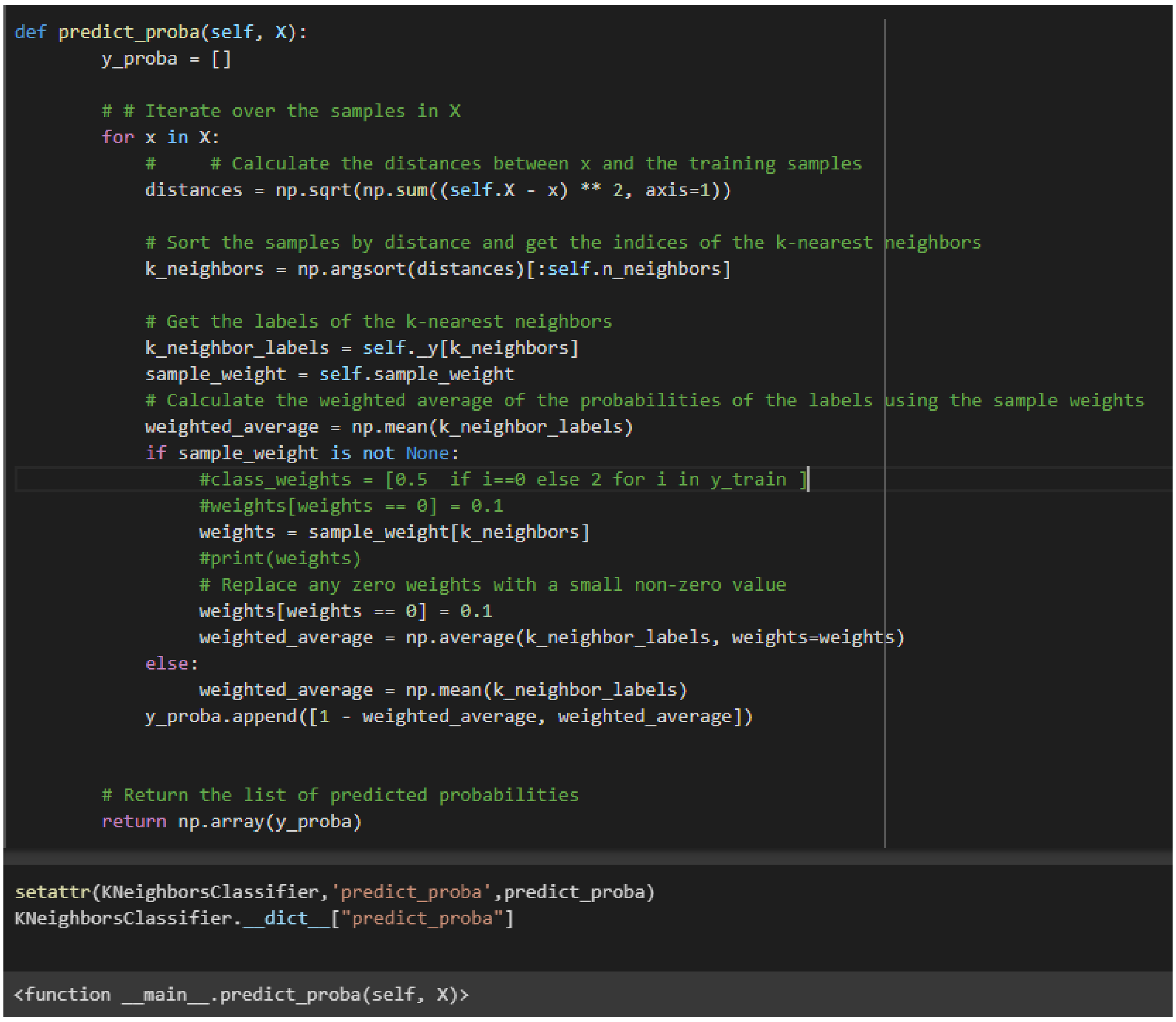


```
[45] def fit(self, X, y, sample_weight):
       #self._validate_params()
       self.sample_weight=sample_weight
       self.X=X
       return self._fit(X, y)
```

```
setattr(KNeighborsClassifier,'fit', fit)
KNeighborsClassifier.__dict__["fit"]
```

```
<function __main__.fit(self, X, y, sample_weight)>
```

☐ **Predict_proba function :**



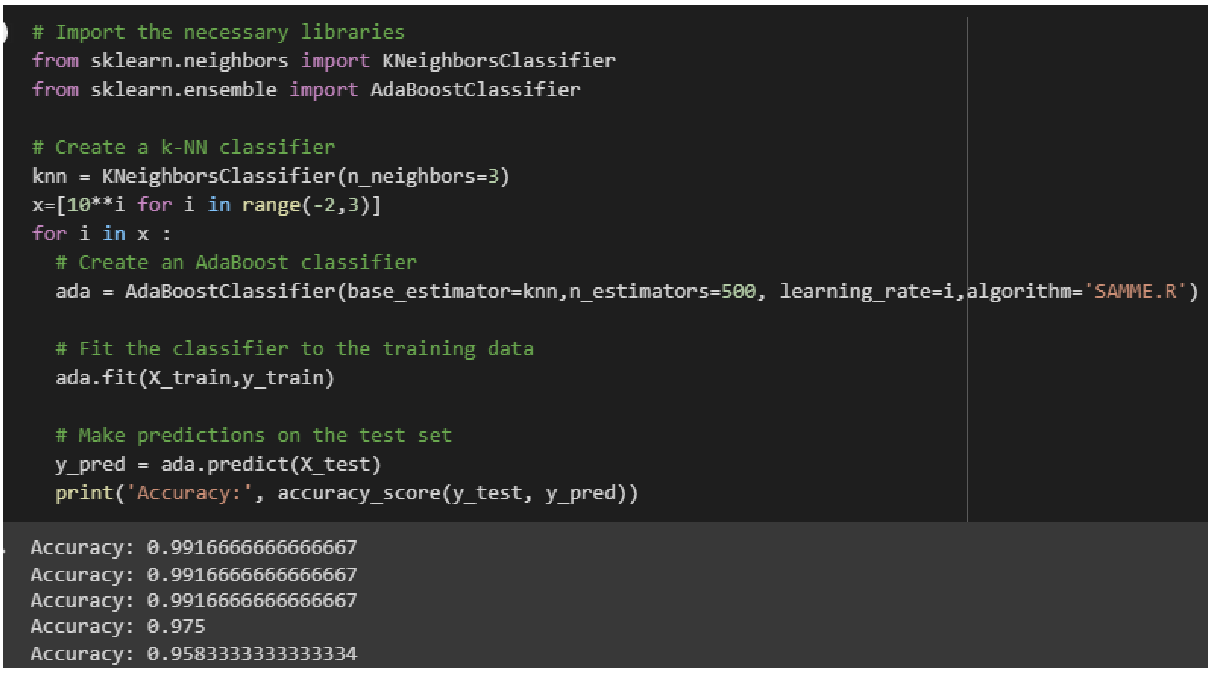
```
def predict_proba(self, X):
        y_proba = []

        # # Iterate over the samples in X
        for x in X:
            #     # Calculate the distances between x and the training samples
            distances = np.sqrt(np.sum((self.X - x) ** 2, axis=1))

            # Sort the samples by distance and get the indices of the k-nearest neighbors
            k_neighbors = np.argsort(distances)[:self.n_neighbors]

            # Get the labels of the k-nearest neighbors
            k_neighbor_labels = self._y[k_neighbors]
            sample_weight = self.sample_weight
            # Calculate the weighted average of the probabilities of the labels using the sample weights
            weighted_average = np.mean(k_neighbor_labels)
            if sample_weight is not None:
                #class_weights = [0.5  if i==0 else 2 for i in y_train ]
                #weights[weights == 0] = 0.1
                weights = sample_weight[k_neighbors]
                #print(weights)
                # Replace any zero weights with a small non-zero value
                weights[weights == 0] = 0.1
                weighted_average = np.average(k_neighbor_labels, weights=weights)
            else:
                weighted_average = np.mean(k_neighbor_labels)
            y_proba.append([1 - weighted_average, weighted_average])


        # Return the list of predicted probabilities
        return np.array(y_proba)
```

```
setattr(KNeighborsClassifier,'predict_proba',predict_proba)
KNeighborsClassifier.__dict__["predict_proba"]
```

```
<function __main__.predict_proba(self, X)>
```

Now we have solved the problem ; **we are finally able to use Knn as base estimator of adaboost .**

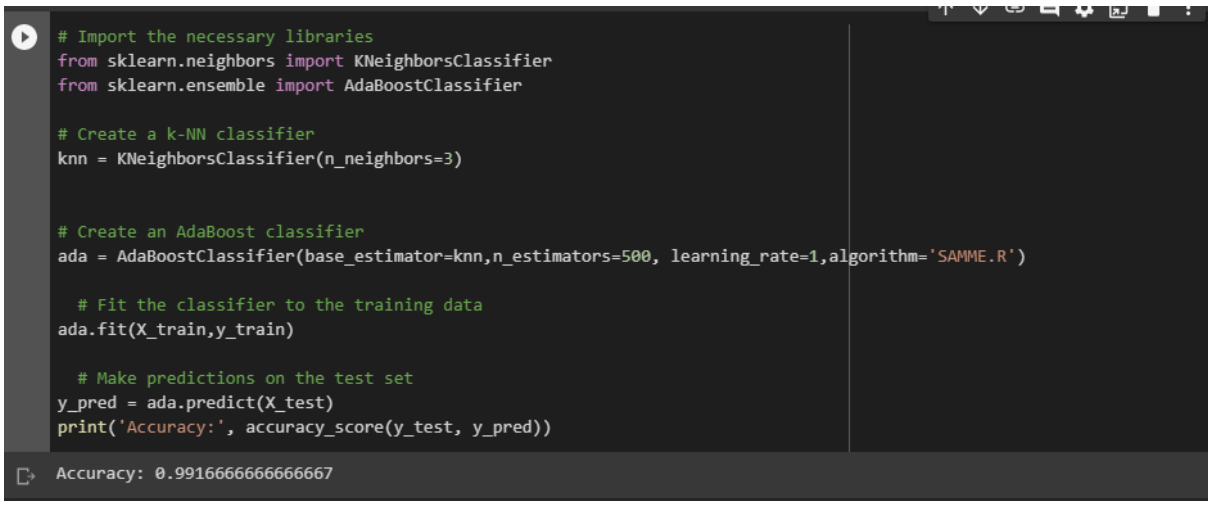
To test the accuracy now we need to specify two parameters : number of estimators and the learning rate since both affect the accuracy.



```
# Import the necessary libraries
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import AdaBoostClassifier

# Create a k-NN classifier
knn = KNeighborsClassifier(n_neighbors=3)
x=[10**i for i in range(-2,3)]
for i in x :
  # Create an AdaBoost classifier
  ada = AdaBoostClassifier(base_estimator=knn,n_estimators=500, learning_rate=i,algorithm='SAMME.R')

  # Fit the classifier to the training data
  ada.fit(X_train,y_train)

  # Make predictions on the test set
  y_pred = ada.predict(X_test)
  print('Accuracy:', accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9916666666666667
Accuracy: 0.9916666666666667
Accuracy: 0.9916666666666667
Accuracy: 0.975
Accuracy: 0.9583333333333334
```

so we chose the **learning rate equal 1** and **number of estimators equal to 500.**
the implementation of adaboost enhanced the accuracy and the confusion matrix as shown below :

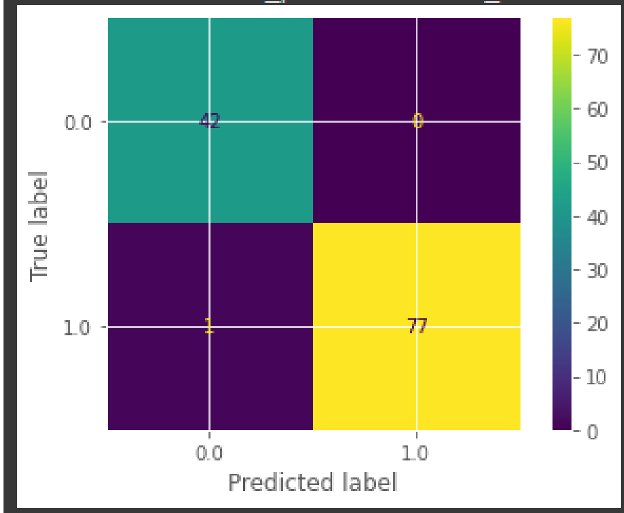


```
# Import the necessary libraries
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import AdaBoostClassifier

# Create a k-NN classifier
knn = KNeighborsClassifier(n_neighbors=3)


# Create an AdaBoost classifier
ada = AdaBoostClassifier(base_estimator=knn,n_estimators=500, learning_rate=1,algorithm='SAMME.R')

  # Fit the classifier to the training data
ada.fit(X_train,y_train)

  # Make predictions on the test set
y_pred = ada.predict(X_test)
print('Accuracy:', accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9916666666666667
```

```
plot_confusion_matrix(ada, X_test, y_test)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7efd8bbb9ac0>
```

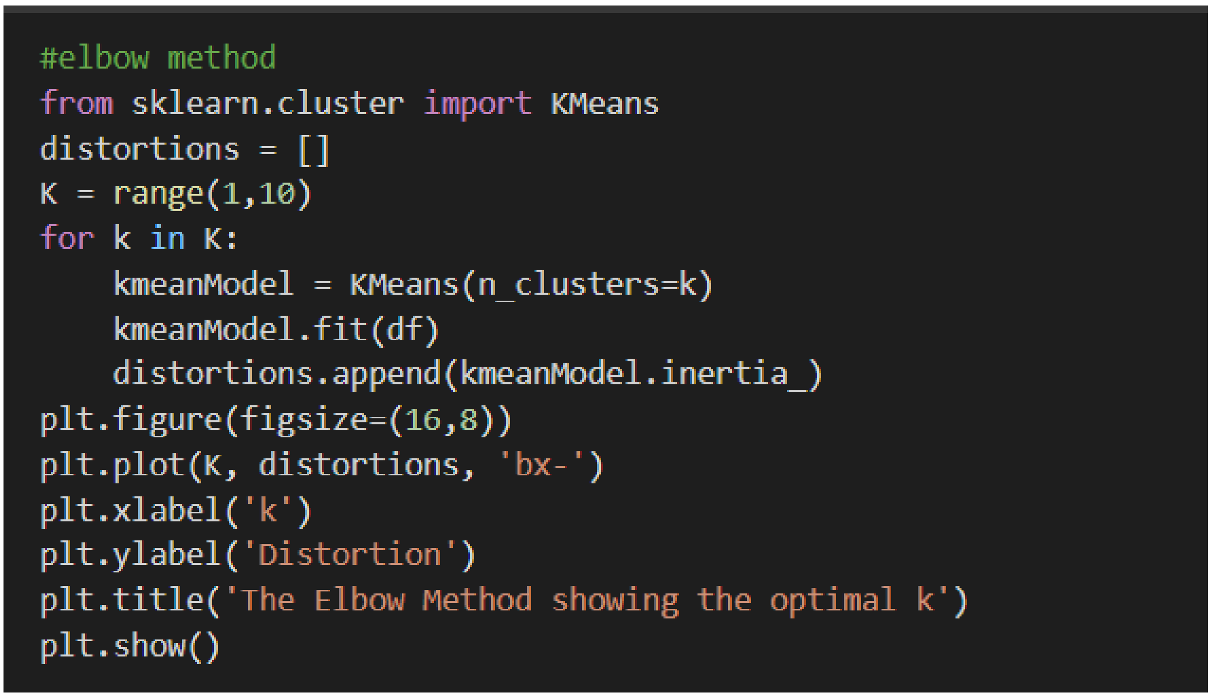


Conclusion :

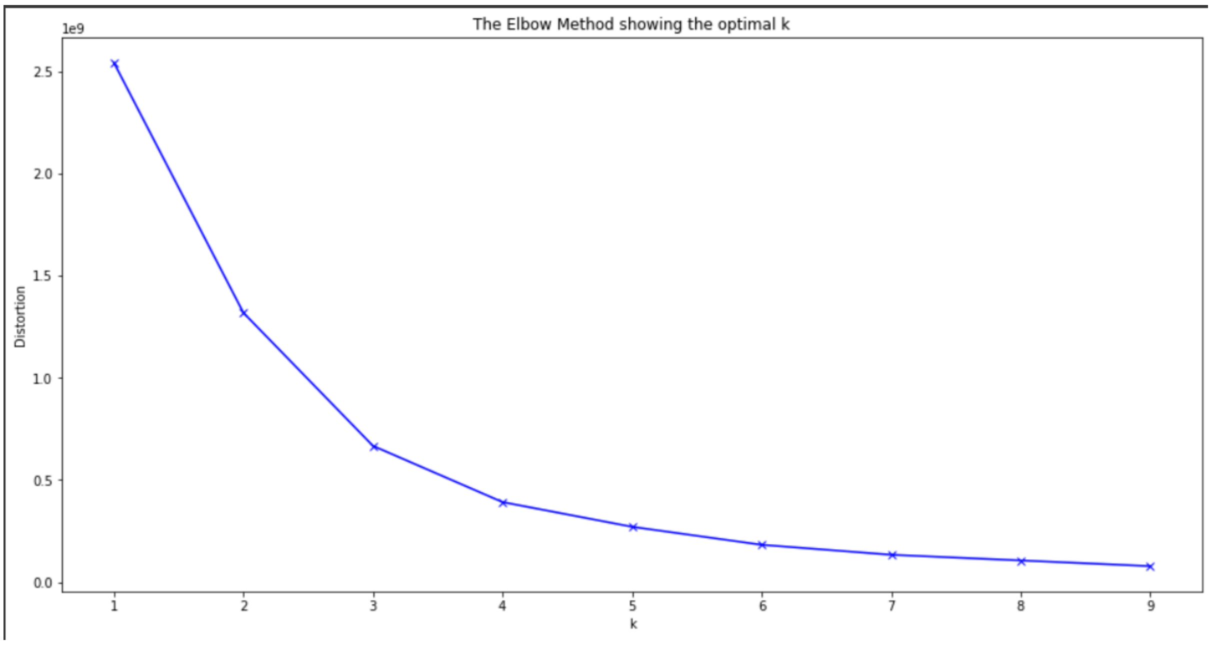
Method	Accuracy	Precision	Recall	F-measure
1st Method : Base	0.975	• 0.965	• 0.98	• 0.975
2nd Method : CFS	0.975	• 0.965	• 0.98	• 0.975
3rd Method : CFS + AdaBoost	0.9916666666666667	0.965	0.98	0.975

- **KNN lab RFE:**

To choose the appropriate k , we used the elbow method :

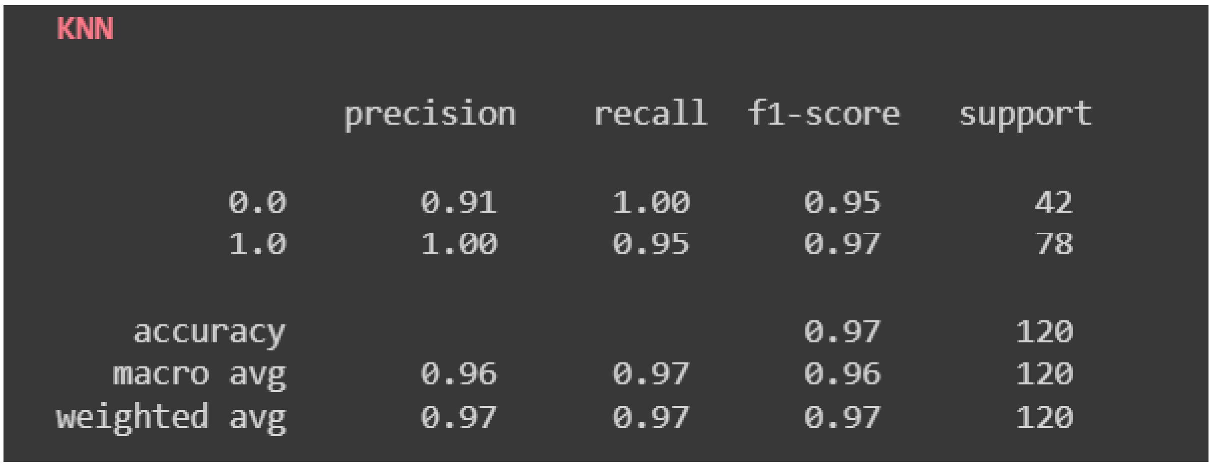


```
#elbow method
from sklearn.cluster import KMeans
distortions = []
K = range(1,10)
for k in K:
    kmeanModel = KMeans(n_clusters=k)
    kmeanModel.fit(df)
    distortions.append(kmeanModel.inertia_)
plt.figure(figsize=(16,8))
plt.plot(K, distortions, 'bx-')
plt.xlabel('k')
plt.ylabel('Distortion')
plt.title('The Elbow Method showing the optimal k')
plt.show()
```

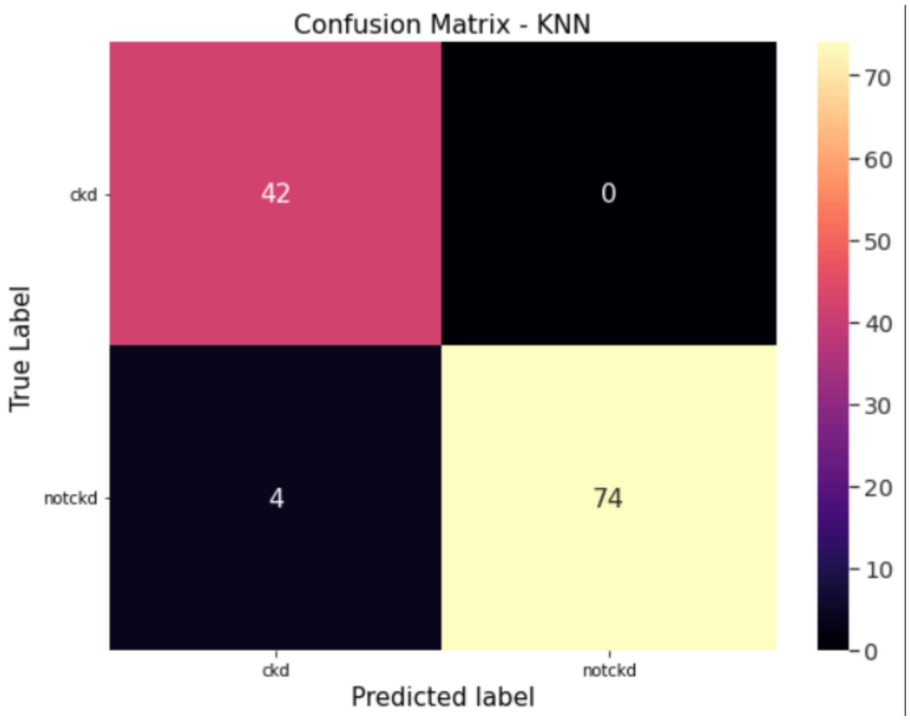


So , as shown in the plot **the best k is 3.**

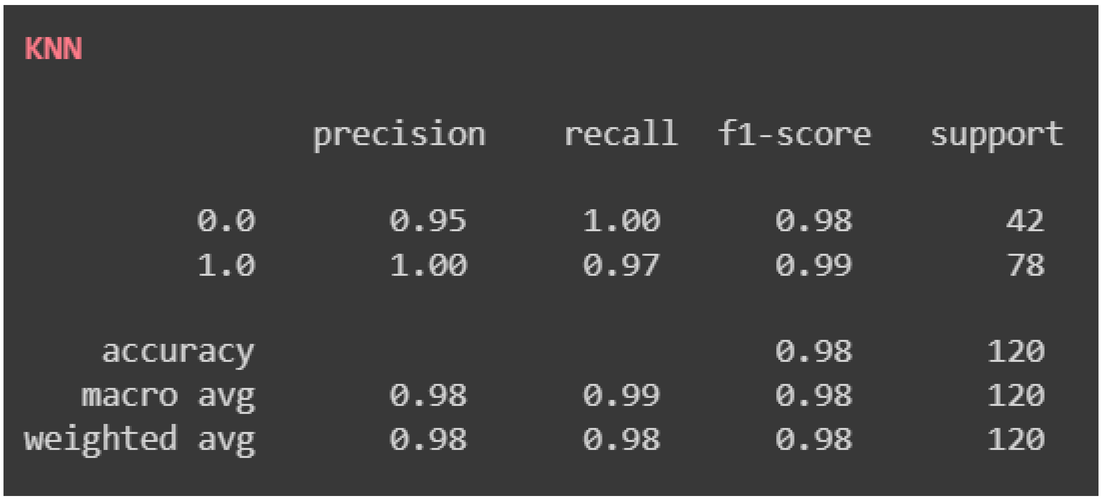
First, we tested KNN without RFE and we achieved accuracy 0.966666667



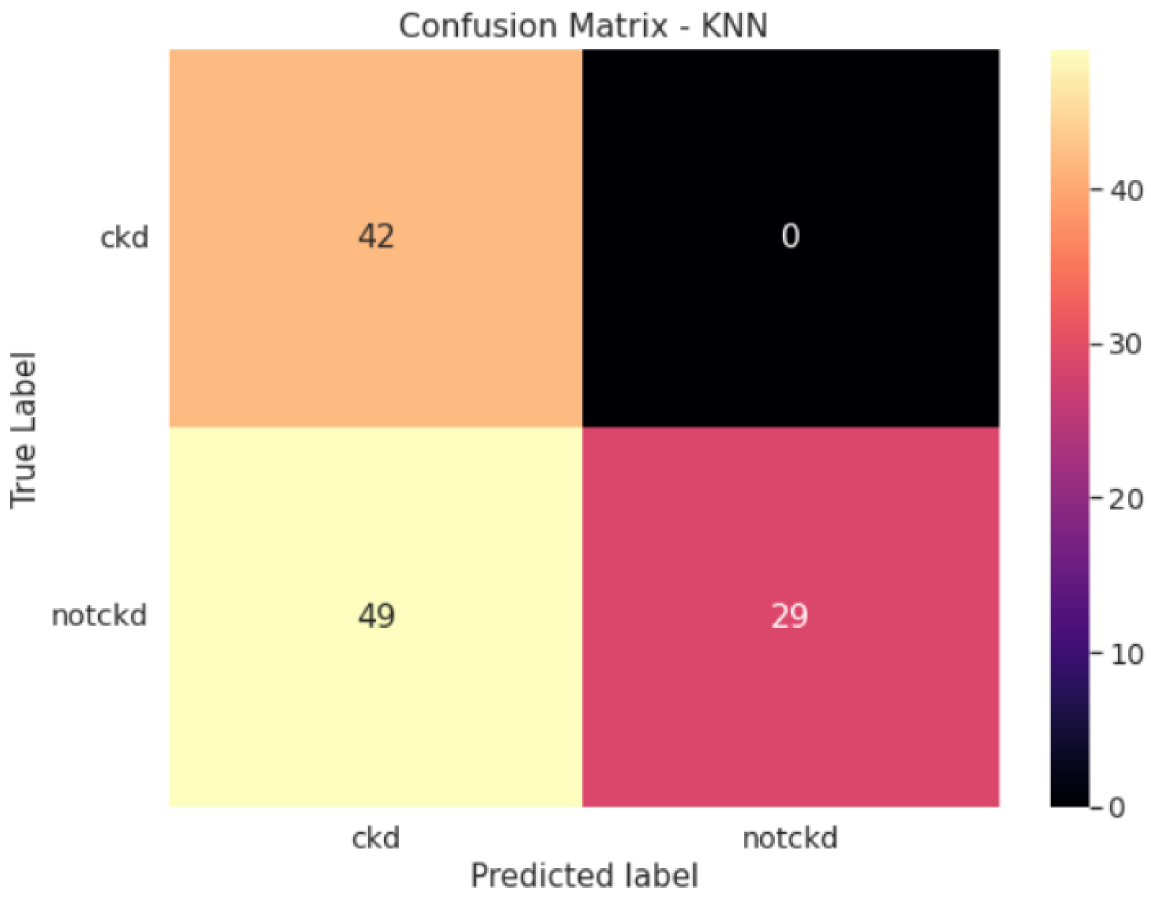
```
KNN

              precision    recall  f1-score   support

         0.0       0.91      1.00      0.95        42
         1.0       1.00      0.95      0.97        78

    accuracy                           0.97       120
   macro avg       0.96      0.97      0.96       120
weighted avg       0.97      0.97      0.97       120
```



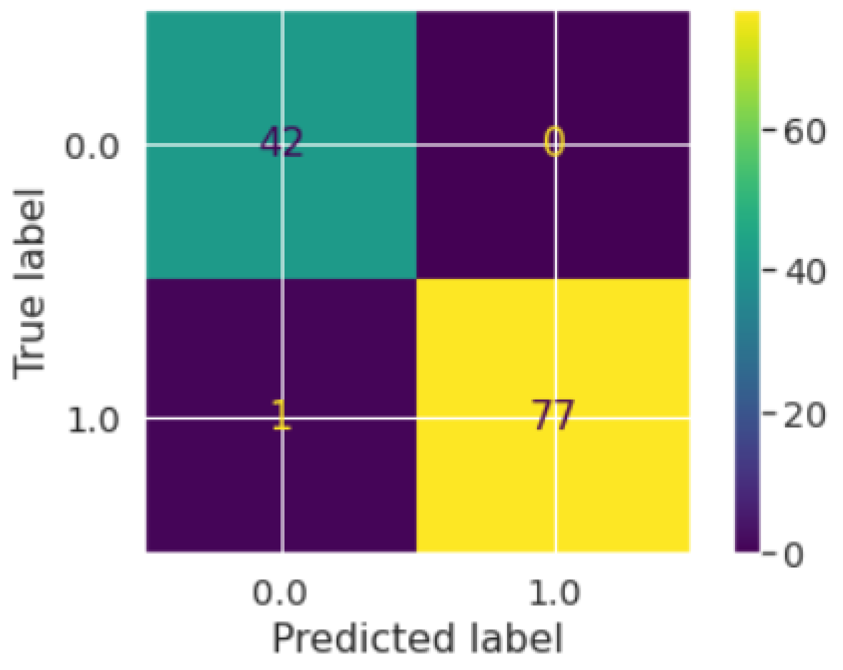
Second, we used recursive feature elimination with cross-validation (RFECV) to select the 9 most important features, which resulted in an accuracy of 0.9833333333.



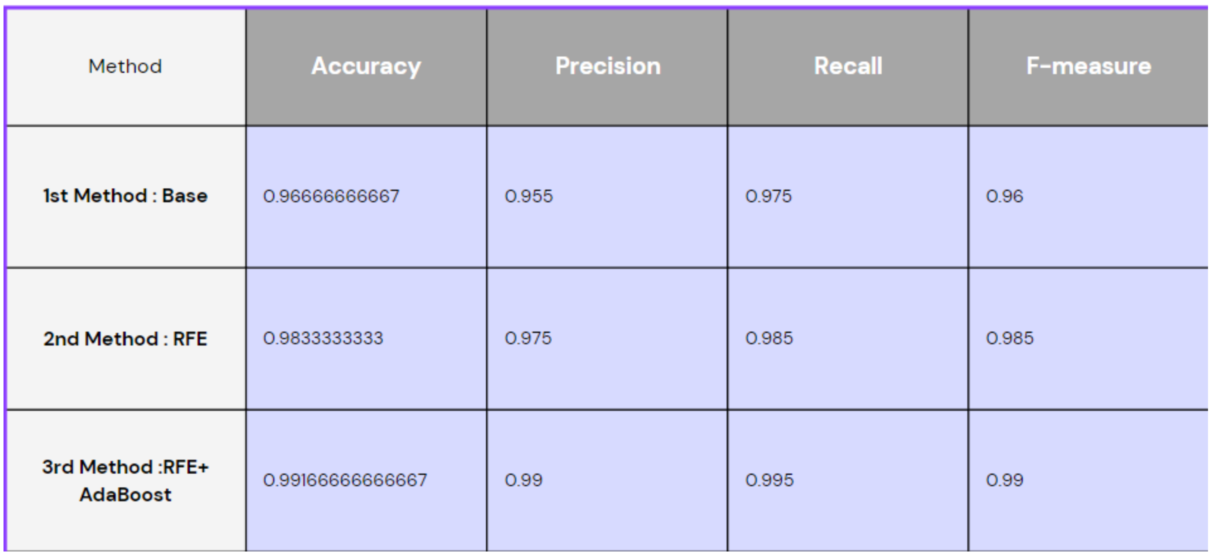
```
KNN

              precision    recall  f1-score   support

         0.0       0.95      1.00      0.98        42
         1.0       1.00      0.97      0.99        78

    accuracy                           0.98       120
   macro avg       0.98      0.99      0.98       120
weighted avg       0.98      0.98      0.98       120
```



Then, we tried to boost our model using the adaboost in which we achieved accuracy equal to 0.99166666666667 .So the adaboost improved our prediction as shown in the confusion matrix below :



CONCLUSION :

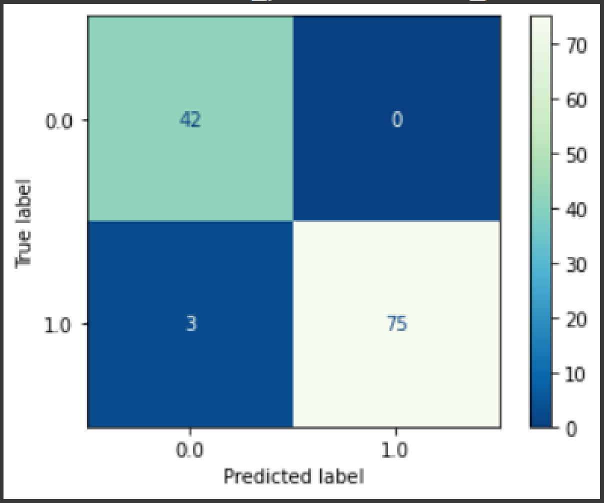


Method	Accuracy	Precision	Recall	F-measure
1st Method : Base	0.96666666667	0.955	0.975	0.96
2nd Method : RFE	0.9833333333	0.975	0.985	0.985
3rd Method :RFE+ AdaBoost	0.99166666666667	0.99	0.995	0.99

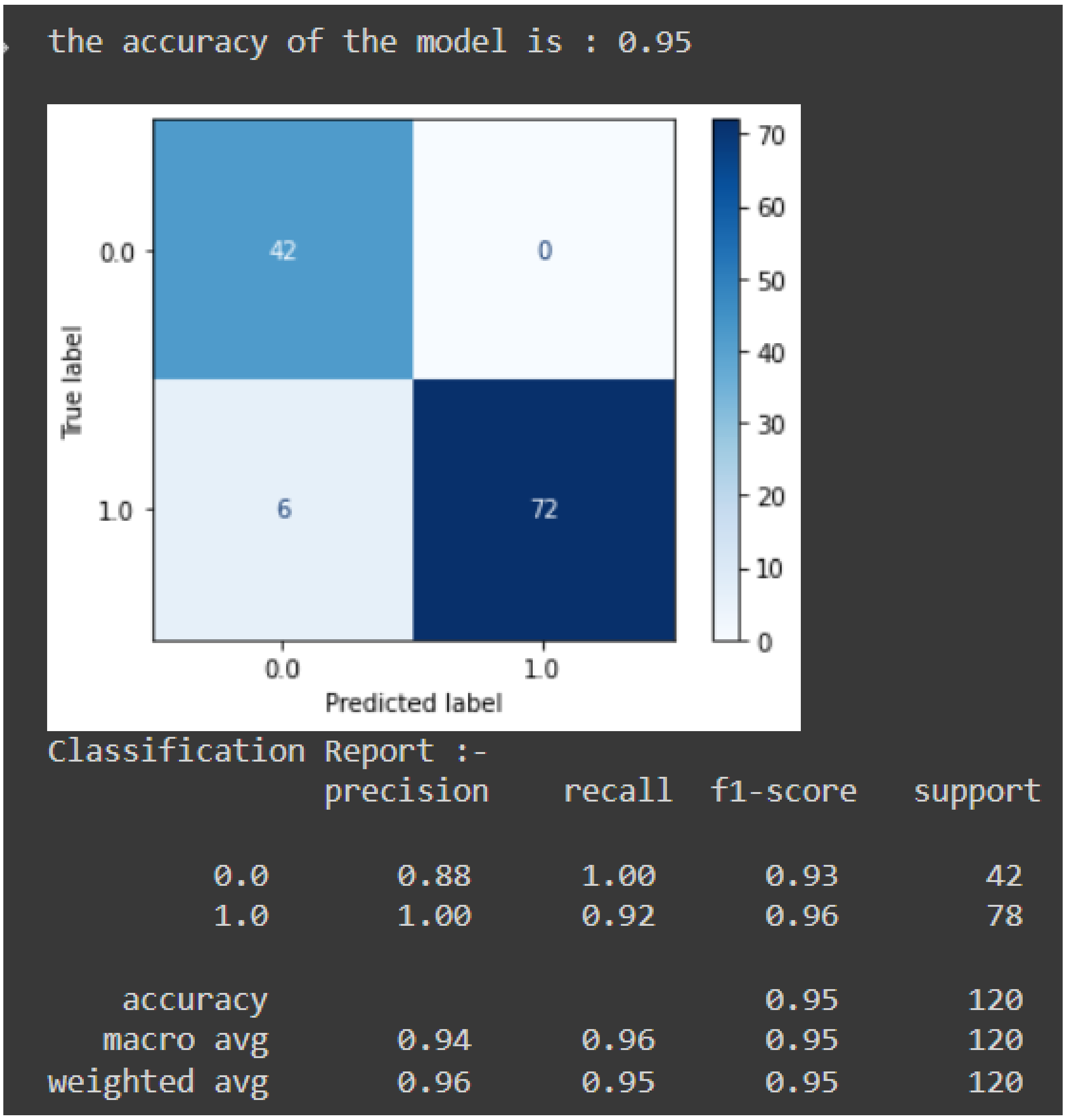
Naive Bayes :

A Naive Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem with strong (naive) independence assumptions. In simple terms, a Naive Bayes classifier assumes that the presence or absence of a particular feature of a class is unrelated to the presence or absence of any other feature.
We first applied the Naive Bayes classifier without performing any feature selection and we achieved accuracy equal to 0.975.

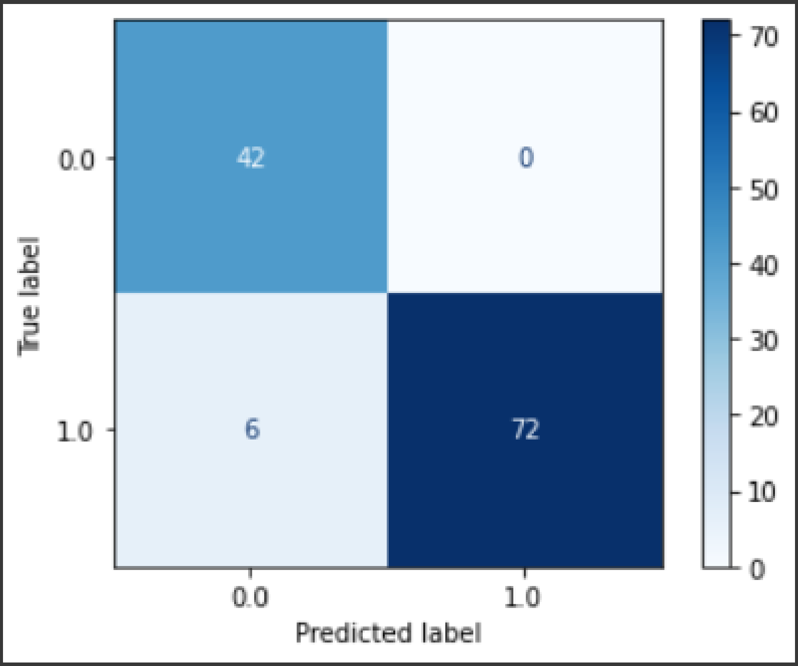
```
Accuracy on train: 1.0
Accuracy on test: 0.975
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f065b0c2370>
```



Next, we applied the Naive Bayes classifier again, this time using the Correlation-based Feature Selection (CFS) method to select the most relevant features and we achieved accuracy equal to 0.95.

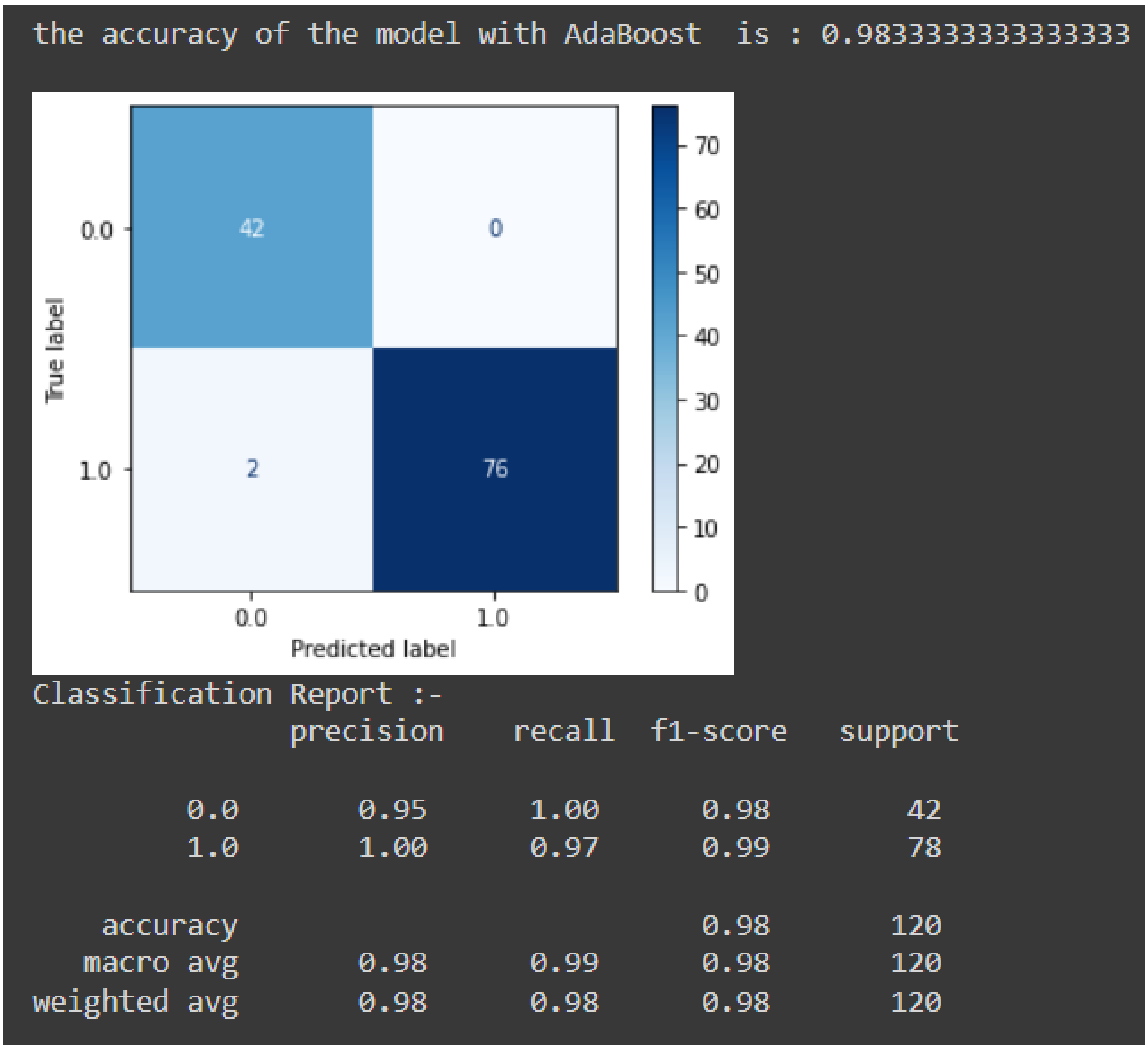


```
the accuracy of the model is : 0.95
```

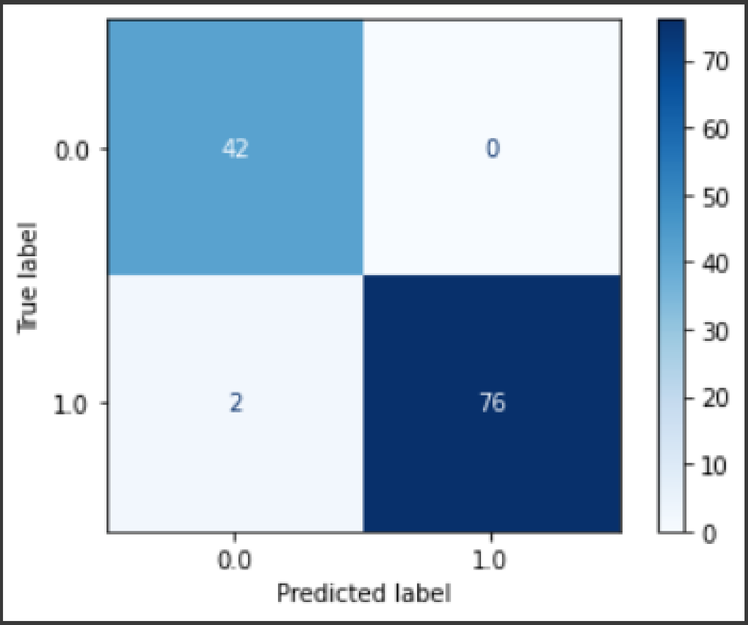


```
Classification Report :-
              precision    recall  f1-score   support

         0.0       0.88      1.00      0.93        42
         1.0       1.00      0.92      0.96        78

    accuracy                           0.95       120
   macro avg       0.94      0.96      0.95       120
weighted avg       0.96      0.95      0.95       120
```

With the addition of the CFS feature selection method, the accuracy of the Naive Bayes classifier improved and we achieved an accuracy equal to 0.983333.
In order to further improve the classification performance of the Naive Bayes classifier, we added the AdaBoost algorithm.



```
the accuracy of the model with AdaBoost  is : 0.9833333333333333
```



```
Classification Report :-
              precision    recall  f1-score   support

         0.0       0.95      1.00      0.98        42
         1.0       1.00      0.97      0.99        78

    accuracy                           0.98       120
   macro avg       0.98      0.99      0.98       120
weighted avg       0.98      0.98      0.98       120
```