IMPLEMENTATION OF MACHINE LEARNING MODELS FOR THE PREDICTION OF KIDNEY DISEASES (CKD)

YOSSER KADDOUR YOSRA KAROUI NOUR BEN AOUICHA
SARRA AYED ARIJ SOULA HELA JLASSI

SUPERVISOR : RAHMA FERJANI

Private Higher School of Engineering and Technology - ESPRIT

2022-2023

TABLE OF CONTENTS

ABSTRACT

This study has been conducted in order to generate an optimal and objective model that tracks the Chronic Kidney Disease (CKD) symptoms.

The methodology used to construct the model is the CRISP DM; which contains the following parts : Business understanding, Data Understanding, Data Preparation, Modeling, evaluation and Deployment.

In the next pages, we will be handling the previous steps in details.

I - INTRODUCTION

Chronic Kidney Disease (CKD) has been one of the main causes of death in recent years affecting over 10% of the population worldwide. Furthermore, unregulated diabetes and hypertension are global causes of CKD, and the prevalence of CKD is now determined by these two risk factors. Moreover, the symptoms of this disease are not specific, which is why researchers have been trying to find a more efficient way to track the disease. In terms of public health, the ability to predict CKD occurrence trends is critical so that decision-makers can take preemptive actions to prevent an increase in the number of patients.

Early detection of kidney impairment may help with rectification, which is not always attainable. To avert major damage, we must gain a better knowledge of a few kidney disease signs. The primary goal of this work is to predict renal illness by analyzing data from those indicators and using machine learning classification algorithms to do so.

II - METHODOLOGY

CRISP-DM, which stands for Cross-Industry Standard Process for Data Mining, is an industry-proven way to guide your data mining efforts. As a methodology, it includes descriptions of the typical phases of a project, the tasks involved with each phase, and an explanation of the relationships between these tasks [2].

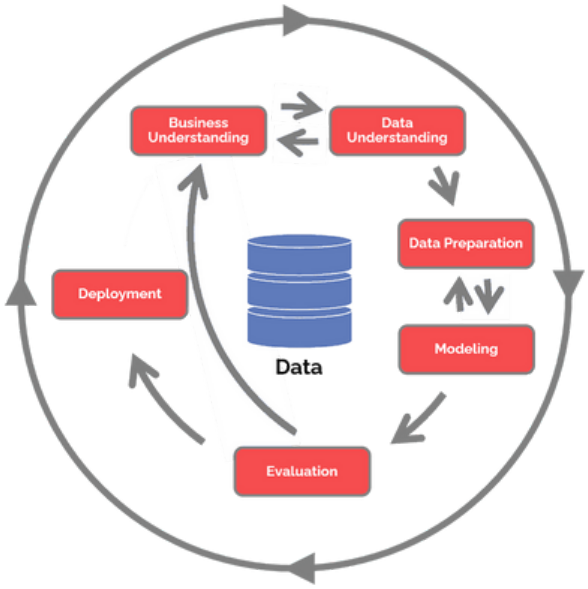

FIG 1 - CRISP-DM DIAGRAM

FIG 1 - CRISP-DM DIAGRAM

III - PHASE I. BUSINESS UNDERSTANDING

As shown in Figure 2, this phase is divided into four activities that will help in gaining a deeper understanding of the business.

FIG 2 - BUSINESS UNDERSTANDING TASKS

III - 1 - DETERMINATION OF BUSINESS OBJECTIVES

Chronic kidney disease (CKD) is a kind of disease in which kidney function gradually declines over months to years. Initially, there are no symptoms; however, subsequent symptoms may include leg swelling, fatigue, vomiting, lack of appetite, and disorientation. heart disease, high blood pressure, bone disease, and anemia are among the complications.
Chronic kidney disease (CKD) is due to the degeneration of the kidney's function and it is caused by two main reasons; diabetes and hypertension. Because there are no apparent symptoms in the early stages of CKD, individuals frequently do not recognize the disease, which eventually leads to full kidney function loss. Although this disease can be treated efficiently if the diagnosis is conducted appropriately and quickly. Thus, the diagnosis from medical personnel remains subjective which means that the machine learning method in this case is more reliable for the prediction.
The primary success factor for this project is to detect behaviors or behavior patterns in the early stages of CKD in order to improve the patient's quality of life. To obtain the most optimal result, the assessment metrics for the model (accuracy rate, recall rate, f-measure rate...) should be adequate for the method used for training.

III - 2 - ASSESSMENT OF THE SITUATION

The approach of this project is driven by the current situation of increasing conclusive identification of CKD, and lack of treatment or user ignorance of the disease, which leads to irreversible kidney failure in the final stages of CKD, such as dialysis for life, financially affecting the health system, as it is a costly treatment that generates the most significant amount of absorption of the world's resources available for health. This might be minimized by using approaches such as machine learning to detect CKD early on. Although the application of machine learning in healthcare and other sectors is useful, its full potential in the field of renal illness has yet to be realized.

III - 3 - THE OBJECTIVES OF DATA MINING

Through the study of these data and collecting the patients having CKD, a description and analysis are created using the original data, guaranteeing that they can be utilized or have the least information to conduct the categorization.
Using the obtained data, a training set is constructed. Several tests are performed to define or discover the optimum classifier technique(s) and to guarantee that the findings are realistic and efficient. To develop the prediction models, the stated classifier is used to train and assess them.
Predictive models frequently do calculations while ongoing transactions are in progress.

IV - PHASE II. DATA UNDERSTANDING

This section describes the data acquired, such as the number of records and fields per record and their identification, the significance of each field, and the description of the initial format, as shown in Figure 4.

FIG 3 - STUDY TASKS AND UNDERSTANDING OF THE DATA

IV - 1 - COLLECTION OF STARTING THE DATA

Dr.P.Soundarapandian provided the data set used in this project. M.D., D.M (Senior Consultant Nephrologist) from Apollo Hospitals in India, and its creator, L.Jerlin Rubini (Research Scholar), who was guided by Dr.P.Eswaran Assistant Professor, Department of Computer Science and Engineering at Alagappa University in India. They permitted and authorized the processing of this data. () samples are included in the dataset. Each sample in this data set includes 24 variables or predictive features (11 numerical variables and 14 categorical (nominal) variables) plus the class, for a total of 25 attributes.

IV - 2 - EXPLORE THE DATA

The data was imported into the Anaconda Jupyter platform and Google Colab environment to perform and visualize all of the data included in the dataset more coherently.

Initially, descriptive statistics on the variables that make up the data are performed. Initially, descriptive statistics on the variables that make up the data are performed. The key properties of the variables are identified in the table below. This result was obtained using Python commands.

Name	Description	Type: unit/ values
bp	blood pressure	numerical feature (in mm/Hg)
sg	specific gravity	nominal feature (1.005,1.010,1.015,1.020,1.025)
al	albumin	nominal feature (0,1,2,3,4,5)
su	sugar	nominal feature (0,1,2,3,4,5)
rbc	red blood cells	nominal feature (normal, abnormal)
pc	pus cell	nominal feature (normal, abnormal)
pcc	pus cell clumps	nominal feature (present, notpresent)
ba	bacteria	nominal feature (present, notpresent)
bgr	blood glucose random	numerical feature (in mgs/dl)
bu	blood urea	numerical feature (in mgs/dl)
sc	serum creatinine	numerical feature (in mgs/dl)
sod	sodium	numerical feature (in mEq/L)
pot	potassium	numerical feature (in mEq/L)
hemo	hemoglobin	numerical feature (in gms)
pcv	packed cell volume	numerical feature
wc	white blood cell count	numerical feature (cell/cumm)
rc	red blood cell count	numerical feature (millions/cmm)
htn	hypertension	nominal feature (yes,no)
dm	diabetes mellitus	nominal feature (yes,no)
cad	coronary artery disease	nominal feature (yes,no)
appet	appetite	nominal feature (good,poor)
pe	pedal edema	nominal feature (yes,no)
ane	anemia	nominal feature (yes,no)
class	class	nominal (ckd,notckd)

TABLE 1 - DESCRIPTION OF CKD DATASET

IV -3 -VERIFICATION OF THE QUALITY OF DATA

Data verification was performed in this portion to assess the consistency of the field values and the amount of distribution of the null values, as well as to identify out-of-range values that could cause noise in the process. This verification procedure was applied to the complete data set that was received. The number of fields where no records were found reached 1012.

The figure below shows summary statistics for each column, which can assist us to confirm our assertion about missing data.

	count	unique	top	freq
age	400	77	60	19
bp	400	11	80	116
sg	400	6	1.020	106
al	400	7	0	199
su	400	7	0	290
rbc	400	3	normal	201
pc	400	3	normal	259
pcc	400	3	notpresent	354
ba	400	3	notpresent	374
bgr	400	147	?	44
bu	400	119	?	19
sc	400	85	1.2	40
sod	400	35	?	87
pot	400	41	?	88
hemo	400	116	?	52
pcv	400	43	?	71
wbcc	400	90	?	106
rbcc	400	46	?	131
htn	400	3	no	251
dm	400	3	no	261
cad	400	3	no	364
appet	400	3	good	317
pe	400	3	no	323
ane	400	3	no	339
class	400	2	ckd	250

FIGURE 4 - SUMMARY STATISTICS

```
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
wbcc    106
htn       2
dm        2
cad       2
appet     1
pe        1
ane       1
class     0
dtype: int64
```

FIGURE 5 - NUMBER OF MISSING VALUES IN EACH FEATURE

Before beginning to analyze the data set, a series of visualizations are created to aid in understanding the features of the information being dealt with and their correlation.

To gain a better understanding of the data, we started by examining the distribution and relationship between variables, as shown in the figure below. We see distinct classifications of positive and negative results in each targeted health attribute.

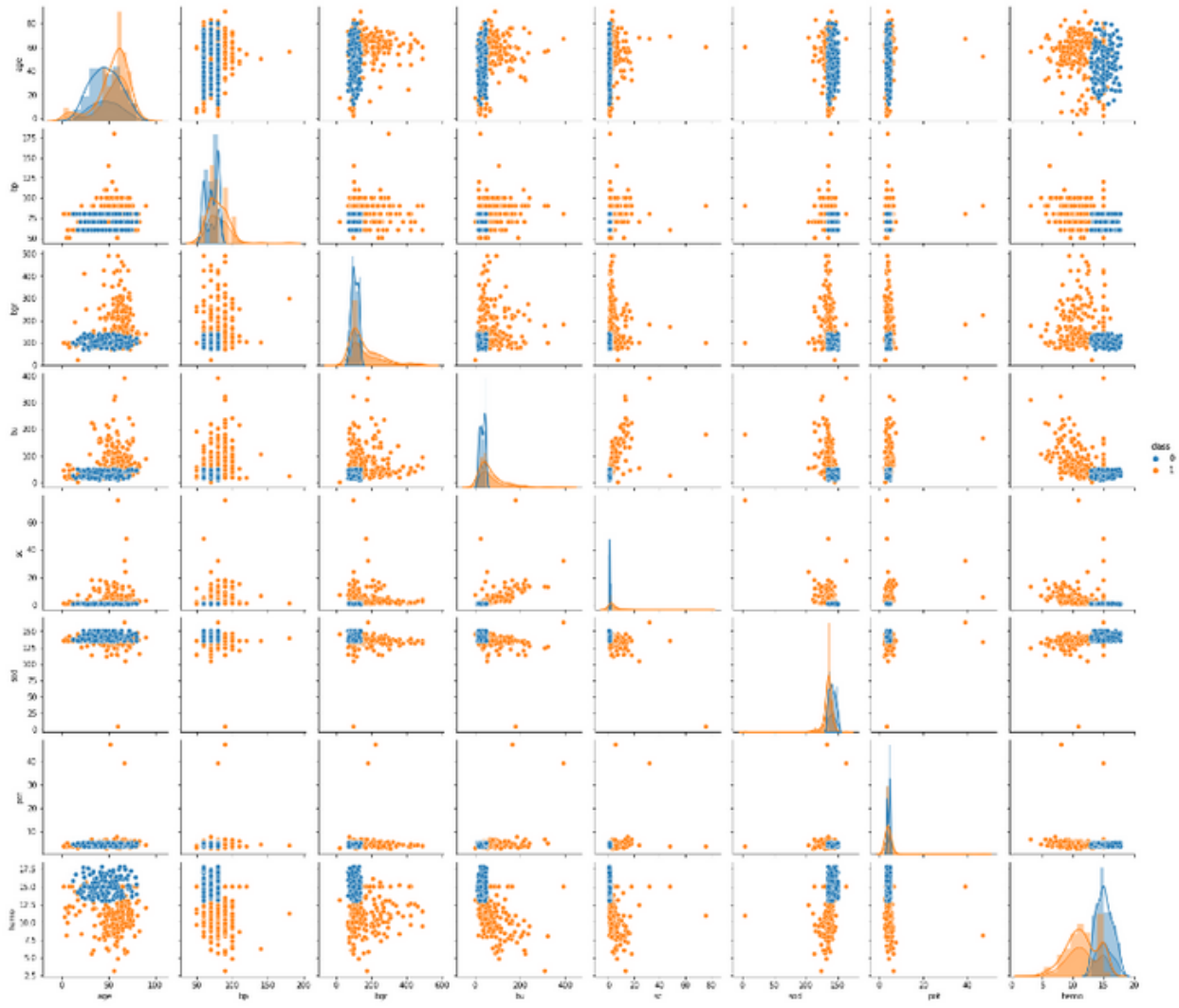

FIGURE 10 - THE DISTRIBUTION AND RELATIONSHIP BETWEEN VARIABLES

As can be seen in the image, we are dealing with an unbalanced dataset, with 150 people not suffering from CKD and 250 people suffering from CKD.

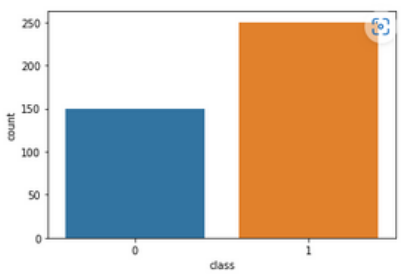

FIGURE 11 - CKD And Not CKD

The figures below describe the relationship between age and whether or not a patient has chronic kidney disease.

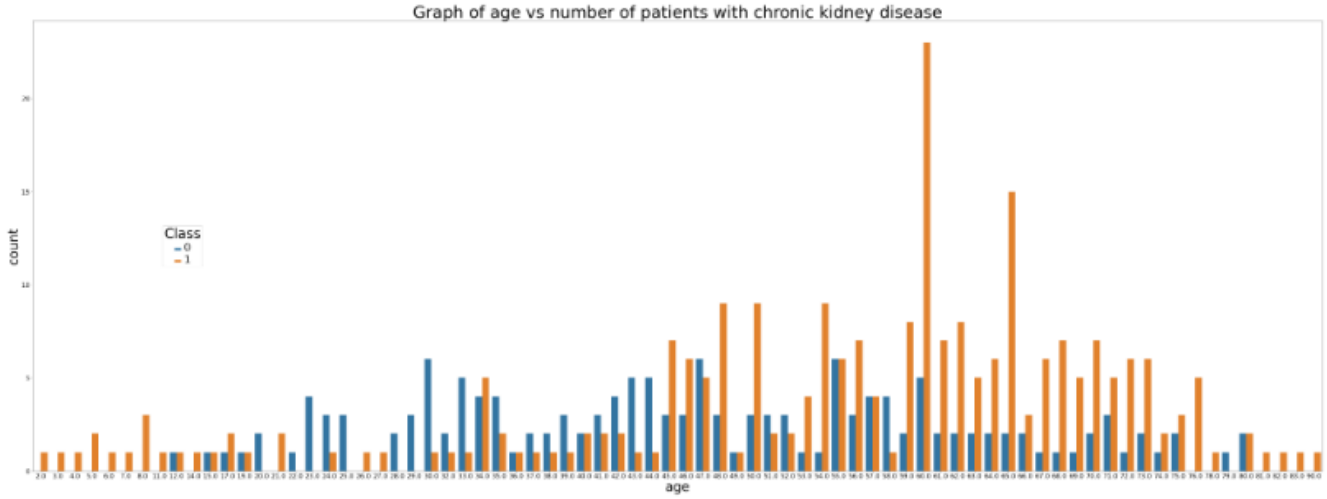

FIGURE 12 - AGE AND NUMBER OF PATIENTS WITH CKD

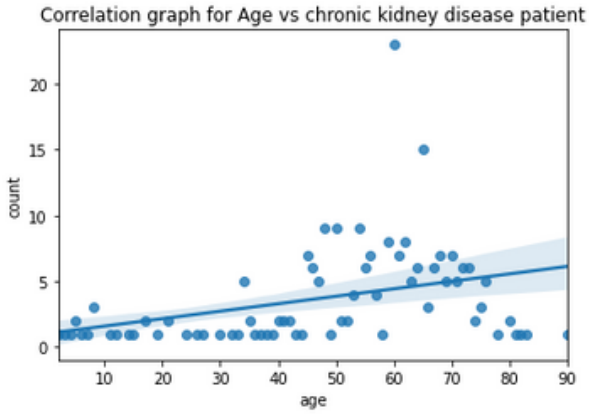

FIGURE 13 - CORRELATION GRAPH FOR AGE VS CKD PATIENT

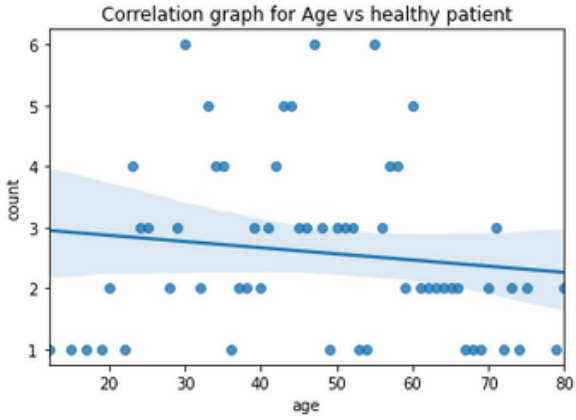

FIGURE 14 - CORRELATION GRAPH FOR AGE VS HEALTHY PATIENT

We have around 150 individuals and 250 chronic kidney disease patients in this dataset. The National Kidney Foundation has related aging with kidney illness, claiming that "more than 50 percent of seniors over the age of 75 are believed to have kidney disease. Kidney disease has also been discovered to be more prevalent among persons over the age of 60 when compared to the rest of the general population." Although age is one element that might contribute to chronic kidney disease, a poor diet/lifestyle is just as important.

The graphs above will assist us to get an overview of the problem.

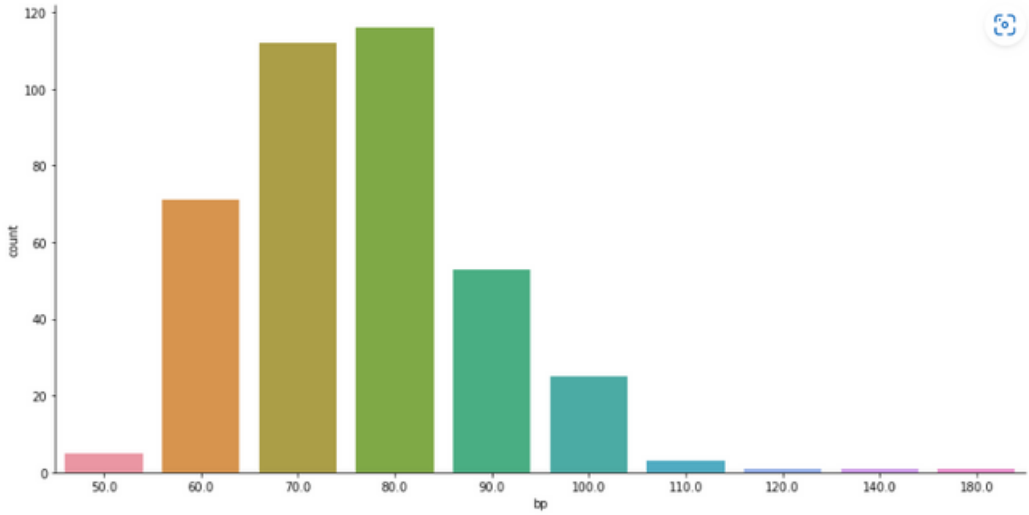

FIGURE 15 - BLOOD PRESSURE-FREQUENCY GRAPH

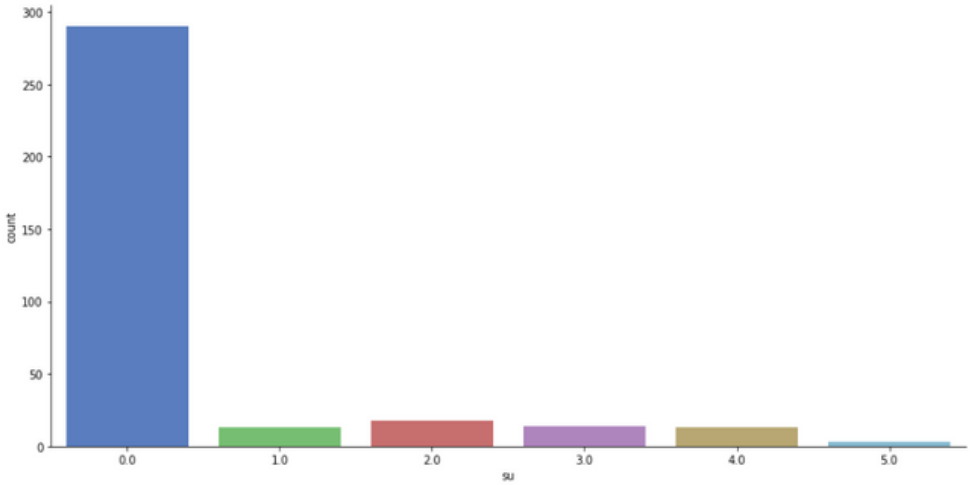

FIGURE 16 - SUGAR-FREQUENCY GRAPH

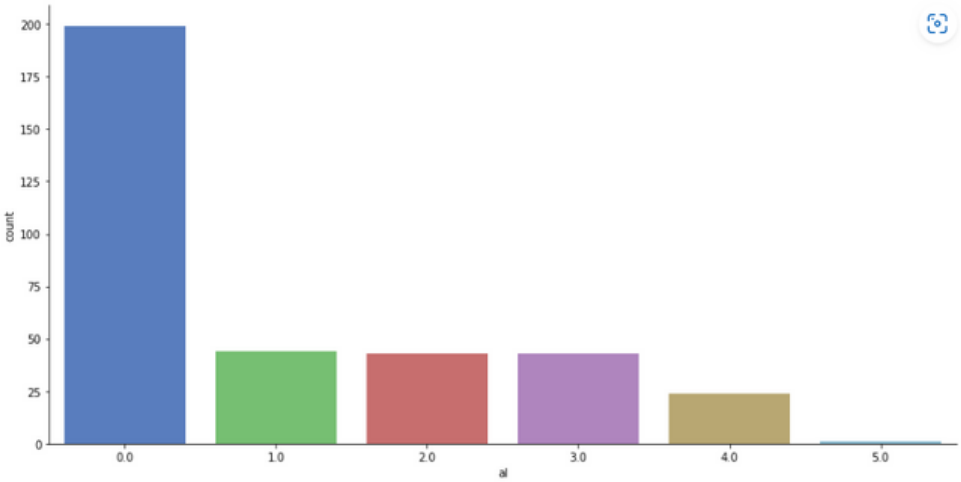

FIGURE 17 - ALBUMIN-FREQUENCY GRAPH

V- PHASE III. DATA PREPARATION

V - 1 - CONVERTING CATEGORICAL FEATURES TO NUMERICAL FEATURES

In this step, we will transform nominal data into numerical one in order to apply the feature selection algorithms on all the features.

```
[13] df['class'] = df['class'].map({'ckd':1,'notckd':0})
     df['htn'] = df['htn'].map({'yes':1,'no':0})
     df['dm'] = df['dm'].map({'yes':1,'no':0})
     df['cad'] = df['cad'].map({'yes':1,'no':0})
     df['appet'] = df['appet'].map({'good':1,'poor':0})
     df['ane'] = df['ane'].map({'yes':1,'no':0})
     df['pe'] = df['pe'].map({'yes':1,'no':0})
     df['ba'] = df['ba'].map({'present':1,'notpresent':0})
     df['pcc'] = df['pcc'].map({'present':1,'notpresent':0})
     df['pc'] = df['pc'].map({'abnormal':1,'normal':0})
     df['rbc'] = df['rbc'].map({'abnormal':1,'normal':0})
```

FIGURE 18 - DATA TRANSFORMATION CODE FROM CATEGORICAL FEATURES TO NUMERICAL FEATURES

As seen in the picture below, the new dataset after converting the categorical features to numerical ones this will allows us to deal with the data more easily.

```
[14] df
```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wbcc	rbcc	htn	dm	cad	appet	pe	ane	class
0	48.0	80.0	1.020	1.0	0.0	NaN	0.0	0.0	0.0	121.0	...	44.0	7800.0	5.2	1.0	1.0	0.0	1.0	0.0	0.0	1
1	7.0	50.0	1.020	4.0	0.0	NaN	0.0	0.0	0.0	NaN	...	38.0	6000.0	NaN	0.0	0.0	0.0	1.0	0.0	0.0	1
2	62.0	80.0	1.010	2.0	3.0	0.0	0.0	0.0	0.0	423.0	...	31.0	7500.0	NaN	0.0	1.0	0.0	0.0	0.0	1.0	1
3	48.0	70.0	1.005	4.0	0.0	0.0	1.0	1.0	0.0	117.0	...	32.0	6700.0	3.9	1.0	0.0	0.0	0.0	1.0	1.0	1
4	51.0	80.0	1.010	2.0	0.0	0.0	0.0	0.0	0.0	106.0	...	35.0	7300.0	4.6	0.0	0.0	0.0	1.0	0.0	0.0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
395	55.0	80.0	1.020	0.0	0.0	0.0	0.0	0.0	0.0	140.0	...	47.0	6700.0	4.9	0.0	0.0	0.0	1.0	0.0	0.0	0
396	42.0	70.0	1.025	0.0	0.0	0.0	0.0	0.0	0.0	75.0	...	54.0	7800.0	6.2	0.0	0.0	0.0	1.0	0.0	0.0	0
397	12.0	80.0	1.020	0.0	0.0	0.0	0.0	0.0	0.0	100.0	...	49.0	6600.0	5.4	0.0	0.0	0.0	1.0	0.0	0.0	0
398	17.0	60.0	1.025	0.0	0.0	0.0	0.0	0.0	0.0	114.0	...	51.0	7200.0	5.9	0.0	0.0	0.0	1.0	0.0	0.0	0
399	58.0	80.0	1.025	0.0	0.0	0.0	0.0	0.0	0.0	131.0	...	53.0	6800.0	6.1	0.0	0.0	0.0	1.0	0.0	0.0	0

400 rows × 25 columns

FIGURE 19 - DATASET AFTER TRANSFORMATION

V - 2 - DEALING WITH MISSING VALUES

We begin by replacing the numerical features' missing values. We used statistical measures, specifically the mean method. To replace missing nominal features, the mode method was used.
The code we used is shown in the figure below.

```
df['age'] = df['age'].fillna(df['age'].mean)
df['bp'] = df['bp'].fillna(df['bp'].mean)
df['bgr'] = df['bgr'].fillna(df['bgr'].median)
df['bu'] = df['bu'].fillna(df['bu'].mean)
df['sc'] = df['sc'].fillna(df['sc'].median)
df['sod'] = df['sod'].fillna(df['sod'].median)
df['pot'] = df['pot'].fillna(df['pot'].median)
df['hemo'] = df['hemo'].fillna(df['hemo'].median)
df['wbcc'] = df['wbcc'].fillna(df['wbcc'].median)
```

```
df['sg'] = df['sg'].fillna(df['sg'].mode()[0])
df['al'] = df['al'].fillna(df['al'].mode()[0])
df['su'] = df['su'].fillna(df['su'].mode()[0])
df['rbc'] = df['rbc'].fillna(df['rbc'].mode()[0])
df['pc'] = df['pc'].fillna(df['pc'].mode()[0])
df['pcc'] = df['pcc'].fillna(df['pcc'].mode()[0])
df['ba'] = df['ba'].fillna(df['ba'].mode()[0])
df['htn'] = df['htn'].fillna(df['htn'].mode()[0])
df['dm'] = df['dm'].fillna(df['dm'].mode()[0])
df['cad'] = df['cad'].fillna(df['cad'].mode()[0])
df['appet'] = df['appet'].fillna(df['appet'].mode()[0])
df['pe'] = df['pe'].fillna(df['pe'].mode()[0])
df['ane'] = df['ane'].fillna(df['ane'].mode()[0])
```

FIGURE 20- MEAN AND MODE METHODS CODE

The figure represents the dataset after missing values have been replaced.

```
[44] df
```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pot	hemo	wbcc	htn	dm	cad	appet	pe	ane	class
0	48.0	80.0	1.020	1.0	0.0	0.0	0.0	0.0	0.0	121.0	...	3.5	15.4	7800.0	1.0	1.0	0.0	1.0	0.0	0.0	1
1	7.0	50.0	1.020	4.0	0.0	0.0	0.0	0.0	0.0	99.0	...	3.5	11.3	6000.0	0.0	0.0	0.0	1.0	0.0	0.0	1
2	62.0	80.0	1.010	2.0	3.0	0.0	0.0	0.0	0.0	423.0	...	3.5	9.6	7500.0	0.0	1.0	0.0	0.0	0.0	1.0	1
3	48.0	70.0	1.005	4.0	0.0	0.0	1.0	1.0	0.0	117.0	...	2.5	11.2	6700.0	1.0	0.0	0.0	0.0	1.0	1.0	1
4	51.0	80.0	1.010	2.0	0.0	0.0	0.0	0.0	0.0	106.0	...	3.5	11.6	7300.0	0.0	0.0	0.0	1.0	0.0	0.0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
395	55.0	80.0	1.020	0.0	0.0	0.0	0.0	0.0	0.0	140.0	...	4.9	15.7	6700.0	0.0	0.0	0.0	1.0	0.0	0.0	0
396	42.0	70.0	1.025	0.0	0.0	0.0	0.0	0.0	0.0	75.0	...	3.5	16.5	7800.0	0.0	0.0	0.0	1.0	0.0	0.0	0
397	12.0	80.0	1.020	0.0	0.0	0.0	0.0	0.0	0.0	100.0	...	4.4	15.8	6600.0	0.0	0.0	0.0	1.0	0.0	0.0	0
398	17.0	60.0	1.025	0.0	0.0	0.0	0.0	0.0	0.0	114.0	...	4.9	14.2	7200.0	0.0	0.0	0.0	1.0	0.0	0.0	0
399	58.0	80.0	1.025	0.0	0.0	0.0	0.0	0.0	0.0	131.0	...	3.5	15.8	6800.0	0.0	0.0	0.0	1.0	0.0	0.0	0

400 rows × 23 columns

FIGURE 21 - DATASET WITHOUT THE MISSING VALUES

V - 3 - FEATURES SELECTION

Many mining algorithms typically do not function well with large numbers of features or attributes. As a result, before using any mining technique, a feature selection approach must be used. Feature selection is the process of determining the most discriminating features in a dataset. The primary goals of feature selection are to avoid overfitting, improve model performance, and provide faster and more cost-effective models. The correlation coefficient and recursive feature elimination (RFE) were applied as feature selection methods in this project.

V - 3 - 1 - ARTICLE I. BOOSTED CLASSIFIER AND FEATURES SELECTION FOR ENHANCING CHRONIC KIDNEY DISEASE DIAGNOSE

Using a Correlation Matrix (Correlation-based Feature Selection (CFS))
The code we used is shown in the figure below.

```
In [39]: # name of the label (can be seen in the dataframe)
         label = 'class'

         # list with feature names (V1, V2, V3, ...)
         features = df.columns.tolist()
         features

Out[39]: ['age',
          'bp',
          'sg',
          'al',
          'su',
          'rbc',
          'pc',
          'pcc',
          'ba',
          'bgr',
          'bu',
          'sc',
          'sod',
          'pot',
          'hemo',
          'pcv',
          'wbcc',
          'rbcc',
          'htn',
          'dm',
          'cad',
          'appet',
          'pe',
          'ane',
          'class']
```

FIGURE 22 - Correlation-based Feature Selection CODE 1

```
: from scipy.stats import pointbiserialr
  from math import sqrt

  def getMerit(subset, label):
      k = len(subset)

      # average feature-class correlation
      rcf_all = []
      for feature in subset:
          coeff = pointbiserialr( df[label], df[feature] )
          rcf_all.append( abs( coeff.correlation ) )
      rcf = np.mean( rcf_all )

      # average feature-feature correlation
      corr = df[subset].corr()
      corr.values[np.tril_indices_from(corr.values)] = np.nan
      corr = abs(corr)
      rff = corr.unstack().mean()

      return (k * rcf) / sqrt(k + k * (k-1) * rff)
```

FIGURE 22 - Correlation-based Feature Selection CODE 2

V - 3 - 2 - ARTICLE II. DIAGNOSIS OF CHRONIC KIDNEY DISEASE USING EFFECTIVE CLASSIFICATION ALGORITHMS AND RECURSIVE FEATURE ELIMINATION TECHNIQUES

Using Recursive Feature Elimination (RFE)

The code we used is shown in the figure below.

```
In [56]: from sklearn.feature_selection import RFE
         from sklearn.linear_model import LogisticRegression

In [57]: model = LogisticRegression()
         rfe = RFE(model, 4)
         fit = rfe.fit(X, y)
         print("Num Features: %s" % (fit.n_features_))
         print("Selected Features: %s" % (fit.support_))
         print("Feature Ranking: %s" % (fit.ranking_))

         Increase the number of iterations (max_iter) or scale the data as shown in:
             https://scikit-learn.org/stable/modules/preprocessing.html
         Please also refer to the documentation for alternative solver options:
             https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
           n_iter_i = _check_optimize_result(
         C:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Convergenc
         ge (status=1):
         STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

         Increase the number of iterations (max_iter) or scale the data as shown in:
             https://scikit-learn.org/stable/modules/preprocessing.html
         Please also refer to the documentation for alternative solver options:
             https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
           n_iter_i = _check_optimize_result(

         Num Features: 4
         Selected Features: [False False False  True False False False False False False False  True
          False False False False False False  True  True False False False False]
         Feature Ranking: [19 12 11  1  4  9  6 15 17 18 20  1 13  8  5 10 21 14  1  1 16  2  3  7]
```

FIGURE 24 - Recursive Feature Elimination CODE

As mentioned in the article, we will be using the four best features identified by the Recursive Feature Elimination technique (RFE).
For that, we set the number of features to four and ran the algorithm, which returned the best four features : al, sc, htn,dm.

VI - PHASE III. MODELING

VI - 1 - ARTICLE I. BOOSTED CLASSIFIER AND FEATURES SELECTION FOR ENHANCING CHRONIC KIDNEY DISEASE DIAGNOSE

VI - 1 - 1 - 1ST MODEL - NB - SVM - KNN - WITHOUT FEATURES SELECTION

In this first model, we will evaluate three classifiers, NB, KNN, and SVM, without using feature selection or ensamble learning.

NAIVE BAYES CLASSIFIER (NB)

The Naive Bayes classification approach is based on the Bayes Theorem. It is one of the most straightforward supervised learning methods. The Naive Bayes classifier is a quick, accurate, and dependable method. On large datasets, Naive Bayes classifiers have great accuracy and speed.

The code we used is shown in the figure below.

```
In [331]: #Import Gaussian Naive Bayes model
          from sklearn.naive_bayes import GaussianNB

          #Create a Gaussian Classifier
          gnb = GaussianNB()

          #Train the model using the training sets
          gnb.fit(X_train, y_train)

          #Predict the response for test dataset
          y_predNB = gnb.predict(X_test)
          y_predNB
```

```
array([0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1,
       0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0,
       1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1], dtype=int64)
```

FIGURE 25 - Naive Bayes CODE

EVALUATING THE MODEL

```
In [332]: #Import scikit-learn metrics module for accuracy calculation
          from sklearn import metrics

          # Model Accuracy, how often is the classifier correct?
          print("Accuracy:",metrics.accuracy_score(y_test, y_predNB))
          print("Precision:",metrics.precision_score(y_test, y_predNB))
          print("Recall:",metrics.recall_score(y_test, y_predNB))
          print("F-measure:",metrics.f1_score(y_test, y_predNB))
```

```
Accuracy: 0.98
Precision: 1.0
Recall: 0.9672131147540983
F-measure: 0.9833333333333333
```

FIGURE 26 - Evaluation CODE - NB

As we can see we obtained an accuracy rating of 98%, which is considered good.
a precision of 100%, a recall of 96.72%, and an F-measure of 98.33%

K-NEAREST NEIGHBORS CLASSIFIER (KNN)

K Nearest Neighbor (KNN) is one of the best machine learning algorithms since it is basic, easy to comprehend, and versatile.

In the figure below, we can see that the optimal k to use is 1 because it has the lowest error rate.

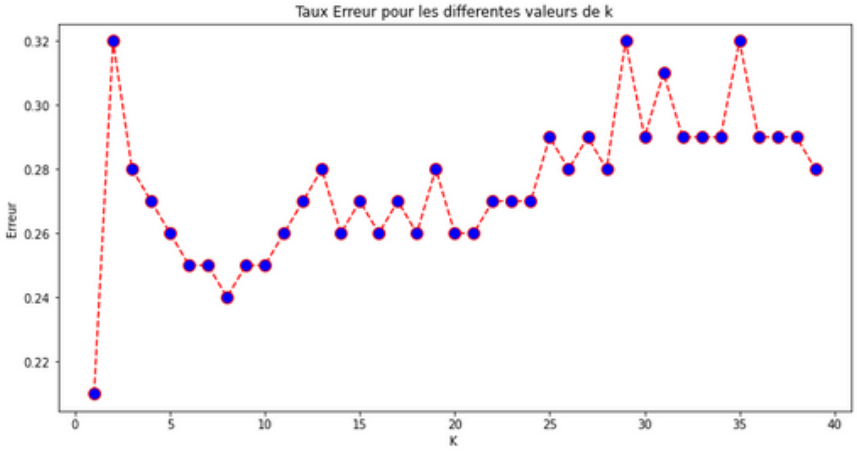

FIGURE 27 - Error Rate For All Different Values Of K

The code we used is shown in the figure below.

```
In [334]: #Import knearest neighbors Classifier model
          from sklearn.neighbors import KNeighborsClassifier

          #Create KNN Classifier
          knn = KNeighborsClassifier(n_neighbors=1)

          #Train the model using the training sets
          knn.fit(X_train, y_train)

          #Predict the response for test dataset
          y_predKNN = knn.predict(X_test)
          y_predKNN
```

```
array([1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0,
       1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1,
       0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1], dtype=int64)
```

FIGURE 28 - KNN CODE

EVALUATING THE MODEL

```
In [335]: #Import scikit-learn metrics module for accuracy calculation
          from sklearn import metrics
          # Model Accuracy, how often is the classifier correct?
          print("Accuracy:",metrics.accuracy_score(y_test, y_predKNN))
          print("Precision:",metrics.precision_score(y_test, y_predKNN))
          print("Recall:",metrics.recall_score(y_test, y_predKNN))
          print("F-measure:",metrics.f1_score(y_test, y_predKNN))
```

```
Accuracy: 0.79
Precision: 0.8448275862068966
Recall: 0.8032786885245902
F-measure: 0.823529411764706
```

FIGURE 29 - Evaluation CODE - KNN

We achieved an accuracy rating of 79%, which is considered acceptable. A precision of 84.48%, a recall of 80.32%, and an F-measure of 82.35%

SUPPORT VECTOR MACHINES (SVM)

When compared to other classifiers such as logistic regression and decision trees, SVM has relatively high accuracy. It is well-known for its kernel trick for dealing with nonlinear input spaces.

The code we used is shown in the figure below.

```
In [22]: #Import svm model
         from sklearn import svm

         #Create a svm Classifier
         clf = svm.SVC(kernel='linear') # Linear Kernel

         #Train the model using the training sets
         clf.fit(X_train, y_train)

         #Predict the response for test dataset
         y_predSVM = clf.predict(X_test)
         y_predSVM
```

```
array([0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1,
       0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0,
       1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0,
       1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1], dtype=int64)
```

FIGURE 30 - SVM CODE

EVALUATING THE MODEL

```
In [21]: #Import scikit-learn metrics module for accuracy calculation
         from sklearn import metrics
         # Model Accuracy, how often is the classifier correct?
         print("Accuracy:",metrics.accuracy_score(y_test, y_predSVM))
         print("Precision:",metrics.precision_score(y_test, y_predSVM))
         print("Recall:",metrics.recall_score(y_test, y_predSVM))
         print("F-measure:",metrics.f1_score(y_test, y_predSVM))
```

```
Accuracy: 0.97
Precision: 0.9833333333333333
Recall: 0.9672131147540983
F-measure: 0.9752066115702478
```

FIGURE 31 - Evaluation CODE - SVM

A good accuracy rating of 97%. A precision of 98.33%, a recall of 96.72%, and an
F-measure of 97.52%

VI - 1 - 2 - 1ST MODEL - NB - SVM - KNN - WITH FEATURS SELECTION (CFS)

In this first model, we will evaluate three classifiers, NB, KNN, and SVM, with using feature selection (CFS)

In this study, features were selected using a Correlation-based Feature Selection (CFS)
to improve the detection of CKD.
Like the first model K-Nearest Neighbour algorithm (kNN), Naive Bayes and Support Vector Machine (SVM) was used as base classifier.

FEATURE SUBSET SELECTION

Feature subset selection aims to reduce computing time and improve the results of prediction of machine learning algorithms.
This is done by reducing the features/attributes in a dataset that are considered unimportant or unable to contribute positively towards the classification.

GENERAL PRINCIPLE

The correlation-based feature selection (CFS) method is a filter approach and therefore independent of the final classification model. It evaluates feature subsets only based on data intrinsic properties, as the name already suggests: correlations.

The goal is to find a feature subset with low feature-feature correlation, to avoid redundancy, and high feature-class correlation to maintain or increase predictive power.

FIGURE 32 - Filter-based Features Selection

Before we dive deeper into the correlation-based feature selection we need to do some preprocessing of the dataset. First, we want to get the column names of all features and the class, respectively.

The code we used is shown in the figures below.

```
In [39]:
# name of the label (can be seen in the dataframe)
label = 'class'

# list with feature names (V1, V2, V3, ...)
features = df.columns.tolist()
features
```

```
Out[39]: ['age',
 'bp',
 'sg',
 'al',
 'su',
 'rbc',
 'pc',
 'pcc',
 'ba',
 'bgr',
 'bu',
 'sc',
 'sod',
 'pot',
 'hemo',
 'pcv',
 'wbcc',
 'rbcc',
 'htn',
 'dm',
 'cad',
 'appet',
 'pe',
 'ane',
 'class']
```

FIGURE 33 - CFS CODE -1-

COMPUTE MERIT

Now we are ready to go to the actual feature selection algorithm. The first thing to implement is the evaluation function (merit), which gets the subset and label name as inputs.

```
: from scipy.stats import pointbiserialr
from math import sqrt

def getMerit(subset, label):
    k = len(subset)

    # average feature-class correlation
    rcf_all = []
    for feature in subset:
        coeff = pointbiserialr( df[label], df[feature] )
        rcf_all.append( abs( coeff.correlation ) )
    rcf = np.mean( rcf_all )

    # average feature-feature correlation
    corr = df[subset].corr()
    corr.values[np.tril_indices_from(corr.values)] = np.nan
    corr = abs(corr)
    rff = corr.unstack().mean()

    return (k * rcf) / sqrt(k + k * (k-1) * rff)
```

FIGURE 34 - CFS CODE -2-

Calculating the average feature-class correlation is quite simple. We iterate through all features in the subset and compute for each feature its Point-biserial correlation coefficient using scipy's pointbiserialr function. As we are only interested in the magnitude of correlation and not the direction we take the absolute value. Thereafter we take the average over all features of that subset which is our

For the average feature-feature correlation it get's a little bit more complicated. We first do the correlation matrix of the subset. Let's assume our subset includes the first 4 features , (age,bp,sg,al). We compute the correlation matrix as follows: $\overline{r_{rf}}$.

```
In [42]: subset = ['age', 'bp', 'sg', 'al']
         corr = df[subset].corr()
         corr
```

Out[42]:

	age	bp	sg	al
age	1.000000	0.135503	-0.161959	0.087907
bp	0.135503	1.000000	-0.166980	0.123518
sg	-0.161959	-0.166980	1.000000	-0.479962
al	0.087907	0.123518	-0.479962	1.000000

FIGURE 35 - CFS CODE -3-

This results in a correlation matrix with redundant values as it is symmetrical and the correlation of each feature with itself is of course 1. Hence, we need to mask redundant values.

```
In [43]: corr.values[np.tril_indices_from(corr.values)] = np.nan
         corr
```

Out[43]:

	age	bp	sg	al
age	NaN	0.135503	-0.161959	0.087907
bp	NaN	NaN	-0.166980	0.123518
sg	NaN	NaN	NaN	-0.479962
al	NaN	NaN	NaN	NaN

FIGURE 36 - CFS CODE -4-

Now we can just unstack the absolute values and take their mean. That's it!

BEST FIRST SEARCH

As already described above, the search starts with iterating through all features and searching for the feature with the highest feature-class correlation.

```
In [44]: best_value = -1
         best_feature = ''
         for feature in features:
             coeff = pointbiserialr( df[label], df[feature] )
             abs_coeff = abs( coeff.correlation )
             if abs_coeff > best_value:
                 best_value = abs_coeff
                 best_feature = feature

         print("Feature %s with merit %.4f"%(best_feature, best_value))

         Feature sg with merit 0.6595
```

FIGURE 37 - CFS CODE -5-

The best first feature is the one with name sg, as it has the highest feature-class correlation

For the following best first search iterations we need a priority queue as data structure. Each item in this queue has a priority associated with it and at request the item with the highest priority will be returned. In our case the items are feature subsets and the priorities are the corresponding merits. For example, the subset with just the feature sg has the merit 0.6595 The implementation is quick and dirty for this blog, but of course it could be enhanced, for example using heap sort etc.

```
class PriorityQueue:
    def __init__(self):
        self.queue = []

    def isEmpty(self):
        return len(self.queue) == 0

    def push(self, item, priority):
        """
        item already in priority queue with smaller priority:
        -> update its priority
        item already in priority queue with higher priority:
        -> do nothing
        if item not in priority queue:
        -> push it
        """
        for index, (i, p) in enumerate(self.queue):
            if (set(i) == set(item)):
                if (p >= priority):
                    break
                del self.queue[index]
                self.queue.append( (item, priority) )
                break
        else:
            self.queue.append( (item, priority) )

    def pop(self):
        # return item with highest priority and remove it from queue
        max_idx = 0
        for index, (i, p) in enumerate(self.queue):
            if (self.queue[max_idx][1] < p):
                max_idx = index
        (item, priority) = self.queue[max_idx]
        del self.queue[max_idx]
        return (item, priority)
```

FIGURE 38 - CFS CODE -6-

Now we are prepared for the actual search. We first initialize a priority queue and push our first subset containing just one feature (sg).

```
# initialize queue
queue = PriorityQueue()

# push first tuple (subset, merit)
queue.push([best_feature], best_value)
```

FIGURE 39 - CFS CODE -7-

For the search we need a list of already visited nodes (subsets), a counter for the number of backtracks, and the maximum number of backtracks, as otherwise the search algorithm would search the whole feature subset space.

```
# list for visited nodes
visited = []

# counter for backtracks
n_backtrack = 0

# limit of backtracks
max_backtrack = 5
```

FIGURE 40 - CFS CODE -8-

The search algorithm works as follows: In the beginning only one feature is in the subset. The algorithm expands this subset with all other possible features and looks for the feature that increases the merit the most. For example, we have the feature sg in our subset. Now every feature is added to the subset and the merit of it evaluated. So the first expansion is the subset (sg, age), then (sg, bp), then (sg, al), and so on. The subset that achieves the highest merit is the new base set and again all features are one by one added and the subsets are evaluated.

```
# repeat until queue is empty
# or the maximum number of backtracks is reached
while not queue.isEmpty():
    # get element of queue with highest merit
    subset, priority = queue.pop()

    # check whether the priority of this subset
    # is higher than the current best subset
    if (priority < best_value):
        n_backtrack += 1
    else:
        best_value = priority
        best_subset = subset

    # goal condition
    if (n_backtrack == max_backtrack):
        break

    # iterate through all features and look of one can
    # increase the merit
    for feature in features:
        temp_subset = subset + [feature]

        # check if this subset has already been evaluated
        for node in visited:
            if (set(node) == set(temp_subset)):
                break
        # if not, ...
        else:
            # ... mark it as visited
            visited.append( temp_subset )
            # ... compute merit
            merit = getMerit(temp_subset, label)
            # and push it to the queue
            queue.push(temp_subset, merit)
```

FIGURE 41 - CFS CODE -9-

After a few minutes we get the subset that achieves the highest merit, which consists of 14 features. Compared to 24

But how does the subset perform compared to a model with all features?

CLASSIFICATION

To prove the functionality we will use three classification models which are Naive Bayes, Support Vector Machine (SVM), and KNN. We train one model with all features and compare it to a model using just the subset.

NAIVE BAYES CLASSIFIER (NB)

The code we used is shown in the figure below.

```
#Import Gaussian Naive Bayes model
from sklearn.naive_bayes import GaussianNB

#Create a Gaussian Classifier
gnb = GaussianNB()

#Train the model using the training sets
gnb.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = gnb.predict(X_test)
```

FIGURE 42- NB CODE

EVALUATING THE MODEL

```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_pred), ": is the confusion matrix")
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred), ": is the accuracy score")
from sklearn.metrics import precision_score
print(precision_score(y_test, y_pred), ": is the precision score")
from sklearn.metrics import recall_score
print(recall_score(y_test, y_pred), ": is the recall score")
from sklearn.metrics import f1_score
print(f1_score(y_test, y_pred), ": is the f1 score")
```

```
[[38  0]
 [ 4 58]] : is the confusion matrix
0.96 : is the accuracy score
1.0 : is the precision score
0.9354838709677419 : is the recall score
0.9666666666666666 : is the f1 score
```

FIGURE 43 - EVALUATION CODE NB

2nd method with CFS feature selection was able to increase the accuracy of base classifier.

K-NEAREST NEIGHBORS CLASSIFIER (KNN)

```
#Import knearest neighbors Classifier model
from sklearn.neighbors import KNeighborsClassifier

#Create KNN Classifier
knn = KNeighborsClassifier(n_neighbors=20)

#Train the model using the training sets
knn.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = knn.predict(X_test)
```

FIGURE 44 - KNN CODE

EVALUATING THE MODEL

```
In [72]: from sklearn.metrics import confusion_matrix
         print(confusion_matrix(y_test, y_pred), ": is the confusion matrix")
         from sklearn.metrics import accuracy_score
         print(accuracy_score(y_test, y_pred), ": is the accuracy score")
         from sklearn.metrics import precision_score
         print(precision_score(y_test, y_pred), ": is the precision score")
         from sklearn.metrics import recall_score
         print(recall_score(y_test, y_pred), ": is the recall score")
         from sklearn.metrics import f1_score
         print(f1_score(y_test, y_pred), ": is the f1 score")

         [[29  9]
          [16 46]] : is the confusion matrix
         0.75 : is the accuracy score
         0.8363636363636363 : is the precision score
         0.7419354838709677 : is the recall score
         0.7863247863247863 : is the f1 score
```

FIGURE 45 - EVALUATION CODE KNN

SUPPORT VECTOR MACHINES (SVM)

```
#Import svm model
from sklearn import svm

#Create a svm Classifier
clf = svm.SVC(kernel='linear') # Linear Kernel

#Train the model using the training sets
clf.fit(X_train, y_train)

#Predict the response for test dataset
y_predSVM = clf.predict(X_test)
```

EVALUATING THE MODEL

```
:  #Import scikit-learn metrics module for accuracy calculation
   from sklearn import metrics
   # Model Accuracy, how often is the classifier correct?
   print("Accuracy:",metrics.accuracy_score(y_test, y_predSVM))
   print("Precision:",metrics.precision_score(y_test, y_predSVM))
   print("Recall:",metrics.recall_score(y_test, y_predSVM))
   print("F-measure:",metrics.f1_score(y_test, y_predSVM))

   Accuracy: 0.95
   Precision: 0.9384615384615385
   Recall: 0.9838709677419355
   F-measure: 0.9606299212598426
```

FIGURE 46 - EVALUATION CODE SVM

VI - 1 - 3 - SECOND MODEL - NB - SVM - KNN - WITH FEATURES SELECTION (CFS) AND ADABOOST

The code we used is shown in the figure below.

Adaboost Classifier

Let's first load the required libraries.

In [69]:
```
from sklearn.ensemble import AdaBoostClassifier
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

Split dataset

To understand model performance, dividing the dataset into a training set and a test set is a good strategy.

Let's split dataset by using function train_test_split(). you need to pass 3 parameters features, target, and test_set size.

In [70]:
```
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

FIGURE 46 - ADABOOST CODE

In [72]:
```
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 1.0
```

Pros

AdaBoost is easy to implement. It iteratively corrects the mistakes of the weak classifier and improves accuracy by combining weak learners. You can use many base classifiers with AdaBoost. AdaBoost is not prone to overfitting. This can be found out via experiment results, but there is no concrete reason available

Cons

AdaBoost is sensitive to noise data. It is highly affected by outliers because it tries to fit each point perfectly. AdaBoost is slower compared to XGBoost.

VI - 2 - ARTICLE II. DIAGNOSIS OF CHRONIC KIDNEY DISEASE USING EFFECTIVE CLASSIFICATION ALGORITHMS AND RECURSIVE FEATURE ELIMINATION TECHNIQUES

We will be testing and evaluating all of the models from SVM, KNN Decision Tree, and Random Forest.

VI - 2 - 1 - RANDOM FOREST

The code we used is shown in the figure below.

Random Forest ¶

```
In [58]: from sklearn.model_selection import train_test_split

In [59]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

In [60]: from sklearn.ensemble import RandomForestClassifier

In [61]: clf=RandomForestClassifier(n_estimators=100)

In [62]: clf.fit(X_train,y_train)
Out[62]: RandomForestClassifier()

In [63]: y_pred=clf.predict(X_test)
```

FIGURE 19 - Random Forest CODE

EVALUATING THE MODEL

```
In [79]: from sklearn import metrics
         # Model Accuracy, how often is the classifier correct?

         print("Accuracy:",(metrics.accuracy_score(y_test, y_pred))*100)
         print("Precision:",(metrics.precision_score(y_test, y_pred))*100)
         print("Recall:",(metrics.recall_score(y_test, y_pred))*100)
         print("F-measure:",(metrics.f1_score(y_test, y_pred))*100)

         Accuracy: 100.0
         Precision: 100.0
         Recall: 100.0
         F-measure: 100.0
```

FIGURE 19 - Evaluation CODE - Random Forest

It is proven by scientists that 100% train accuracy with Random Forest has nothing to do with overfitting due to the fact that it consists of fully grown trees, bootstrapped data and the majority vote rule to make predictions.
As commonly known, overfitting is when a model is no longer as accurate as we want it to be on data we care about and this is not the case with the random forest.
So we can conclude that this model is an accurate one.

VI - 2 - 2 - DECISION TREE

A decision tree is a tree structure that looks like a flowchart, with an internal node representing a feature (or attribute), a branch representing a decision rule, and each leaf node representing the outcome. The root node is the node at the top of a decision tree. It learns to partition based on attribute value.

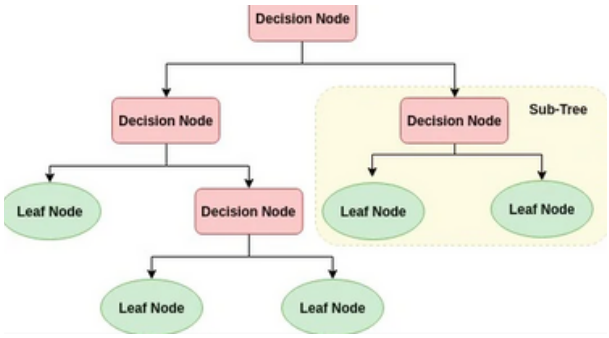

FIGURE 19 - Decision Tree Example

The code we used is shown in the figure below.

Decision Tree

```
In [99]: from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
         # Create Decision Tree classifer object
         clf = DecisionTreeClassifier()

         # Train Decision Tree Classifer
         clf = clf.fit(X_train,y_train)

         #Predict the response for test dataset
         y_predt = clf.predict(X_test)
         y_predt

Out[99]: array([0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1,
                1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1,
                1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1,
                1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0,
                0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1], dtype=int64)
```

FIGURE 19 - Decision Tree CODE

We begin by explaining how a decision tree algorithm works. Choose the best attribute, make it a decision node, then divide the dataset into smaller subsets. Begin tree construction by repeating this process for each child until one of the conditions is met: either all the tuples belong to the same attribute value, there are no more leftover attributes, or there are no more instances.

EVALUATING THE MODEL

Accuracy can be calculated by comparing actual test set values to predicted values. Let's see how effectively the classifier or model can predict if the disease is present or not.

```
In [77]: from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
         # Model Accuracy, how often is the classifier correct?

         print("Accuracy:",(metrics.accuracy_score(y_test, y_predt))*100)
         print("Precision:",(metrics.precision_score(y_test, y_predt))*100)
         print("Recall:",(metrics.recall_score(y_test, y_predt))*100)
         print("F-measure:",(metrics.f1_score(y_test, y_predt))*100)

         Accuracy: 99.0
         Precision: 100.0
         Recall: 98.27586206896551
         F-measure: 99.13043478260869
```

FIGURE 19 - Evaluation CODE - Decision Tree

We obtained an accuracy rating of 99%, which is considered good. a precision of 100%, a recall of 98.27%, and an F-measure of 99.13%

VISUALIZING DECESION TREE

Each internal node in the decision tree chart has a decision rule that separates the data. Gini refers to the Gini ratio, which estimates the node's impurity. A node is said to be pure if all of its records belong to the same class; such nodes are known as the leaf node.

The resulting tree is shown in the figure below.

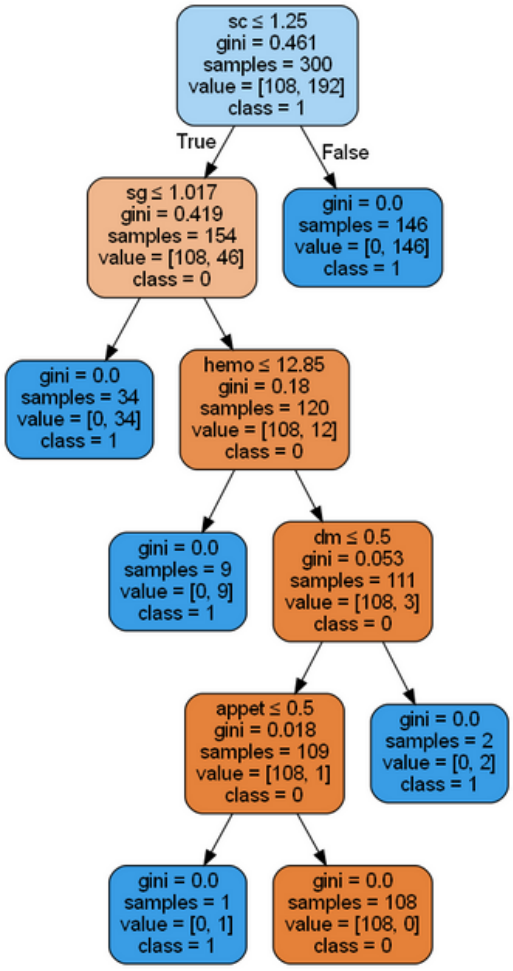

FIGURE 19 - Decision Tree

Decision trees are easy to comprehend and visualize. It can also recognize non-linear patterns. It requires less data preprocessing from the user, such as avoiding the need to normalize columns.
It is suitable for variable selection and can be used for feature engineering, such as missing value prediction.
The decision tree makes no assumptions about distribution because the approach is non-parametric. It is, nevertheless, prone to noisy data. It is capable of overfitting noisy data.
A small difference in data (or variance) can result in a different decision tree. Techniques such as bagging and boosting can assist in reducing this.
Because we are working with unbalanced data, we know decision trees are biased when the dataset is uneven; therefore, it is advised that the dataset be balanced before generating the decision tree.

VI - 2- 3 - KNN

KNN algorithm works on the similarity between new and stored data points (training points) and classifies the new test point into the most similar class among the available classes.

KNN algorithm is nonparametric, and it is called the lazy learning algorithm, meaning that it does not learn from the training dataset, but rather stores the training
dataset. When classifying the new dataset (test data), it classifies the new data based on the value of k, where it uses the Euclidean distance to measure the distance
between the new point and the stored training points. the e new point is classified into a class with the maximum number of neighbors. %e Euclidean distance function

Choosing the value of K

K is a critical parameter in the K-NN algorithm. Hence, we need to keep in mind some points before we decide on a value of K.

Using error curves is a common method to determine the value of K.

The code we used is shown in the figure below.

KNN

```
In [127]: from sklearn.neighbors import KNeighborsClassifier

In [128]: error = []
          # Calculer l'erreur pour k entre 1 et 40
          #Pour chaque itération, l'erreur moyenne pour les valeurs prédites
          #de l'ensemble de test est calculée et sauvegardée ds la liste Erreur.
          for i in range(1, 40):
              knn = KNeighborsClassifier(i)
              knn_model = knn.fit(X_train, y_train)
              pred_i = knn_model.predict(X_test)
              error.append(np.mean(pred_i != y_test))
          plt.figure(figsize=(12, 6))
          plt.plot(range(1, 40), error, color='red', linestyle='dashed', marker='o',
                   markerfacecolor='blue', markersize=10)
          plt.title('Taux Erreur pour les differentes valeurs de k')
          plt.xlabel('K ')
          plt.ylabel('Erreur')

Out[128]: Text(0, 0.5, 'Erreur')
```

FIGURE 19 - KNN CODE

According to this graph below, the best variation is given in the point of absciss 11.

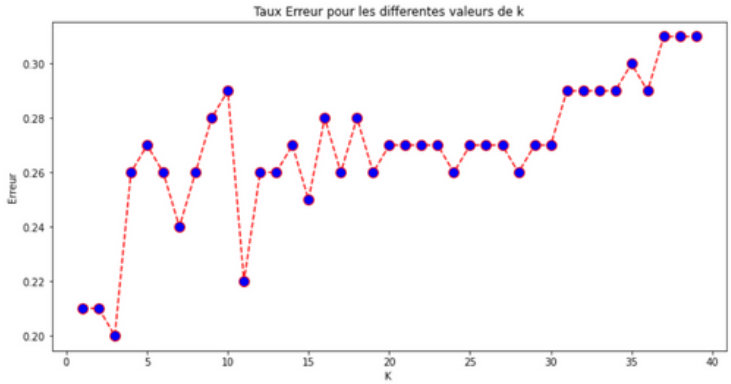

FIGURE 27 - Error Rate For All Different Values Of K

```
In [132]: #Import knearest neighbors Classifier model
          from sklearn.neighbors import KNeighborsClassifier

          #Create KNN Classifier
          knn = KNeighborsClassifier(n_neighbors=11)

          #Train the model using the training sets
          knn.fit(X_train, y_train)

          #Predict the response for test dataset
          y_pred = knn.predict(X_test)
```

EVALUATING THE MODEL

```
In [135]: from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
          # Model Accuracy, how often is the classifier correct?

          print("Accuracy:",(metrics.accuracy_score(y_test, y_pred))*100)
          print("Precision:",(metrics.precision_score(y_test, y_pred))*100)
          print("Recall:",(metrics.recall_score(y_test, y_pred))*100)
          print("F-measure:",(metrics.f1_score(y_test, y_pred))*100)

          Accuracy: 78.0
          Precision: 96.0
          Recall: 70.58823529411765
          F-measure: 81.35593220338983
```

FIGURE 19 - Evaluation CODE - KNN

We obtained an accuracy rating of 78%,
a precision of 96%, a recall of 70%, and an F-measure of 81.35%

VI - 2- 4 - SVM

The code we used is shown in the figure below.

Entrée [51]:
```
# divide the data into 80% training set and 20% test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=22)
```

Params by default

Entrée [52]:
```
# svm model
svm = SVC()
svm.fit(X_train, y_train)
```

Out[52]:
```
SVC()
```

Entrée [62]:
```
from sklearn.metrics import classification_report
y_pred_svc_1 = svm.predict(X_test)
print(classification_report(y_pred_svc_1, y_test))
```

FIGURE 19 - SVM CODE

EVALUATING THE MODEL

Linear kernel

Entrée [63]:
```
# Kernel linear
svm = SVC(kernel="linear")
svm.fit(X_train, y_train)
from sklearn.metrics import classification_report
y_pred_svc_2 = svm.predict(X_test)
print(classification_report(y_pred_svc_2, y_test))
```

```
              precision    recall  f1-score   support

           0       0.97      0.85      0.90        33
           1       0.90      0.98      0.94        47

    accuracy                           0.93        80
   macro avg       0.93      0.91      0.92        80
weighted avg       0.93      0.93      0.92        80
```

FIGURE 19 - Evaluation CODE - SVM 1

Polynomial kernel

Entrée [64]:
```
# Kernel poly
svm = SVC(kernel="poly")
svm.fit(X_train, y_train)
from sklearn.metrics import classification_report
y_pred_svc_3 = svm.predict(X_test)
print(classification_report(y_pred_svc_3, y_test))
```

```
              precision    recall  f1-score   support

           0       0.72      0.64      0.68        33
           1       0.76      0.83      0.80        47

    accuracy                           0.75        80
   macro avg       0.74      0.73      0.74        80
weighted avg       0.75      0.75      0.75        80
```

FIGURE 19 - Evaluation CODE - SVM 2

Sigmoid kernel

Entrée [65]:
```
# Kernel sigmoid
svm = SVC(kernel="sigmoid")
svm.fit(X_train, y_train)
from sklearn.metrics import classification_report
y_pred_svc_4 = svm.predict(X_test)
print(classification_report(y_pred_svc_4, y_test))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         9
           1       0.82      0.59      0.69        71

    accuracy                           0.53        80
   macro avg       0.41      0.30      0.34        80
weighted avg       0.73      0.53      0.61        80
```

FIGURE 19 - Evaluation CODE - SVM 3

Comparative table

Entrée [71]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
#We will display a table that orders the models from the
s1=accuracy_score(y_test, y_pred_svc_1)
s2=accuracy_score(y_test, y_pred_svc_2)
s3=accuracy_score(y_test, y_pred_svc_3)
s4=accuracy_score(y_test, y_pred_svc_4)

models = pd.DataFrame({
    'Model': ['Support Vector Machines RBF', 'Support Vec
              'Support Vector Machines sigmoïde'],
    'Score': [s1,s2, s3,
              s4]})
models.sort_values(by="Score",ascending=False)
```

Out[71]:

	Model	Score
1	Support Vector Machines Linaire	0.9250
2	Support Vector Machines polynomiale	0.7500
0	Support Vector Machines RBF	0.7125
3	Support Vector Machines sigmoïde	0.5250

FIGURE 19 - COMPARATIVE TABLE - SVM

```
plt.show()
```

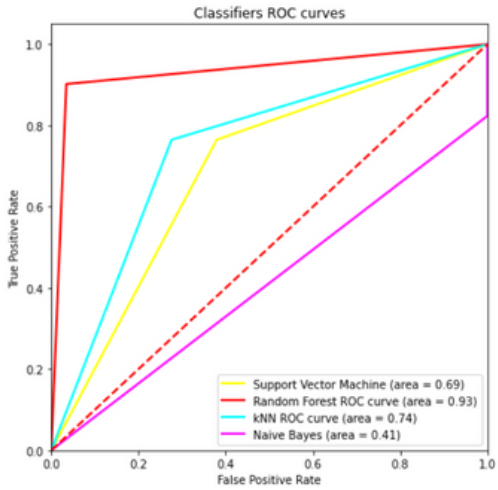

FIGURE 19 - COMPARATIVE GRAPH - SVM

SVMs are a family of machine learning algorithms that can be used to solve classification, regression and anomaly detection problems. They are known for their strong theoretical guarantees, their great flexibility as well as their simplicity of use even without much knowledge of data mining.

For this model ,svm Linear is the best solution . We obtained an accuracy rating of 92%, which is considered good.

VII - PHASE IV. OVERALL EVALUATION

VII - 1 - ARTICLE I. BOOSTED CLASSIFIER AND FEATURES SELECTION FOR ENHANCING CHRONIC KIDNEY DISEASE DIAGNOSE

To obtain the best model, we computed the accuracies as shown below.
The model which solves best the problem is the one with th ehighest accuracy which is the first model built with NB with the accuracy rate of 98%.

Models	NB	KNN	SVM
FIRST MODEL	98%	79%	97%
SECOND MODEL	96%	75%	95%

VII - 2 - ARTICLE II. DIAGNOSIS OF CHRONIC KIDNEY DISEASE USING EFFECTIVE CLASSIFICATION ALGORITHMS AND RECURSIVE FEATURE ELIMINATION TECHNIQUES

The purpose of developing these machine learning models is to create an objective model capable of performing an accurate diagnosis of the patient's health status, and for that, as mentioned in the second article, we created the Random Forest classifier, the Decision Tree classifier, SVM, and KNN.
To decide which of these models is the most accurate, we assessed the accuracy rate, which led us to the conclusion that the Random Forest Classifier is the most accurate since it has the highest accuracy of all as shown below.

Models	Random Forest	Decision Tree	KNN	SVM
Accuracy Rate (%)	100	99	78	93

BIBLIOGRAPHY

[1]:https://www.cdc.gov/kidneydisease/basics.html#:~:text=disease%20(CKD).-,About%20Chronic%20Kidney%20Disease,as%20heart%20disease%20and%20stroke.

[2]:https://www.ibm.com/docs/en/spss-modeler/saas?topic=dm-crisp-help-overview

[3]https://www.datacamp.com/tutorial/decision-tree-classification-python

[4]https://www.datacamp.com/tutorial/naive-bayes-scikit-learn

[5]https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn

[6]https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python

[7]https://benchpartner.com/q/what-are-overfitting-and-underfitting-why-does-the-decision-tree-algorithm-suffer-often-with-overfitting-problem