NOV.2022

REPORT ON THE DIAGNOSIS OF CHRONIC KIDNEY DISEASE

PREPARED BY :
ICHRAK HAMDI
MOHAMED AMRO
ABIR TABARKI
AHLEM ZAGHOUANI
NOUR EL HOUDA ZRAIBIA

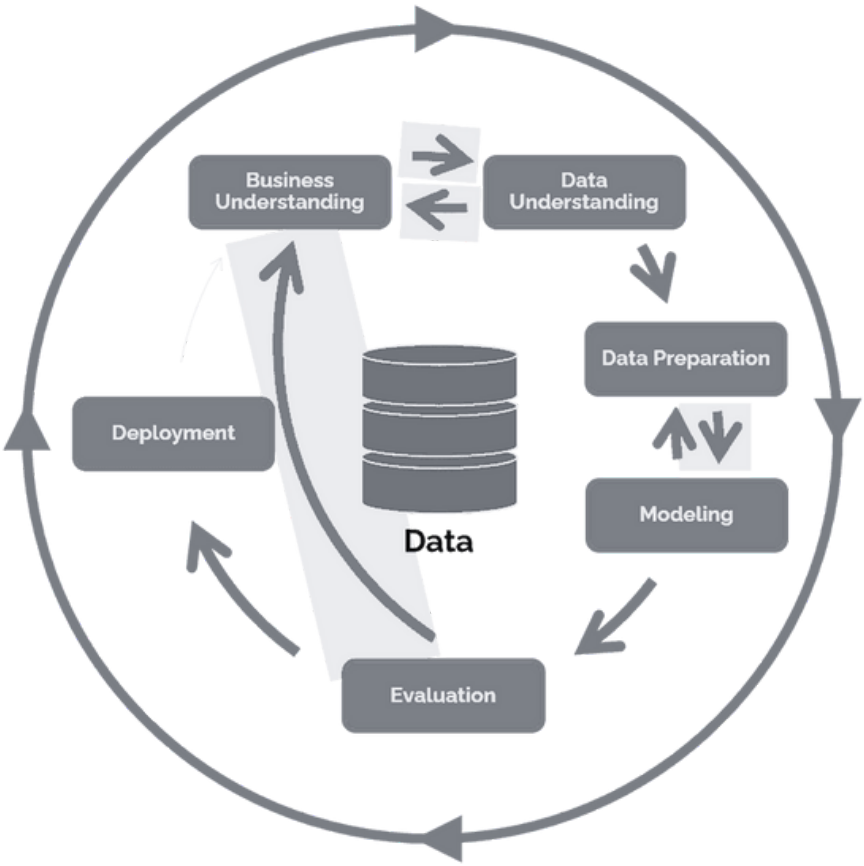
Contents

METHODOLOGY

This study used the CRISP-DM methodology, which defines a non-rigid sequence of six phases that allows the construction and implementation of a DM model to be used in a real environment, helping to support business decisions.

The CRISP-DM lifecycle is composed by the following stages:

- Business Understanding,
- Data Understanding,
- Data Preparation,
- Modeling,
- Evaluation,
- Deployment.



I-Business understanding:

1-Presenting the problem

1-1-What is chronic kidney disease?

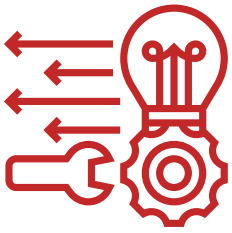
Chronic kidney disease includes conditions that damage your kidneys and decrease their ability to keep you healthy by filtering wastes from your blood. If kidney disease worsens, wastes can build to high levels in your blood and make you feel sick. You may develop complications like:

- high blood pressure
- anemia (low blood count)
- weak bones
- poor nutritional health
- nerve damage

Kidney disease also increases your risk of having heart and blood vessel disease. These problems may happen slowly over a long time. Early detection and treatment can often keep chronic kidney disease from getting worse. When kidney disease progresses, it may eventually lead to kidney failure, which requires dialysis or a kidney transplant to maintain life.

1-2-What are the main causes of chronic kidney disease?

Diabetes and high blood pressure, or hypertension, are responsible for two-thirds of chronic kidney disease cases.
Diabetes: Diabetes occurs when your blood sugar remains too high. Over time, unmanaged blood sugar can cause damage to many organs in your body, including the kidneys and heart and blood vessels, nerves, and eyes.
High blood pressure: High blood pressure occurs when your blood pressure against the walls of your blood vessels increases. If uncontrolled or poorly controlled, high blood pressure can be a leading cause of heart attacks, strokes, and chronic kidney disease. Also, chronic kidney disease can cause high blood pressure.



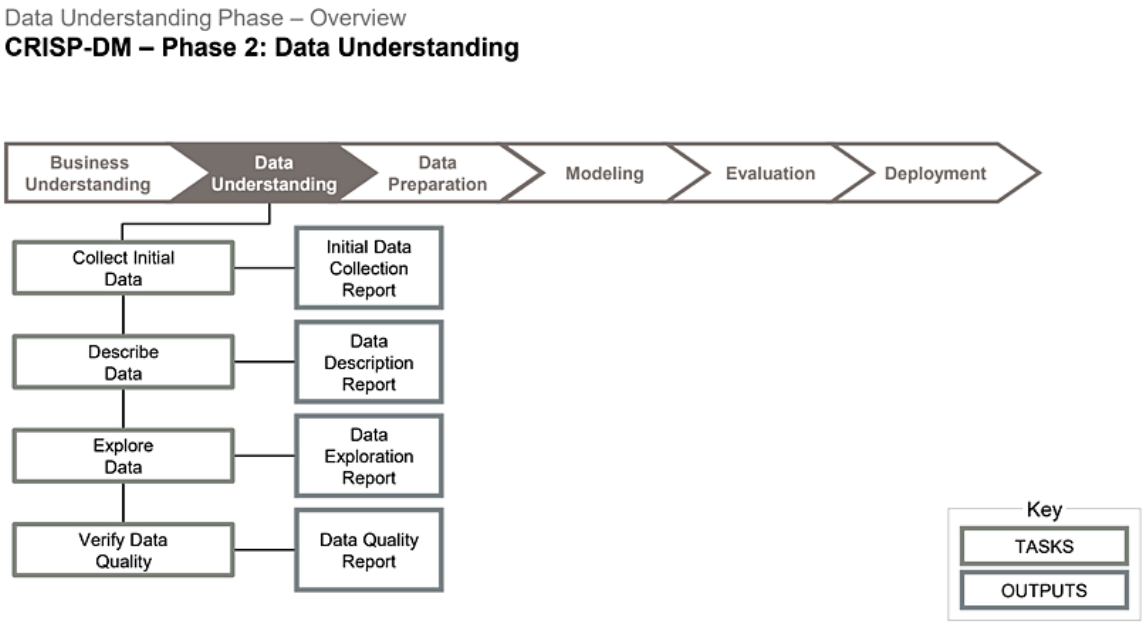
2-Impact and Consequences of not solving the problem:

- CKD is top ten leading causes of death in the world
- kidney failure is the end-stage impact of the chronic kidney disease and the symptoms can be seen from the complications due to kidney function. When the symptoms become more severe, the disease can only be treated with dialysis and transplantation.
- The average number of chronic kidney disease incidents reached more than 200 cases per million people in a year in many countries in the world. Some countries such as the USA, Taiwan and some areas in Mexico have reached 400 cases per million people per year. There are two leading causes of chronic kidney disease, i.e. diabetes and hypertension. In 2012, diabetes and hypertension are categorized among the world's top 10 causes of death. Diabetes was ranked ninth and killed 1.5 million people.meanwhile, hypertension was ranked tenth and caused 1.1 million death.

3-Why is it so important to solve the problem?

If we can detect it early we can prevent it.
Such a disease can be cured if the diagnosis is made appropriately and quickly,
and because there are relatively few nephrologists, not all CKD patients receive a proper diagnosis. The level of awareness of chronic kidney disease in various healthcare providers remains low even though they have more than 10 years of experience in the medical field. Therefore, an automated and accurate diagnostic method for CKD is essential to help medical personnel

II-Data understanding:



1.DATA COLLECTION :

To carry out this study, the CKD dataset used in this research was extracted from the UCI Machine Learning Repository , collected from the Apollo Hospitals, India , by b. Jerlin Rubini (Research Scholar), who was guided by Dr.P.Eswaran Assistant Professor, Department of Computer Science and Engineering at Alagappa University,India.
They permitted and authorized the processing of this data during a nearly 2-month period in 2015.
The dataset includes a total of 400 patient samples from the University of California .

2. DATA DESCRIPTION :

The CKD issuer has gathered information on 400 people,The dataset contains information on 24 variables (11 numeric ,14 nominal) including patient age , vital signs , measures of 8 different substances in the subject's body and tests about the existence of 5 diseases and appetite state and finally information about the subject class : is the subject diagnosed or not with CKD? which can be (CKD or notCKD).
Table 1 and 2 lists the attributes from the original dataset.

Name	Description	Type: unit/ values
bp	hypertension	nominal feature (yes,no)
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	diabetes mellitus	nominal feature (yes,no)
bu	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. Nominal Attributes description of CKD dataset

Name	Description	Type: unit/ values
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	blood pressure	numerical feature (in mm/Hg)
dm	blood glucose random	numerical feature (in mgs/dl)
cad	blood urea	numerical feature (in mgs/dl)
sc	serum creatinine	numerical feature (in mgs/dl)
sod	sodium	numerical feature (in mEq/L)
age	Age	numerical feature (in years)

Table 2. Numerical Attributes description of CKD dataset

A.NUMERICAL FEATURES :

Blood Pressure

Blood pressure is a measure of the force that your heart uses to pump blood around your body.
It is measured in millimetres of mercury (mmHg) and is given as 2 figures:

- systolic pressure – the pressure when your heart pushes blood out
- diastolic pressure – the pressure when your heart rests between beats

Blood Glucose Random

Random blood sugar (RBS) or random blood glucose (RBG) measures blood glucose regardless of when you last ate. Several random measurements may be taken throughout the day. Random testing is useful because glucose levels in healthy people do not vary widely throughout the day.
A blood sugar level of 200 mg/dL or higher indicates you have diabetes.

Blood Urea

A BUN test measures the amount of urea nitrogen that's in your blood. Urea nitrogen levels in your blood are one marker that allows healthcare providers to understand how well your kidneys are working. A small amount of urea nitrogen in your blood is normal. If you have too much urea nitrogen in your blood, your kidneys aren't filtering it properly. You may have a condition that's affecting your kidneys health.

Serum Creatinine

Creatinine is a waste product in your blood that comes from your muscles. Healthy kidneys filter creatinine out of your blood through your urine
Serum creatinine test results are measured in milligrams per deciliter (mg/dL).
Normal creatinine levels vary based on your sex, age and amount of muscle. In general, a normal level is:

- 0.7 - 1.3 mg/dL for males
- 0.6 - 1.1 mg/dL for females

Sodium

The sodium blood test measures the concentration of sodium in the blood. Sodium can also be measured using a urine test.
Kidneys get rid of the extra sodium in your urine. If your sodium blood levels are too high or too low, it may mean that you have a kidney problem,
The normal range for blood sodium levels is 135 to 145 milliequivalents per liter (mEq/L).

Hemoglobin

A hemoglobin test measures the levels of hemoglobin in your blood. Hemoglobin is a protein in your red blood cells that carries oxygen from your lungs to the rest of your body.
The normal ranges

- Women 12.0 to 15.5 gm/dl
- Men 13.5 to 17.5 gm/dl
- Children 11 to 16 g/dl
- Pregnant Women 11 to 12 g/dl

Packed Cell Volume

Also known as the haematocrit test, the PCV or Packed Cell Volume Test is a test done to diagnose polycythaemia, dehydration or anaemia in certain patients.
The PCV test measures how much of the blood consists of cells. If the PCV returns a reading of 50%, it means that 50 ml of the cells are present in exactly 100 ml of blood.
There are certain conditions that contribute to the low reading in the PCV. These include:Kidney diseases
The normal ranges

- In females, 35.5 to 44.9%.
- In males, 38.3% to 48.6%
- For pregnant females 33-38%.

White Blood Cell Count

A WBC count is a blood test to measure the number of white blood cells (WBCs) in the blood.
The normal number of WBCs in the blood is 4,500 to 11,000 WBCs per microliter (4.5 to 11.0 × 109/L).

White Blood Cell Count

A WBC count is a blood test to measure the number of white blood cells (WBCs) in the blood.
The normal number of WBCs in the blood is 4,500 to 11,000 WBCs per microliter (4.5 to 11.0 × 109/L).

Red Blood Cell Count

A red blood cell (RBC) count measures the number of red blood cells, also known as erythrocytes, in your blood
A normal range in adults

- 4.35 to 5.65 million red blood cells per microliter (mcL) of blood for men
- 3.92 to 5.13 million red blood cells per mcL of blood for women.

B.CATEGORICAL FEATURES :

Specific Gravity

Urine specific gravity is a laboratory test that shows the concentration of all chemical particles in the urine.
The normal range for urine specific gravity is 1.005 to 1.030.
Decreased urine specific gravity may be due to Kidney failure

Albumin

An albumin blood test measures the amount of albumin in your blood. Low albumin levels can be a sign of liver or kidney disease or another medical condition. Albumin is a protein made by your liver.
The normal range is 3.4 to 5.4 g/dL (34 to 54 g/L).

Sugar

A blood sugar test can be used to determine the amount of glucose in the blood.
A fasting blood sugar :

- level of 99 mg/dL or lower is normal
- 100 to 125 mg/dL indicates you have prediabetes
- and 126 mg/dL or higher indicates you have diabetes.

Red Blood Cells

Red blood cells are round with a flattish, indented center, like doughnuts without a hole. Your healthcare provider can check on the size, shape, and health of your red blood cells using a blood test.
The normal and abnormal morphology of the red blood cells of a patient is very helpful to doctors in detecting a disease. With the advancement of digital image processing technology can be used to identify normal and abnormal blood cells of a patient.

Pus Cell

Pus cells are a collection of dead, white blood cells that accumulates when the body's immune system activates in response to an infection resulting from normal and abnormal body processes occurring in an individual.

Pus Cells Clumps

In the case of laboratory investigations, urine analysis is the third major screening test advised by physicians. This analysis is generally conducted to detect and screen the various byproducts excreted through urine,
The main reasons that can lead to the occurrence of pus cells in urine include: kidneys.
The normal range for pus cells from the urine is 0-5.

Bacteria

A bacteria culture is a test to confirm whether you have a bacterial infection. The test can also identify what type of bacteria caused the infection, which helps guide treatment decisions. For a bacteria culture test, a healthcare provider takes a sample of blood, stool, urine, skin, mucus or spinal fluid.
up to 10,000 colonies of bacteria/ml are considered normal.

Hypertension

High blood pressure, also called hypertension, is blood pressure that is higher than normal. Your blood pressure changes throughout the day based on your activities. Having blood pressure measures consistently above normal may result in a diagnosis of high blood pressure (or hypertension).
Normal range :

- 140/90 mm Hg for people younger than age 65
- 150/80 mm Hg for those ages 65 and older.

Coronary Artery Disease

Coronary artery disease is a common heart condition. The major blood vessels that supply the heart (coronary arteries) struggle to send enough blood, oxygen and nutrients to the heart muscle.

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.

Anemia

Anemia is a condition in which you lack enough healthy red blood cells to carry adequate oxygen to your body's tissues. Having anemia, also referred to as low hemoglobin, can make you feel tired and weak.

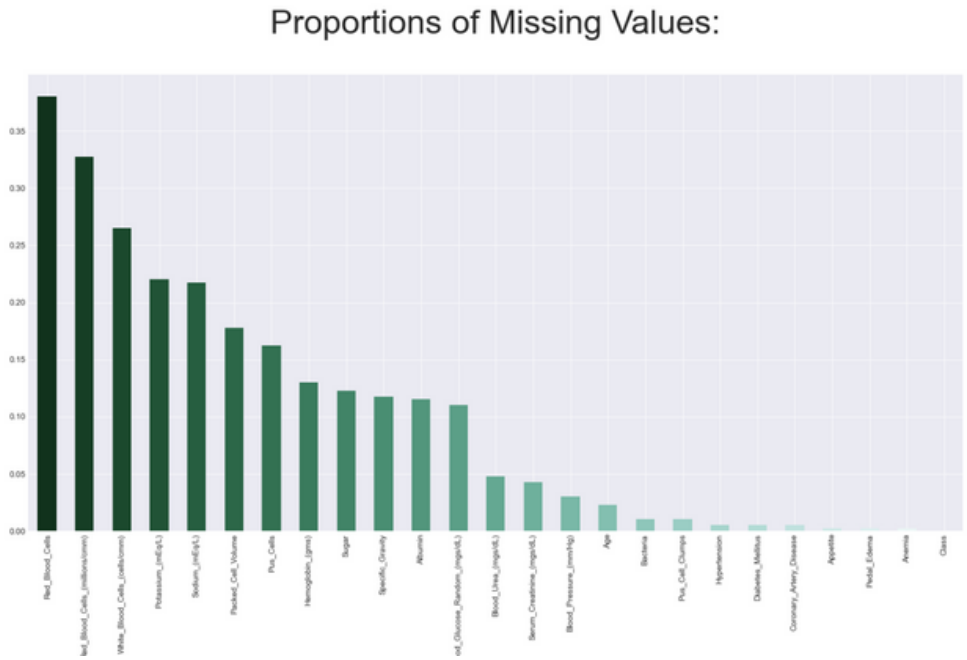
3. VERIFICATION OF THE QUALITY OF DATA:

The purpose of data verification is to test that the gathered data are as accurate as possible.
Everything from spelling errors to inaccurate numbers to data loss could jeopardize the process , so the dataset needs to be checked to ensure its consistency .

- Description of missing values from our data

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
wc	106
rc	131
htn	2
dm	2
cad	2
appet	1
pe	1
ane	1
classification	0

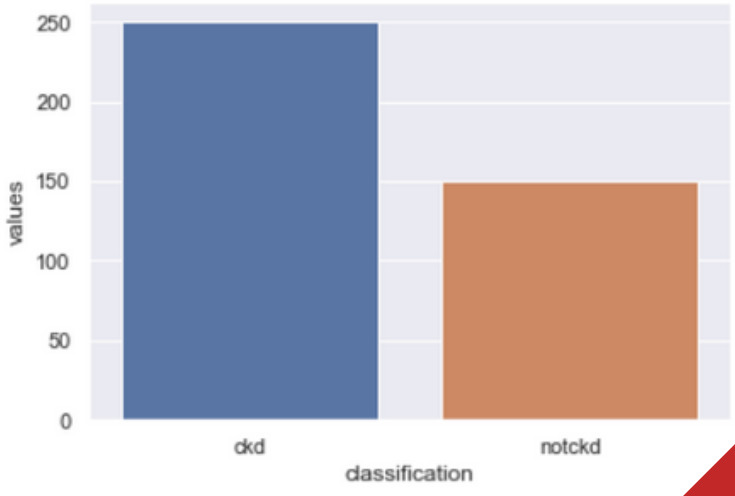
dtype: int64



It must be noted every attribute contained missing values except the class attribute, due to possibly to the fault of the receiver input, sensor error or reluctance on data resource.

- Description of data :

Out of 400 samples, 250 samples belonged to the CKD group (62.5%), and the other 150 samples to the non-CKD group by 37.5% implying an imbalanced dataset.



4.DATA EXPLORATION :

We started by importing libraries and loading our dataset and exploring it.

Import libraries

```
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import matplotlib.pyplot as plt
import os , sys
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import seaborn as sns #beautiful visualisation
sns.set()
from sklearn.preprocessing import StandardScaler
import seaborn as sns
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

Loading Data

```
import pandas as pd

data = []
with open('chronic_kidney_disease (1).arff', "r") as f:
    for line in f:
        line = line.replace('\n', '')
        data.append(line.split(','))


names = ['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', 'no_name']

df = pd.DataFrame(data, columns=names)
```

```
df.shape
```

```
(400, 26)
```

Display features names

```
df.columns
```

```
Index(['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', 'no_name'],
      dtype='object')
```

display 5 first rows

```
df.head()
```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	wbcc	rbcc	htn	dm	cad	appet	pe	ane	class	no_name
0	48	80	1.020	1	0	?	normal	notpresent	notpresent	121	...	7800	5.2	yes	yes	no	good	no	no	ckd	None
1	7	50	1.020	4	0	?	normal	notpresent	notpresent	?	...	6000	?	no	no	no	good	no	no	ckd	None
2	62	80	1.010	2	3	normal	normal	notpresent	notpresent	423	...	7500	?	no	yes	no	poor	no	yes	ckd	None
3	48	70	1.005	4	0	normal	abnormal	present	notpresent	117	...	6700	3.9	yes	no	no	poor	yes	yes	ckd	None
4	51	80	1.010	2	0	normal	normal	notpresent	notpresent	106	...	7300	4.6	no	no	no	good	no	no	ckd	None

5 rows × 26 columns

Extract all the values used for each column to better understand the defects of our dataset:

```
for i in df.columns:
    print('unique values in "{}":\n'.format(i),df[i].unique())
```

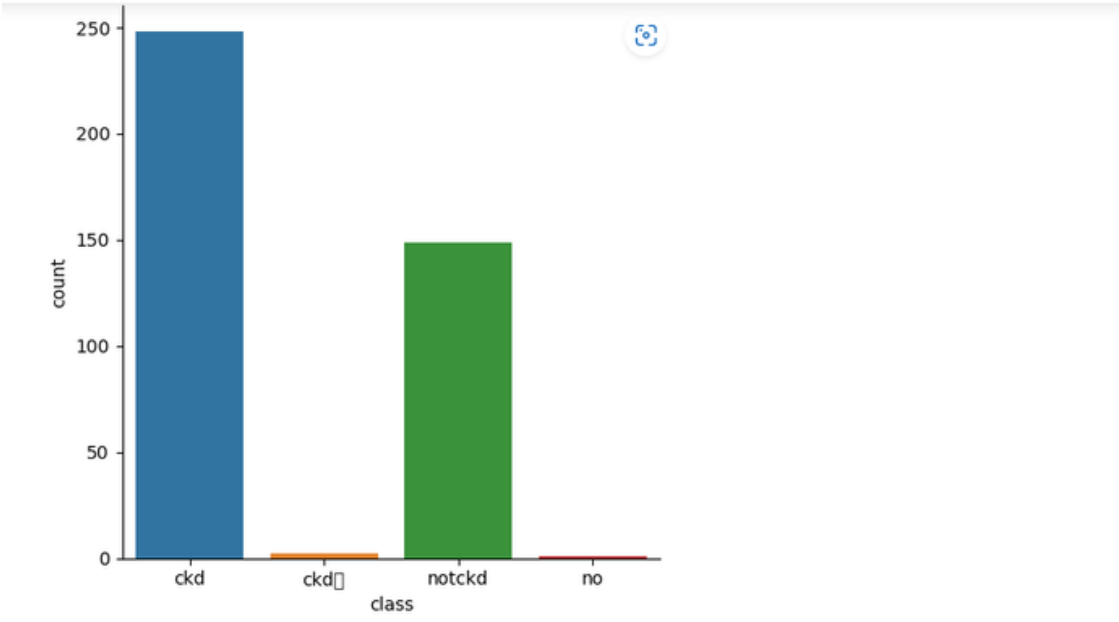
```
unique values in "age":
 ['48' '7' '62' '51' '60' '68' '24' '52' '53' '50' '63' '40' '47' '61' '21'
 '42' '75' '69' '?' '73' '70' '65' '76' '72' '82' '46' '45' '35' '54' '11'
 '59' '67' '15' '55' '44' '26' '64' '56' '5' '74' '38' '58' '71' '34' '17'
 '12' '43' '41' '57' '8' '39' '66' '81' '14' '27' '83' '30' '4' '3' '6'
 '32' '80' '49' '90' '78' '19' '2' '33' '36' '37' '23' '25' '20' '29' '28'
 '22' '79']
unique values in "bp":
 ['80' '50' '70' '90' '?' '100' '60' '110' '140' '180' '120']
unique values in "sg":
 ['1.020' '1.010' '1.005' '1.015' '?' '1.025']
unique values in "al":
 ['1' '4' '2' '3' '0' '?' '5']
unique values in "su":
 ['0' '3' '4' '1' '?' '2' '5']
unique values in "rbc":
 ['?' 'normal' 'abnormal']
unique values in "pc":
 ['normal' 'abnormal' '?']
unique values in "pcc":
 ['notpresent' 'present' '?']
unique values in "ba":
 ['notpresent' 'present' '?']
unique values in "bgr":
 ['121' '?' '423' '117' '106' '74' '100' '410' '138' '70' '490' '380' '208'
 '98' '157' '76' '99' '114' '263' '173' '95' '108' '156' '264' '123' '93'
 '107' '159' '140' '171' '270' '92' '137' '204' '79' '207' '124' '144'
```

Display informations about Dataframe

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 26 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   age      400 non-null    object
 1   bp       400 non-null    object
 2   sg       400 non-null    object
 3   al       400 non-null    object
 4   su       400 non-null    object
 5   rbc      400 non-null    object
 6   pc       400 non-null    object
 7   pcc      400 non-null    object
 8   ba       400 non-null    object
 9   bgr      400 non-null    object
 10  bu       400 non-null    object
 11  sc       400 non-null    object
 12  sod      400 non-null    object
 13  pot      400 non-null    object
 14  hemo     400 non-null    object
 15  pcv      400 non-null    object
 16  wbcc     400 non-null    object
 17  rbcc     400 non-null    object
 18  htn      400 non-null    object
 19  dm       400 non-null    object
 20  cad      400 non-null    object
 21  appet    400 non-null    object
 22  pe       400 non-null    object
 23  ane      400 non-null    object
 24  class    400 non-null    object
 25  no_name  3 non-null      object
dtypes: object(26)
```

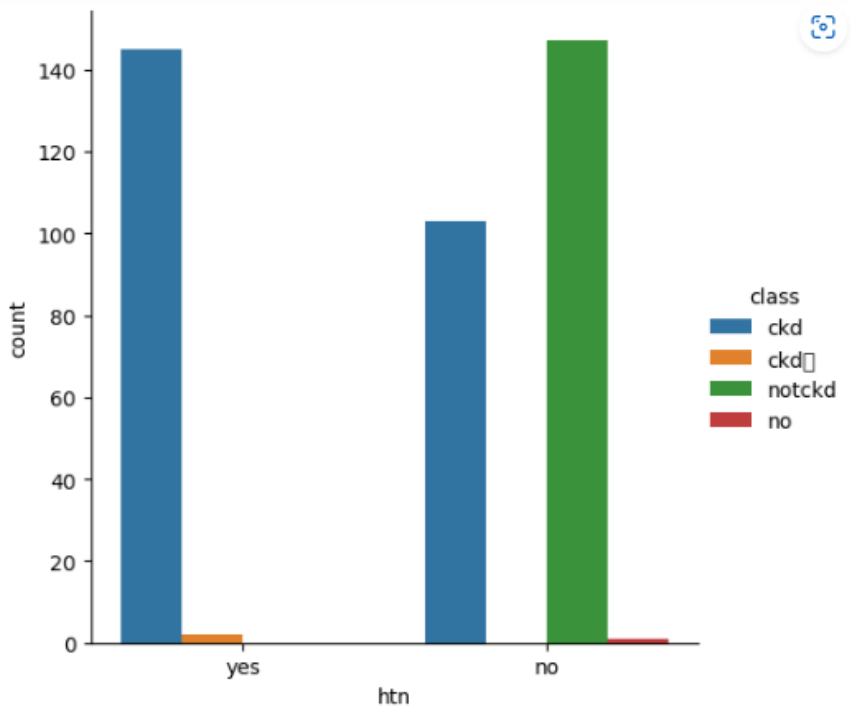
We notice that all variables are object while most of our features are reals and integers.



So we notice that our dataset is not balanced. 37.5% of observations have the value of class = 0 and 62.5% have a value = 1.

Explore the variation of each feature with the target variable

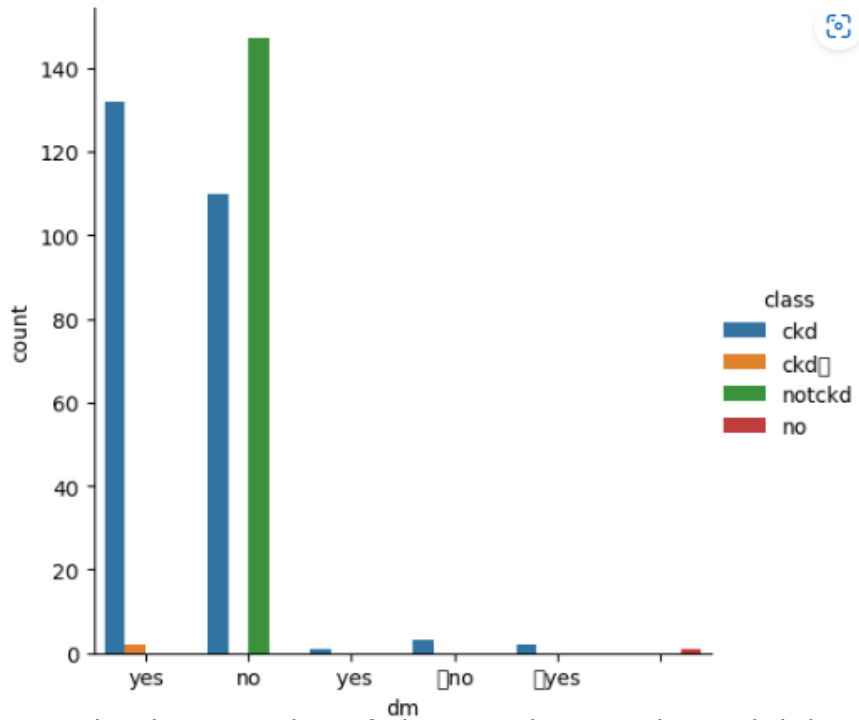
```
sns.catplot(x='htn',hue='class',data=df,kind="count")
```



we noticed that the value 'ckd' and 'notckd' for the class column was sometimes miswritten (by adding a space or writing 'no' only instead of notckd)

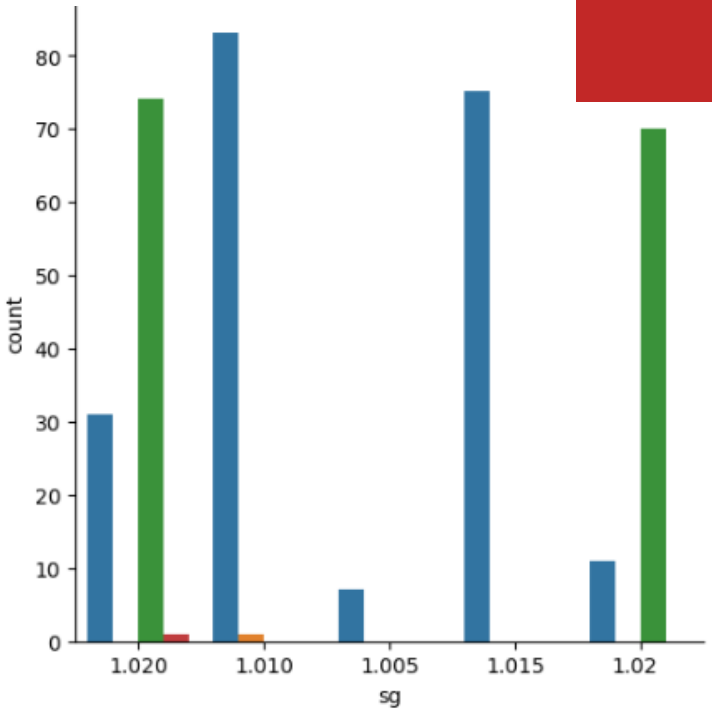
Most cases that have a value equal to 'yes' for the htn variable have ckd disease

```
sns.catplot(x='dm',hue='class',data=df,kind="count")
```

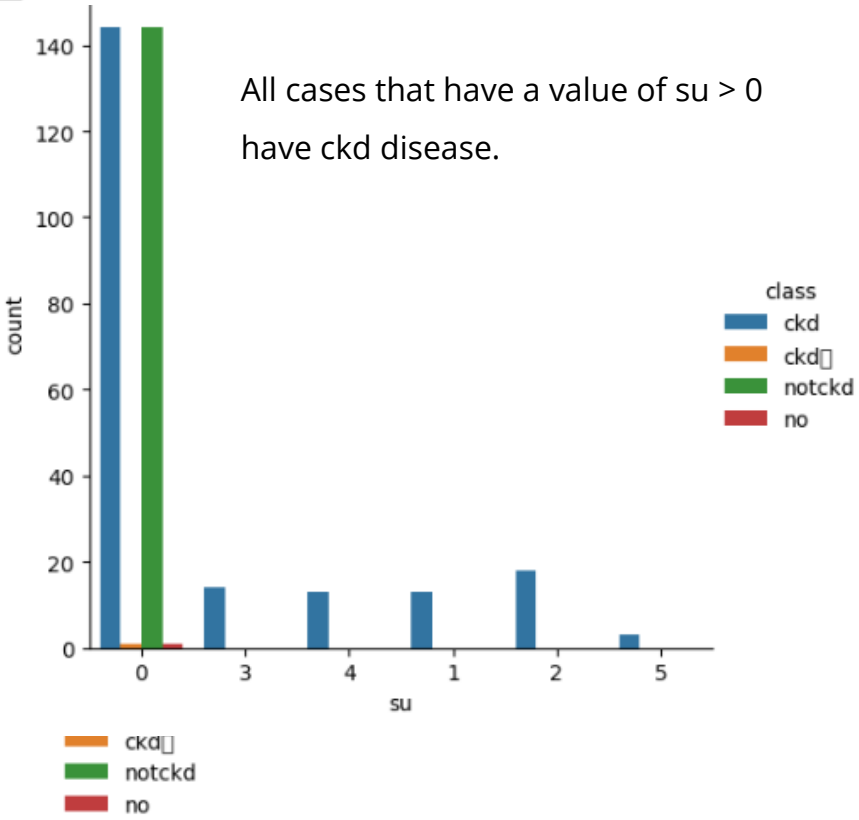


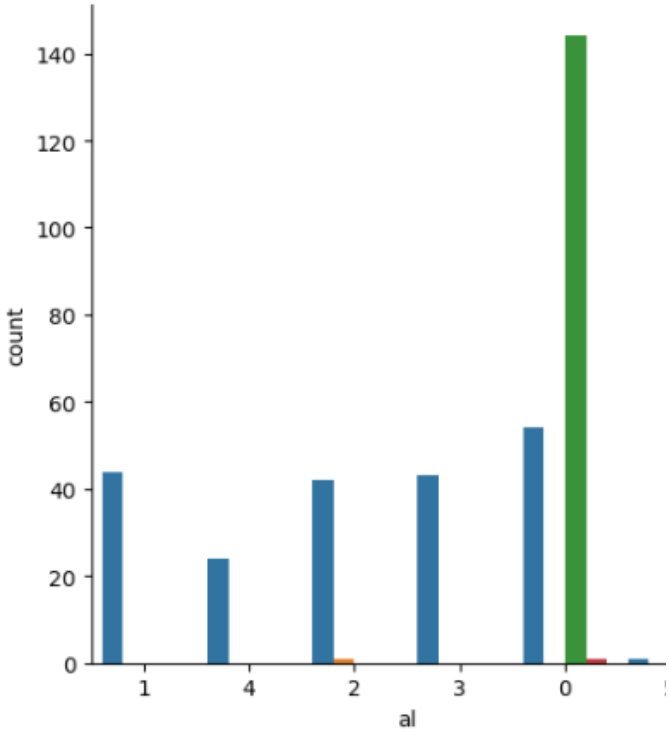
Most observations that have a value of 'dm' equal to 'yes' have ckd disease.

We also notice that 'yes' and 'no' are sometimes badly written when adding a space.

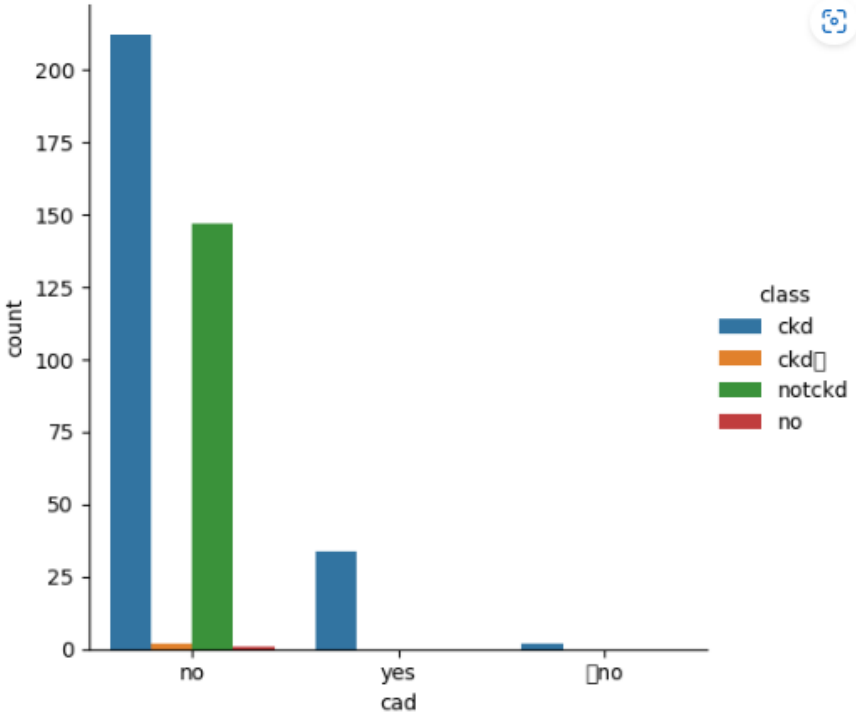


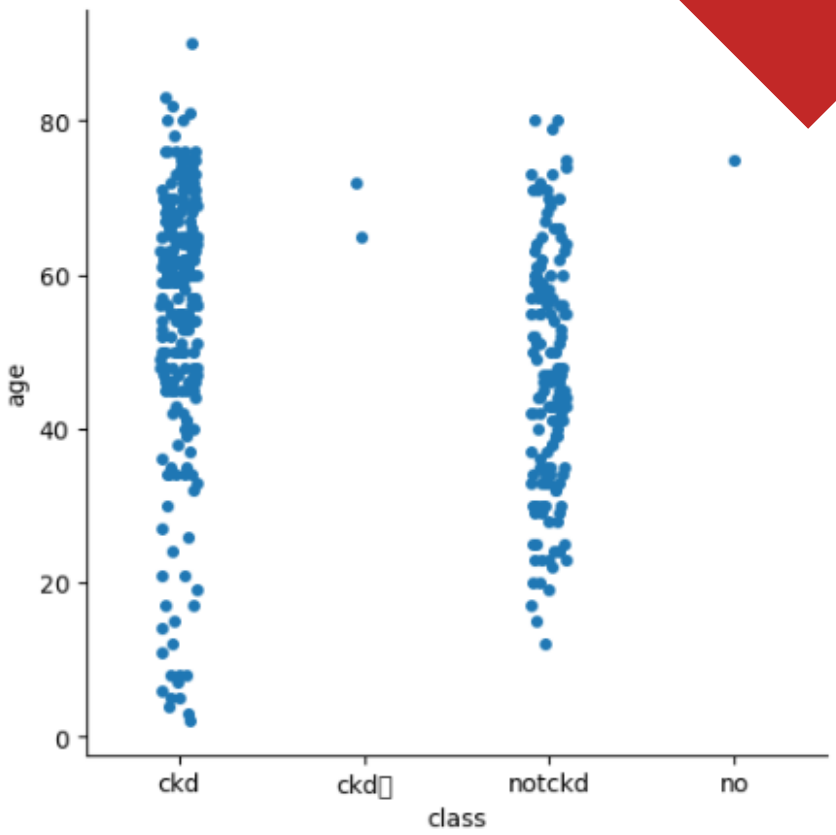
All cases that have a value of su > 0 have ckd disease.

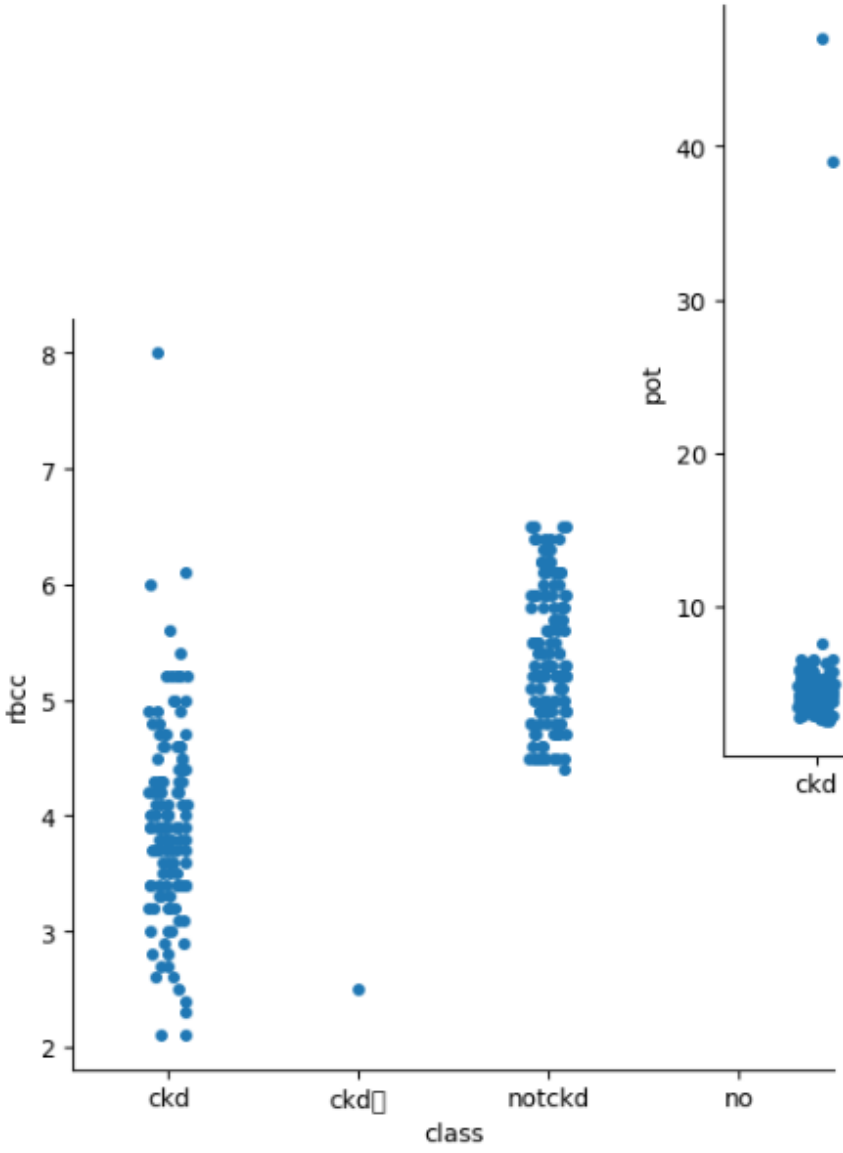


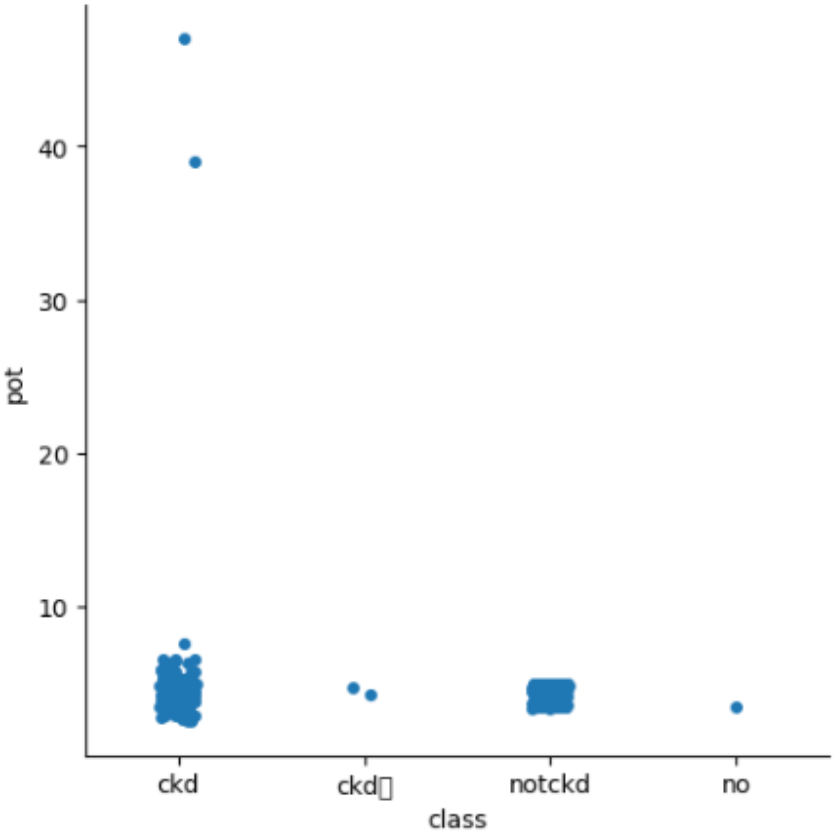


Most observations that do not have ckd disease have a value of 'al' equal to 0.



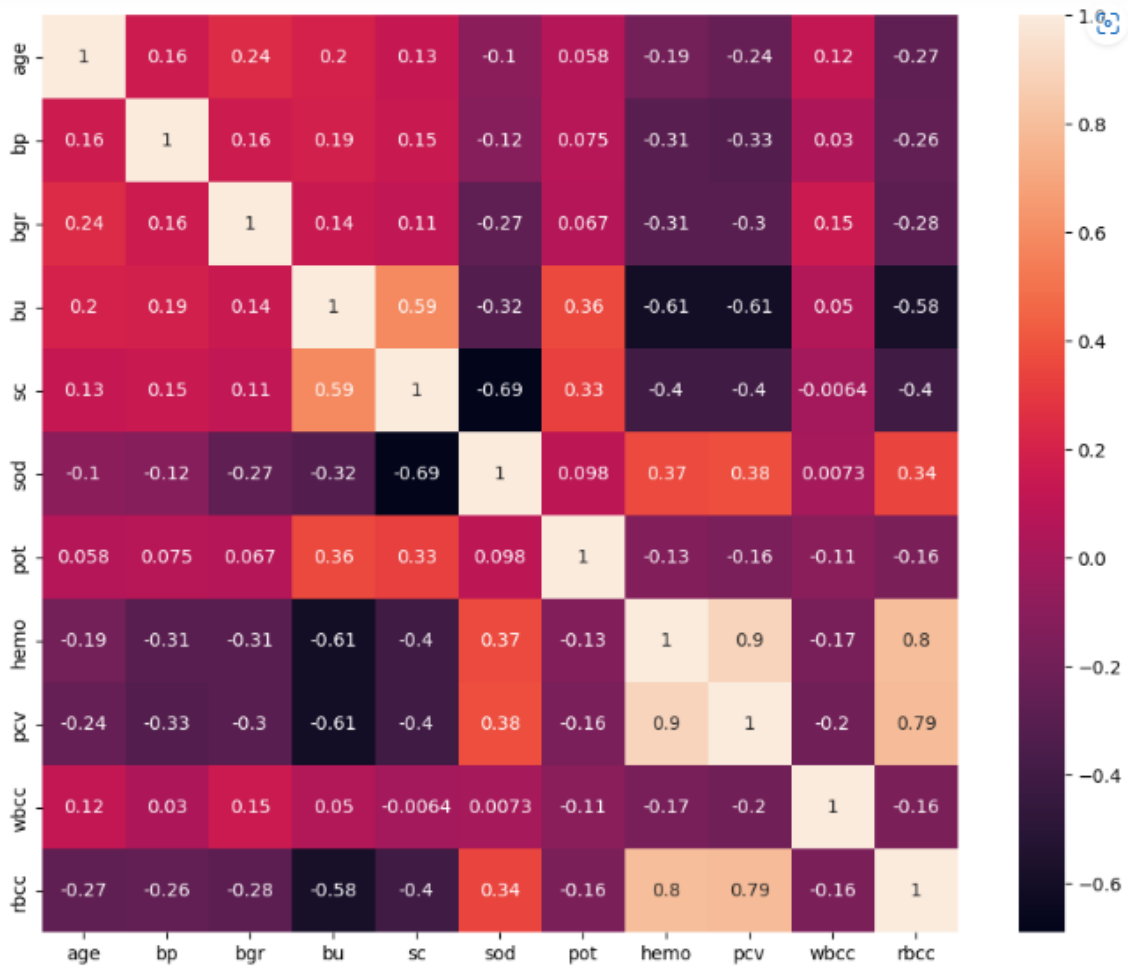

age
0
20
40
60
80
ckd
ckd
notckd
no
class


rbcc
2
3
4
5
6
7
8
ckd
ckd
notckd
no
class

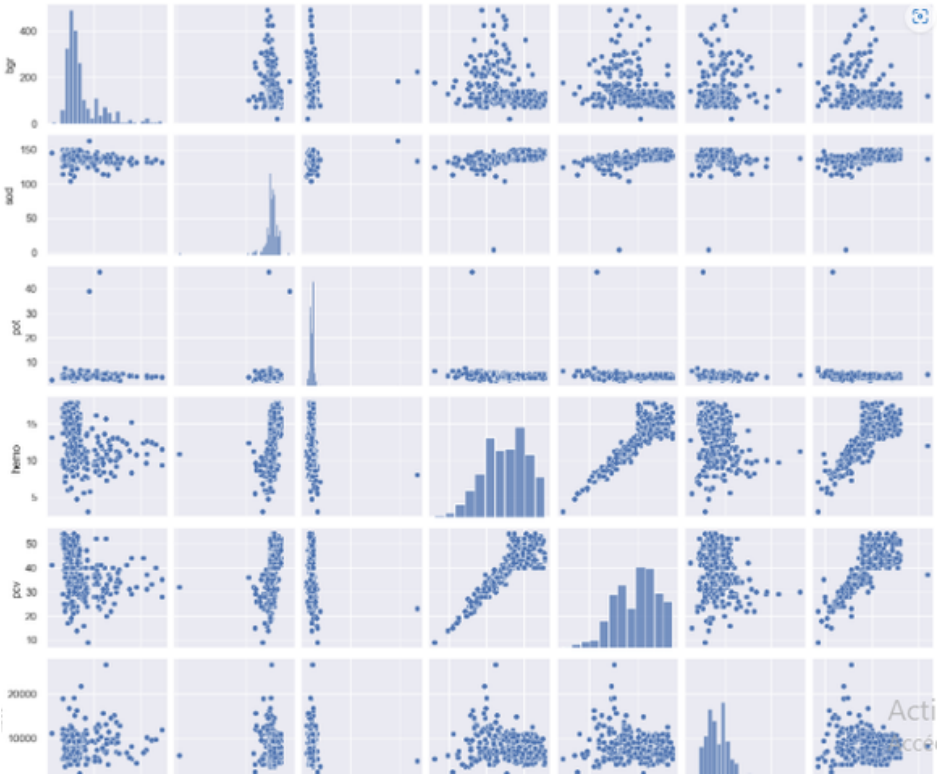

pot
10
20
30
40
ckd
ckd
notckd
no
class

Correlation matrix

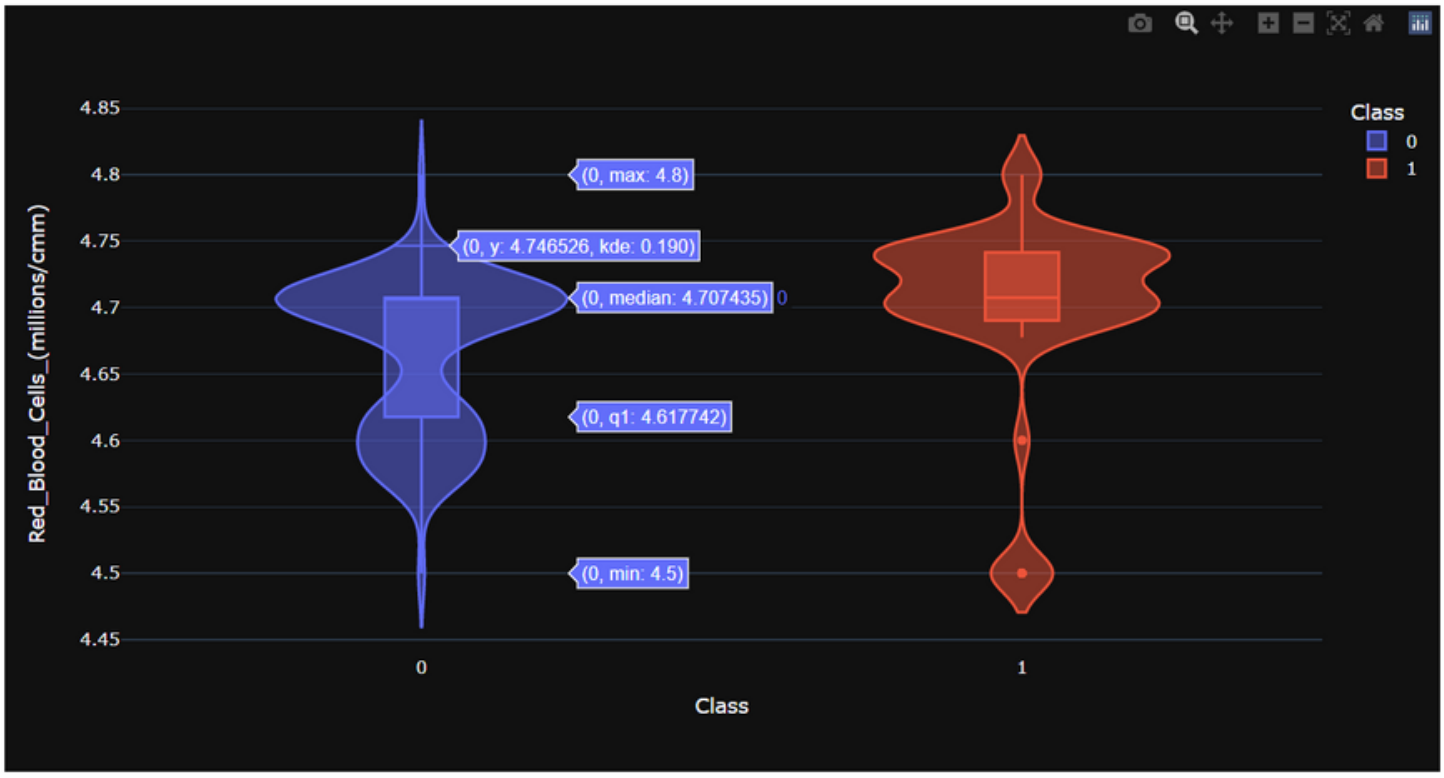
```
corrmat = df.corr()
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corrmat, vmax=1, square=True,annot=True);
```



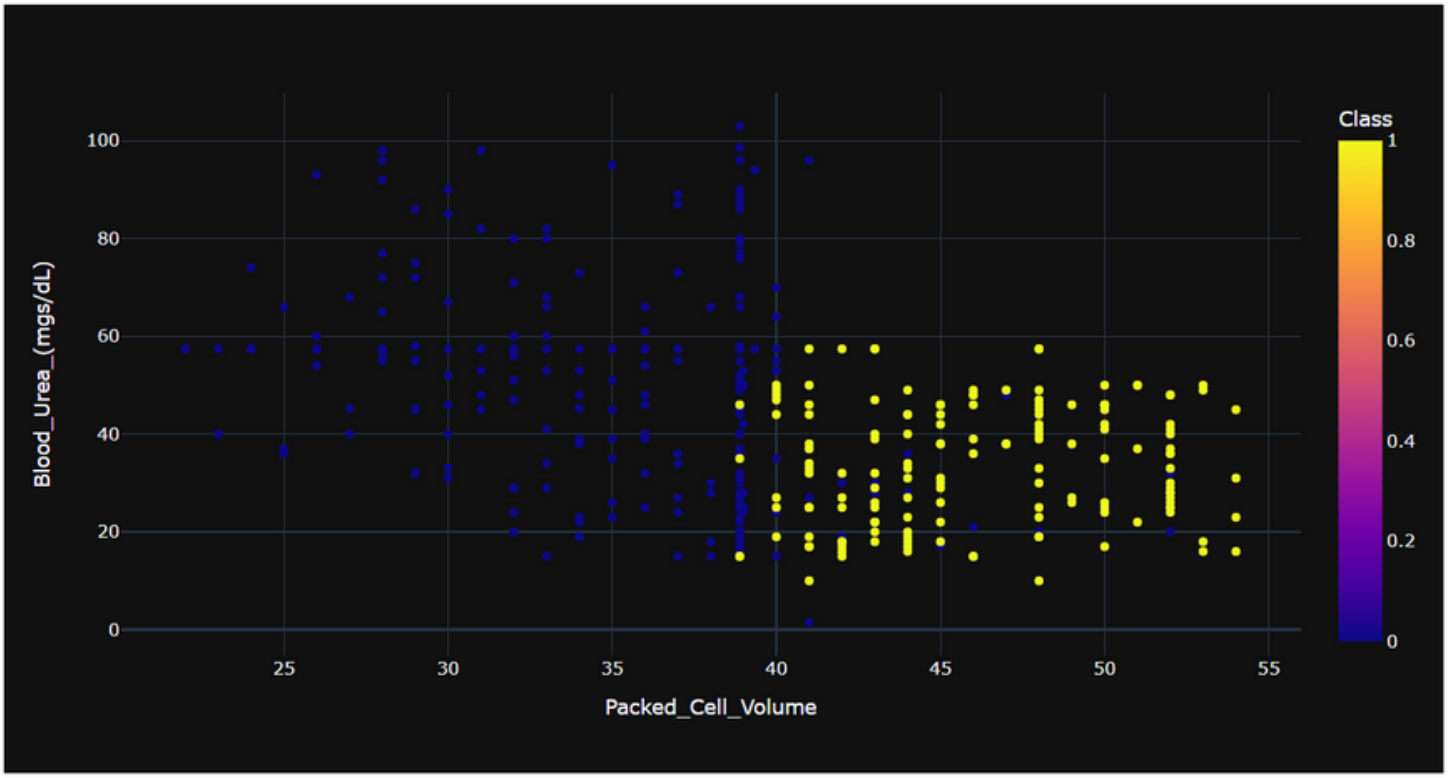
DATA VISUALISATION



```
violin('Red_Blood_Cells_(millions/cmm)')
```




```
scatter('Packed_Cell_Volume','Blood_Urea_(mgs/dL)')
```



III-Data Preparation :

Once the data has been collected, it must be transformed or preprocessed into a usable subset by checking for questionable, missing or ambiguous cases.

First, we noticed a lot of typing errors for example, in classification Column we have ckd\t so we have to convert it to 'ckd' and the same thing for 'dm' '\tyes' & ' yes' ,etc ..

Therefore, we started by fixing them and replacing them by nan values to not influence our dataset. Furthermore,when exploring the types of our columns using df.info() we noticed a lot of types errors, columns that are supposed to be floats are objects so we proceeded with fixing that

Fixing types error mistakes and typing errors

```
df["classification"]=df["classification"].apply(lambda x:'ckd' if x=="ckd\t" else x)

df["cad"]=df["cad"].apply(lambda x:'no' if x=="\tno" else x)

df["dm"]=df["dm"].apply(lambda x:'no' if x=="\tno" else x)
df["dm"]=df["dm"].apply(lambda x:'yes' if x=="\tyes" else x)
df["dm"]=df["dm"].apply(lambda x:'yes' if x==' yes' else x)
```

```
#fixing mistakes and converting pcv to numerical
df["pcv"]=df["pcv"].apply(lambda x:np.nan if x=="\t?" else x)
df["pcv"]=df["pcv"].apply(lambda x: float(x))
```

```
#fixing mistakes and converting wc to numerical
df["wc"]=df["wc"].apply(lambda x:'6200' if x=="\t6200" else x)
df["wc"]=df["wc"].apply(lambda x:'8400' if x=="\t8400" else x)
df["wc"]=df["wc"].apply(lambda x:np.nan if x=="\t?" else x)
df["wc"]=df["wc"].apply(lambda x: float(x))
```

```
#fixing mistakes and converting rc to numerical
df["rc"]=df["rc"].apply(lambda x:np.nan if x=="\t?" else x)
df["rc"]=df["rc"].apply(lambda x: float(x))
```

Afterwords, we replaced the numerical missing values with the "mean" strategy and categorical missing values with the mode.

Let's replace the missing values of the numeric variables with the mean.

```
NumericCols=["age","bp","bgr","bu","sc","sod","pot","hemo","pcv","wbcc","rbcc"]
for col in NumericCols:
    df[col].fillna(value=df[col].mean(), inplace=True)
```

Let's replace the missing values of the categorical variables with the mode.

```
df = df.fillna(df.mode().iloc[0])
df
```

```
df.isna().sum().sum()
```

```
0
```

outliers (irréelles)

For pot examples:

```
df.query("pot>30")
```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	wbcc	rbcc	htn	dm	cad	appet	pe	ane	class	no_name
61	67.0	80.0	1.010	1	3	normal	abnormal	notpresent	notpresent	182.0	...	NaN	NaN	no	no	no	good	yes	no	ckd	None
128	52.0	90.0	1.015	4	3	normal	abnormal	notpresent	notpresent	224.0	...	5000.0	2.9	yes	yes	no	good	no	yes	ckd	None

2 rows × 26 columns

```
df[61:62][['pot']]=df['pot'].mean()
df[128:129][['pot']]=df['pot'].mean()
```

Verification that values have been changed:

```
print(df[128:129][['pot']])
print(df[61:62][['pot']])
```

```
      pot
128  47.0
      pot
61   39.0
```

For the example of the variable 'rbcc':

```
df.query("rbcc>7")
```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	wbcc	rbcc	htn	dm	cad	appet	pe	ane	class	no_name
133	70.0	100.0	1.015	4	0	normal	normal	notpresent	notpresent	118.0	...	8400.0	8.0	yes	no	no	good	no	no	ckd	None

1 rows × 26 columns

```
df[133:134][['rbcc']]=df['rbcc'].mean()
```

Verification that values have been changed:

```
df[133:134][['rbcc']]
```

	rbcc
133	8.0

Then, we converted the categorical data to numerical using label encoding, a very simple approach that transfers each value in a column to a number.

Type conversion of categorical variables that have some real values and encoding of the rest.

```
df["su"]=df["su"].astype('float')
df["al"]=df["al"].astype('float')
df["sg"]=df["sg"].astype('float')
```

```
Yes_No_Cols=["ane","pe","cad","dm","htn"]
for col in Yes_No_Cols:
    df[col] = df[col].map({'yes': 1,'\tyes': 1,' yes':1, 'no': 0, '\tno':0})
```

```
df["class"] = df["class"].map({'ckd': 1,'ckd\t':1, 'notckd': 0})
```

```
df["appet"] = df["appet"].map({'good': 1, 'poor': 0})
```

```
df["rbc"] = df["rbc"].map({'normal': 1,'abnormal': 0})
df["pc"] = df["pc"].map({'normal': 1,'abnormal': 0})
```

```
df["pcc"] = df["pcc"].map({'present': 1,'notpresent': 0})
df["ba"] = df["ba"].map({'present': 1,'notpresent': 0})
```

Another technique we used in data preparation is feature scaling to allow the model to process the samples of numerical features with a normalized range of values.

Normalisation

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
NumericCols=["age","bp","bgr","bu","sc","sod","pot","hemo","pcv","wbcc","rbcc"]
df[NumericCols] = sc.fit_transform(df[NumericCols])
```

```
df
```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wbcc	rbcc	htn	dm	cad	appet	pe	ane	class
0	-0.205464	0.262338	1.020	1.0	0.0	1	1	0	0	-0.361987	...	0.628372	-0.240519	5.869017e-01	1	1	0	1	0	0	1
1	-2.623810	-1.966580	1.020	4.0	0.0	1	1	0	0	0.000000	...	-0.108649	-0.954786	1.058283e-15	0	0	0	1	0	0	1
2	0.620313	0.262338	1.010	2.0	3.0	1	1	0	0	3.681441	...	-0.968506	-0.359563	1.058283e-15	0	1	0	0	0	1	1
3	-0.205464	-0.480635	1.005	4.0	0.0	1	0	1	0	-0.415543	...	-0.845669	-0.677016	-9.620759e-01	1	0	0	0	1	1	1
4	-0.028511	0.262338	1.010	2.0	0.0	1	1	0	0	-0.562820	...	-0.477159	-0.438926	-1.280110e-01	0	0	0	1	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
395	0.207425	0.262338	1.020	0.0	0.0	1	1	0	0	-0.107600	...	0.996882	-0.677016	2.294454e-01	0	0	0	1	0	0	0
396	-0.559368	-0.480635	1.025	0.0	0.0	1	1	0	0	-0.977874	...	1.856739	-0.240519	1.778423e+00	0	0	0	1	0	0	0
397	-2.328890	0.262338	1.020	0.0	0.0	1	1	0	0	-0.643153	...	1.242555	-0.716697	8.252060e-01	0	0	0	1	0	0	0

1-Article 1 :

In this project, we used the RFE method to extract the most important features of a prediction. Recursive Feature Elimination (RFE) algorithm is very popular due to its ease of use and configurations and its effectiveness in selecting features in training datasets relevant to predicting target variables and eliminating weak features.

```
RFE :
Column: age, Selected=True, Rank: 1
Column: bp, Selected=False, Rank: 6
Column: sg, Selected=True, Rank: 1
Column: al, Selected=True, Rank: 1
Column: su, Selected=False, Rank: 3
Column: rbc, Selected=False, Rank: 8
Column: pc, Selected=False, Rank: 7
Column: pcc, Selected=False, Rank: 11
Column: ba, Selected=False, Rank: 12
Column: bgr, Selected=True, Rank: 1
Column: bu, Selected=True, Rank: 1
Column: sc, Selected=True, Rank: 1
Column: sod, Selected=True, Rank: 1
Column: pot, Selected=False, Rank: 5
Column: hemo, Selected=True, Rank: 1
Column: pcv, Selected=True, Rank: 1
Column: wc, Selected=False, Rank: 9
Column: rc, Selected=True, Rank: 1
Column: htn, Selected=True, Rank: 1
Column: dm, Selected=True, Rank: 1
Column: cad, Selected=False, Rank: 13
Column: appet, Selected=False, Rank: 2
Column: pe, Selected=False, Rank: 4
Column: ane, Selected=False, Rank: 10
################
```

```
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import RFECV

# define RFE
##rfe = RFE(estimator=SVR(kernel="linear"), n_features_to_select=10)
rfe =RFE(estimator=RandomForestClassifier() , n_features_to_select=12)

# define RFECV
rfecv =RFECV(estimator=RandomForestClassifier() , min_features_to_select=12)

# fit RFE
rfe = rfe.fit(x, y)

# fit RFECV
rfecv = rfecv.fit(x,y)

# summarize all features
print ('RFE : ')
for i in range(x.shape[1]):
  print('Column: %s, Selected=%s, Rank: %d' % (list(df.columns)[i], rfe.suppor

print('################')

print ('RFECV : ')
# summarize all features
for i in range(x.shape[1]):
  print('Column: %s, Selected=%s, Rank: %d' % (list(df.columns)[i], rfecv.supp
```

2-Article 2 :

Correlation-based Feature Selection (CFS) is suitable to be applied to multivariate data.

CFS works by calculating the interaction between feature

Correlation-based feature selection :

```
features = df.columns.tolist()
features.remove('class')
features
```

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

```
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)
```

```
best_value = -1
best_feature = ''
for feature in features:
    coeff = pointbiserialr( df['class'], 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 hemo with merit 0.7296
```

```
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)
```

```
# initialize queue
queue = PriorityQueue()

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

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

# counter for backtracks
n_backtrack = 0

# limit of backtracks
max_backtrack = 5
```

```
# 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, 'class')
        # and push it to the queue
        queue.push(temp_subset, merit)
```

```
print(best_subset)
```

```
['hemo', 'sg', 'dm', 'pcv', 'al', 'htn']
```

```
df1=df[list(best_subset)+['class']]
```

```
df1
```

	hemo	sg	dm	pcv	al	htn	class
0	1.059271	1.020	1	0.628372	1.0	1	1
1	-0.452097	1.020	0	-0.108649	4.0	0	1
2	-1.078762	1.010	1	-0.968506	2.0	0	1
3	-0.488960	1.005	0	-0.845669	4.0	1	1
4	-0.341509	1.010	0	-0.477159	2.0	0	1
...	...	...	...	...	...	...	...
395	1.169859	1.020	0	0.996882	0.0	0	0
396	1.464760	1.025	0	1.856739	0.0	0	0
397	1.206722	1.020	0	1.242555	0.0	0	0
398	0.616920	1.025	0	1.488229	0.0	0	0
399	1.206722	1.025	0	1.733902	0.0	0	0

400 rows × 7 columns

IV-Modeling :

Oversampling is used when the amount of data collected is insufficient.

SMOTE (synthetic minority oversampling technique) is one of the most commonly used oversampling methods to solve the imbalance

Oversampling

```
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
```

```
os = SMOTE(random_state=0)
X = df.loc[:, df.columns != 'class']
y = df.loc[:, df.columns == 'class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
columns = X_train.columns
os_data_X,os_data_y=os.fit_resample(X_train, y_train)
os_data_X = pd.DataFrame(data=os_data_X,columns=columns ) #contain columns
os_data_y= pd.DataFrame(data=os_data_y,columns=['class']) #contain class
```

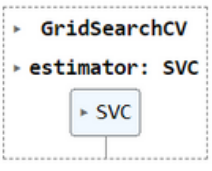
```
os_data_X1=os_data_X
os_data_X=os_data_X[list(best_subset)]
X_test = pd.DataFrame(data=X_test,columns=columns)
X_test1=X_test
X_test=X_test[list(best_subset)]
```

1-Article 1 : :

Modeling Without feature selection

SVM

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV

# defining parameter range
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf','sigmoid','poly','linear']}

grid = GridSearchCV(SVC(), param_grid, refit = True)

# fitting the model for grid search
grid.fit(os_data_X1,  os_data_y.values.ravel())
```




```
best_parameters = grid.best_params_
best_parameters
```

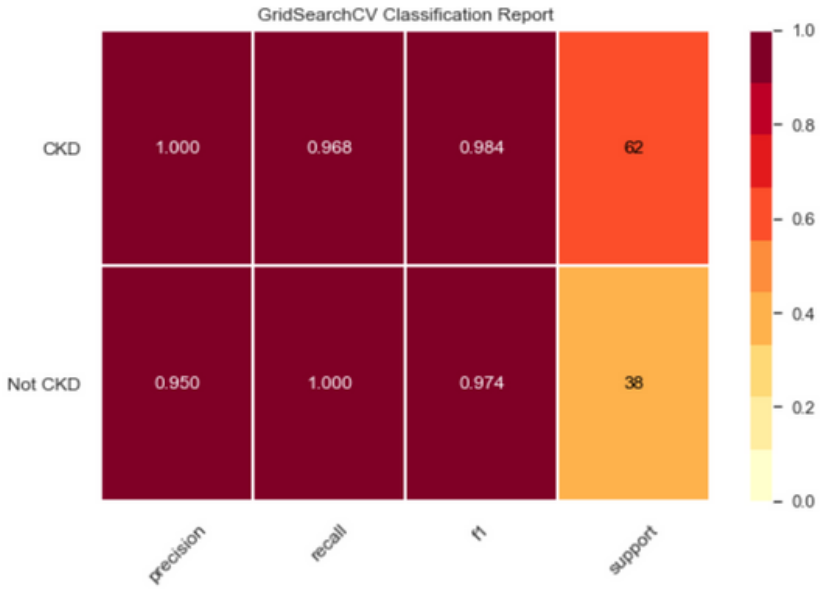
```
{'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
```

```
# make predictions on test set
y_pred1=grid.predict(X_test1)
```

```
# compute and print accuracy score
print('SVM model accuracy score with default hyperparameters: {0:0.4f}'. forma
```

```
SVM model accuracy score with default hyperparameters: 0.9800
```

```
from sklearn.model_selection import TimeSeriesSplit
import yellowbrick
from yellowbrick.classifier import ClassificationReport
from yellowbrick.datasets import load_occupancy

# Specify the target classes
classes = ["Not CKD", "CKD"]

visualizer = ClassificationReport(grid, classes=classes, support=True)
visualizer.fit(os_data_X1, os_data_y)
visualizer.score(X_test1, y_test)    # Evaluate the model on the test data
visualizer.show()
```



KNN

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
leaf_size = list(range(2,20))# Par défaut c'est 30
n_neighbors = list(range(1,25))
p=[1,2] #p = 1 manhattan_distance/p=2 euclidean_distance
#Convert to dictionary
hyperparameters = dict(leaf_size=leaf_size, n_neighbors=n_neighbors, p=p)
#Create new KNN object
knn_2 = KNeighborsClassifier()
#Use GridSearch
clf = GridSearchCV(knn_2, hyperparameters, cv=10, n_jobs=-1)
#n_jobs: number of CPU'sfor execution.
#cv: number of folds of the cross validation

#Fit the model
best_model = clf.fit(os_data_X1, os_data_y.values.ravel())
#Print The value of best Hyperparameters
print('Best leaf_size:', best_model.best_estimator_.get_params()['leaf_size'])
print('Best p:', best_model.best_estimator_.get_params()['p'])
print('Best n_neighbors:', best_model.best_estimator_.get_params()['n_neighbor
```

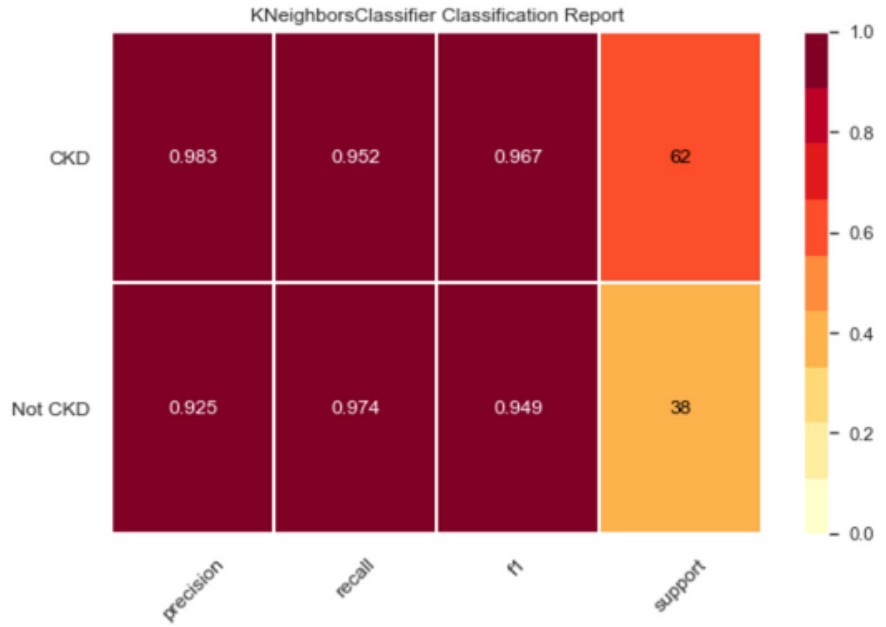
```
Best leaf_size: 2
Best p: 2
Best n_neighbors: 1
```

```
knn = KNeighborsClassifier(n_neighbors=1,p=2,leaf_size=2)
knn_model = knn.fit(os_data_X1, os_data_y.values.ravel())
y_pred_knn = knn_model.predict(X_test1)
```

```
print('Accuracy of K-NN classifier on training set: {:.2f}'
     .format(knn.score(os_data_X1, os_data_y)))
print('Accuracy of K-NN classifier on test set: {:.2f}'
     .format(knn.score(X_test1, y_test)))
```

```
Accuracy of K-NN classifier on training set: 1.00
Accuracy of K-NN classifier on test set: 0.96
```

```
from sklearn.model_selection import TimeSeriesSplit
import yellowbrick
from yellowbrick.classifier import ClassificationReport
from yellowbrick.datasets import load_occupancy

# Specify the target classes
classes = ["Not CKD", "CKD"]

visualizer = ClassificationReport(knn_model, classes=classes, support=True)
visualizer.fit(os_data_X1, os_data_y)
visualizer.score(X_test1, y_test)    # Evaluate the model on the test data
visualizer.show()
```



Decision tree

```
from sklearn.tree import DecisionTreeClassifier
modeldt =  DecisionTreeClassifier()
```

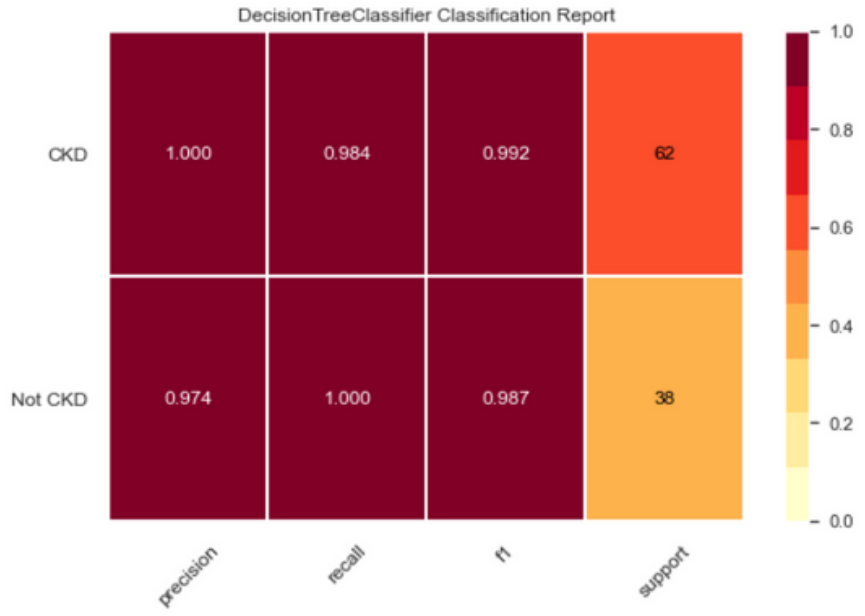
```
modeldt.fit(os_data_X1, os_data_y.values.ravel())
```

```
DecisionTreeClassifier
DecisionTreeClassifier()
```

```
from sklearn.metrics import classification_report
y_preddt = modeldt.predict(X_test1)
y_preddt
print(classification_report(y_test, y_preddt))
```

```
              precision    recall  f1-score   support

           0       0.97      1.00      0.99        38
           1       1.00      0.98      0.99        62

    accuracy                           0.99       100
   macro avg       0.99      0.99      0.99       100
weighted avg       0.99      0.99      0.99       100
```

```
from sklearn.model_selection import TimeSeriesSplit
import yellowbrick
from yellowbrick.classifier import ClassificationReport
from yellowbrick.datasets import load_occupancy

# Specify the target classes
classes = ["Not CKD", "CKD"]

visualizer = ClassificationReport(modeldt, classes=classes, support=True)
visualizer.fit(os_data_X1, os_data_y)
visualizer.score(X_test1, y_test)    # Evaluate the model on the test data
visualizer.show()
```



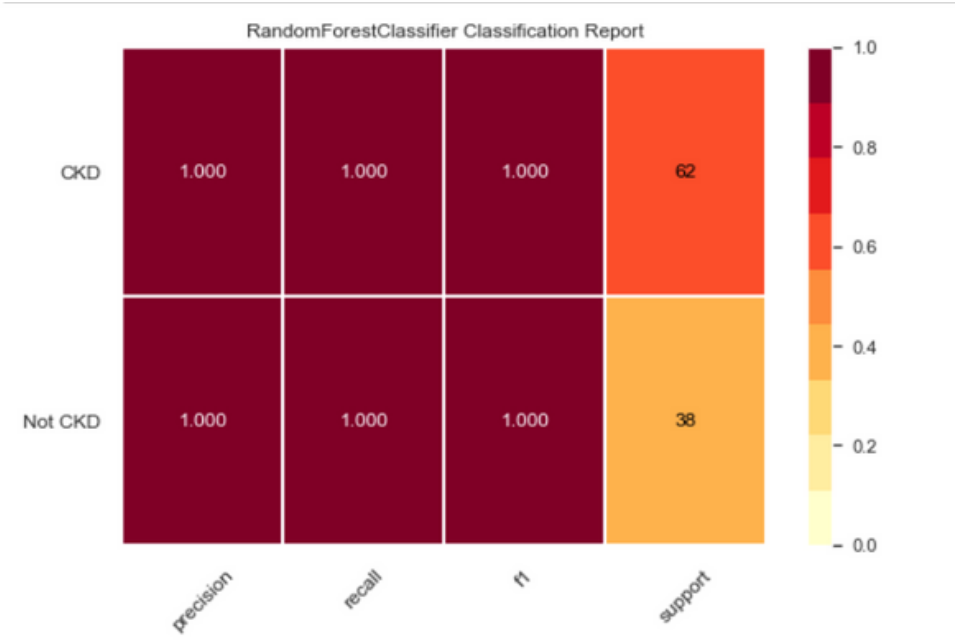
Random forrest

```
#Import Random Forest Model
from sklearn.ensemble import RandomForestClassifier

#Create a Gaussian Classifier
clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(os_data_X1, os_data_y.values.ravel())

y_pred=clf.predict(X_test1)
```

```
print(classification_report(y_test, y_preddt))
```

```
              precision    recall  f1-score   support

           0       0.97      1.00      0.99        38
           1       1.00      0.98      0.99        62

    accuracy                           0.99       100
   macro avg       0.99      0.99      0.99       100
weighted avg       0.99      0.99      0.99       100
```

```
from sklearn.model_selection import TimeSeriesSplit
import yellowbrick
from yellowbrick.classifier import ClassificationReport
from yellowbrick.datasets import load_occupancy

# Specify the target classes
classes = ["Not CKD", "CKD"]

visualizer = ClassificationReport(clf, classes=classes, support=True)
visualizer.fit(os_data_X1, os_data_y)
visualizer.score(X_test1, y_test)    # Evaluate the model on the test data
visualizer.show()
```



Modeling with feature selection

KNN

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
leaf_size = list(range(2,20))# Par défaut c'est 30
n_neighbors = list(range(1,25))
p=[1,2] #p = 1 manhattan_distance/p=2 euclidean_distance
#Convert to dictionary
hyperparameters = dict(leaf_size=leaf_size, n_neighbors=n_neighbors, p=p)
#Create new KNN object
knn_2 = KNeighborsClassifier()
#Use GridSearch
clf = GridSearchCV(knn_2, hyperparameters, cv=10, n_jobs=-1)
#n_jobs: number of CPU'sfor execution.
#cv: number of folds of the cross validation

#Fit the model
best_model = clf.fit(os_data_X, os_data_y.values.ravel())
#Print The value of best Hyperparameters
print('Best leaf_size:', best_model.best_estimator_.get_params()['leaf_size'])
print('Best p:', best_model.best_estimator_.get_params()['p'])
print('Best n_neighbors:', best_model.best_estimator_.get_params()['n_neighbor
```

```
Best leaf_size: 2
Best p: 1
Best n_neighbors: 1
```

```
# Afficher les paramètres qui donnent les meilleurs performances
best_parameters = clf.best_params_
print(best_parameters)
```

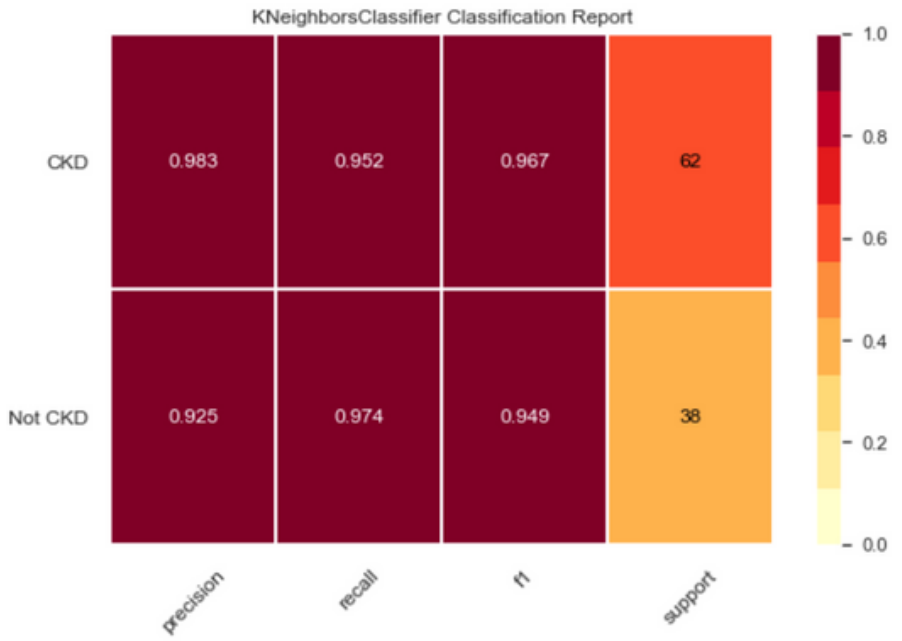
```
{'leaf_size': 2, 'n_neighbors': 1, 'p': 1}
```

```
knn = KNeighborsClassifier(n_neighbors=5,p=2,leaf_size=2 )
knn_model = knn.fit(os_data_X, os_data_y.values.ravel())
y_pred_knn = knn_model.predict(X_test)
```

```
print('Accuracy of K-NN classifier on training set: {:.2f}'
     .format(knn.score(os_data_X, os_data_y)))
print('Accuracy of K-NN classifier on test set: {:.2f}'
     .format(knn.score(X_test, y_test)))
```

```
Accuracy of K-NN classifier on training set: 0.98
Accuracy of K-NN classifier on test set: 0.96
```

```
from sklearn.model_selection import TimeSeriesSplit
import yellowbrick
from yellowbrick.classifier import ClassificationReport
from yellowbrick.datasets import load_occupancy

# Specify the target classes
classes = ["Not CKD", "CKD"]

visualizer = ClassificationReport(knn_model, classes=classes, support=True)
visualizer.fit(os_data_X, os_data_y)
visualizer.score(X_test, y_test)    # Evaluate the model on the test data
visualizer.show()
```



svm

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV

# defining parameter range
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf','sigmoid','poly','linear']}

grid = GridSearchCV(SVC(), param_grid, refit = True)

# fitting the model for grid search
grid.fit(os_data_X,  os_data_y.values.ravel())
```

GridSearchCV
estimator: SVC
SVC

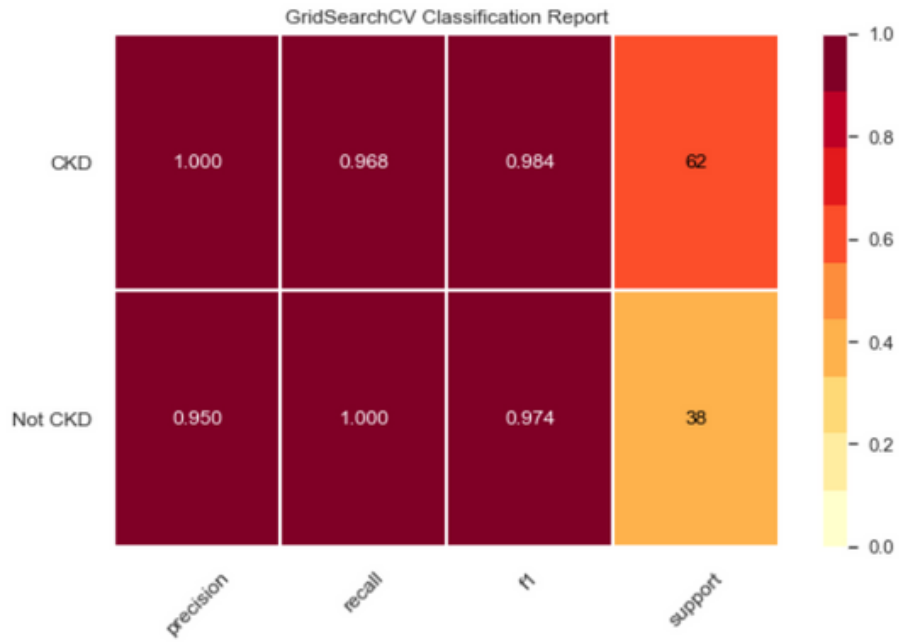
```
# make predictions on test set
y_pred1=grid.predict(X_test)
```

```
# compute and print accuracy score
print('SVM model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pr
```

```
SVM model accuracy score: 0.9800
```

```
from sklearn.model_selection import TimeSeriesSplit
import yellowbrick
from yellowbrick.classifier import ClassificationReport
from yellowbrick.datasets import load_occupancy

# Specify the target classes
classes = ["Not CKD", "CKD"]

visualizer = ClassificationReport(grid, classes=classes, support=True)
visualizer.fit(os_data_X, os_data_y)
visualizer.score(X_test, y_test)    # Evaluate the model on the test data
visualizer.show()
```

```
from sklearn.model_selection import TimeSeriesSplit
import yellowbrick
from yellowbrick.classifier import ClassificationReport
from yellowbrick.datasets import load_occupancy

# Specify the target classes
classes = ["Not CKD", "CKD"]

visualizer = ClassificationReport(grid, classes=classes, support=True)
visualizer.fit(os_data_X, os_data_y)
visualizer.score(X_test, y_test)    # Evaluate the model on the test data
visualizer.show()
```



Decision tree

```
from sklearn.tree import DecisionTreeClassifier
modeldt =  DecisionTreeClassifier()
```

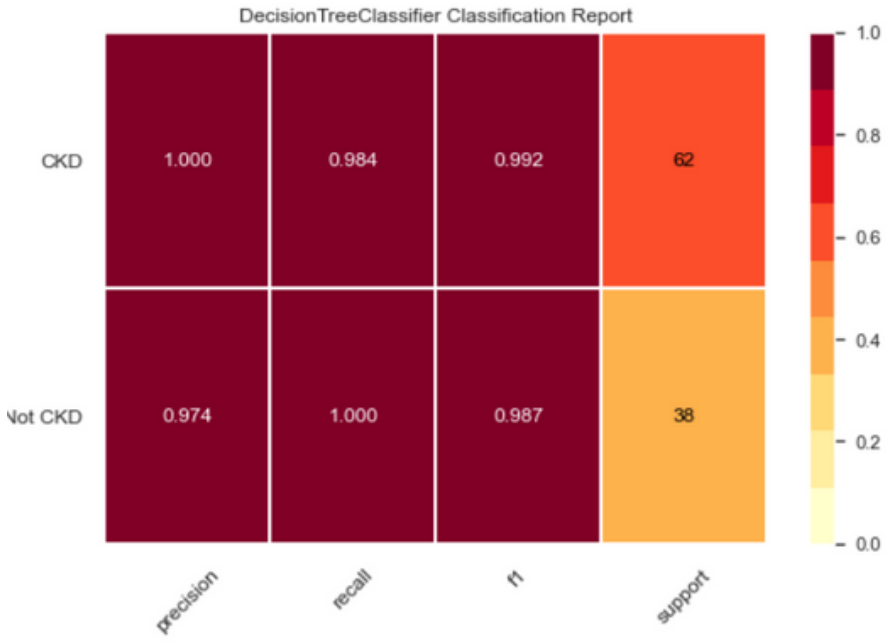
```
modeldt.fit(os_data_X, os_data_y.values.ravel())
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
from sklearn.metrics import classification_report
y_preddt = modeldt.predict(X_test)
y_preddt
print(classification_report(y_test, y_preddt))
```

```
              precision    recall  f1-score   support

           0       0.97      1.00      0.99        38
           1       1.00      0.98      0.99        62

    accuracy                           0.99       100
   macro avg       0.99      0.99      0.99       100
weighted avg       0.99      0.99      0.99       100
```

```
from sklearn.model_selection import TimeSeriesSplit
import yellowbrick
from yellowbrick.classifier import ClassificationReport
from yellowbrick.datasets import load_occupancy

# Specify the target classes
classes = ["Not CKD", "CKD"]

visualizer = ClassificationReport(modeldt, classes=classes, support=True)
visualizer.fit(os_data_X, os_data_y)
visualizer.score(X_test, y_test)    # Evaluate the model on the test data
visualizer.show()
```



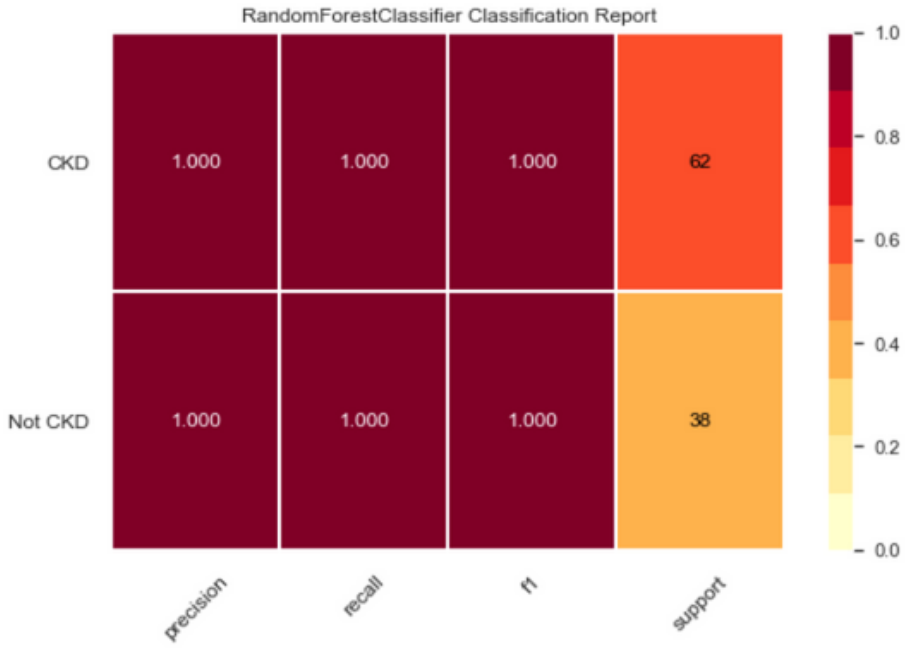
Random forest

```
#Import Random Forest Model
from sklearn.ensemble import RandomForestClassifier

#Create a Gaussian Classifier
clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(os_data_X, os_data_y.values.ravel())

y_pred=clf.predict(X_test)
```

```
print(classification_report(y_test, y_preddt))
```

```
              precision    recall  f1-score   support

           0       0.97      1.00      0.99        38
           1       1.00      0.98      0.99        62

    accuracy                           0.99       100
   macro avg       0.99      0.99      0.99       100
weighted avg       0.99      0.99      0.99       100
```

```
from sklearn.model_selection import TimeSeriesSplit
import yellowbrick
from yellowbrick.classifier import ClassificationReport
from yellowbrick.datasets import load_occupancy

# Specify the target classes
classes = ["Not CKD", "CKD"]

visualizer = ClassificationReport(clf, classes=classes, support=True)
visualizer.fit(os_data_X, os_data_y)
visualizer.score(X_test, y_test)    # Evaluate the model on the test data
visualizer.show()
```



2-Article 2 :

Creating and testing the models (without feature selection)

KNN

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV

leaf_size = list(range(2,20))# Par défaut c'est 30 [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
n_neighbors = list(range(1,25))
p=[1,2] #p = 1 manhattan_distance/p=2 euclidean_distance
#Convert to dictionary
hyperparameters = dict(leaf_size=leaf_size, n_neighbors=n_neighbors, p=p)
#Create new KNN object
knn_2 = KNeighborsClassifier()
#Use GridSearch
clf = GridSearchCV(knn_2, hyperparameters, cv=10, n_jobs=-1)
#n_jobs: number of CPU'sfor execution.
#cv: number of folds of the cross validation

#Fit the model
best_model = clf.fit(os_data_X1, os_data_y.values.ravel())
#Print The value of best Hyperparameters
print('Best leaf_size:', best_model.best_estimator_.get_params()['leaf_size'])
print('Best p:', best_model.best_estimator_.get_params()['p'])
print('Best n_neighbors:', best_model.best_estimator_.get_params()['n_neighbors'])
```

```
Best leaf_size: 2
Best p: 1
Best n_neighbors: 2
```

```
# display parameters which give the best performances
best_parameters = clf.best_params_
print(best_parameters)
```

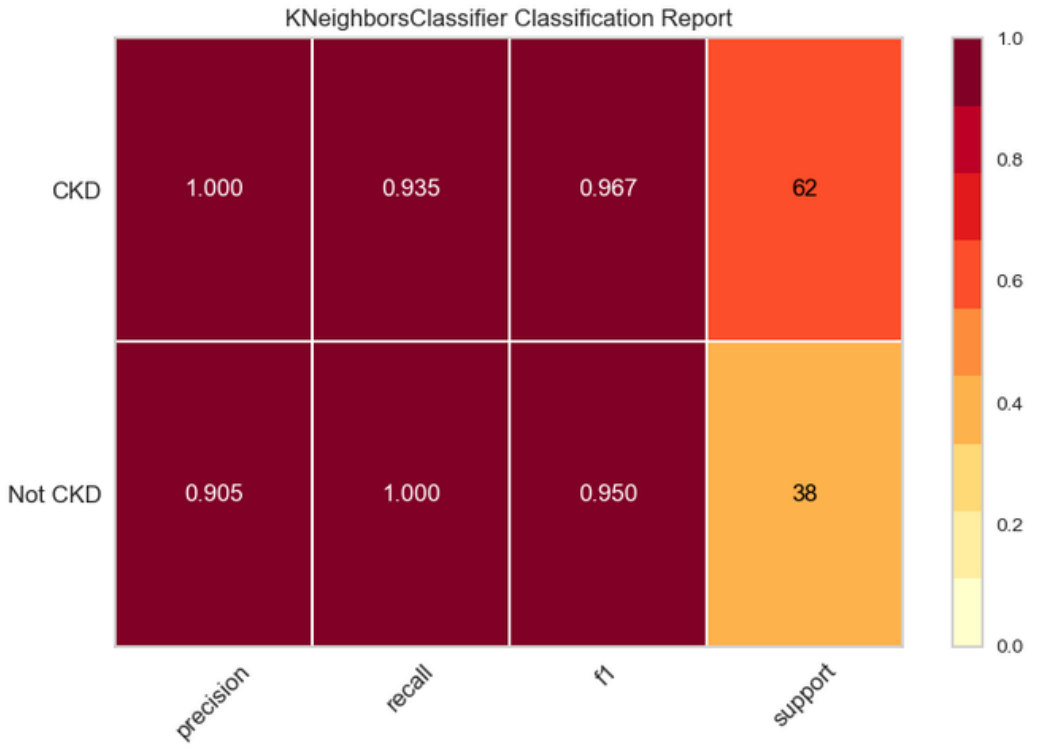
```
{'leaf_size': 2, 'n_neighbors': 2, 'p': 1}
```

```
knn = KNeighborsClassifier(n_neighbors=1,p=2,leaf_size=2)
knn_model = knn.fit(os_data_X1, os_data_y.values.ravel())
y_pred_knn = knn_model.predict(X_test1)
```

```
print('Accuracy of K-NN classifier on training set: {:.2f}'
     .format(knn.score(os_data_X1, os_data_y)))
print('Accuracy of K-NN classifier on test set: {:.2f}'
     .format(knn.score(X_test1, y_test)))
```

```
Accuracy of K-NN classifier on training set: 1.00
Accuracy of K-NN classifier on test set: 0.96
```

```
from sklearn.model_selection import TimeSeriesSplit
import yellowbrick
from yellowbrick.classifier import ClassificationReport
from yellowbrick.datasets import load_occupancy

# Specify the target classes
classes = ["Not CKD", "CKD"]

visualizer = ClassificationReport(knn_model, classes=classes, support=True)
visualizer.fit(os_data_X1, os_data_y)
visualizer.score(X_test1, y_test)    # Evaluate the model on the test data
visualizer.show()
```



Naive Bayes

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(os_data_X1, os_data_y.values.ravel())
```

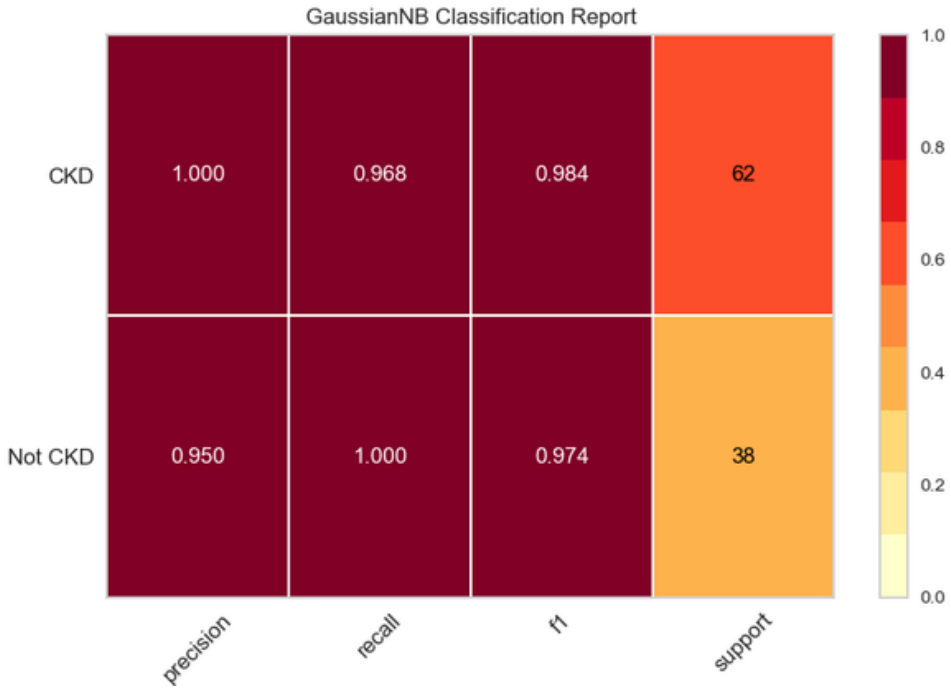
```
GaussianNB()
```

```
# making predictions on the testing set
y_pred = gnb.predict(X_test1)
```

```
# comparing actual response values (y_test) with predicted response values (y_pred)
from sklearn import metrics
print("Gaussian Naive Bayes model accuracy:", metrics.accuracy_score(y_test, y_pred))
```

```
Gaussian Naive Bayes model accuracy: 0.98
```

```
from sklearn.model_selection import TimeSeriesSplit
import yellowbrick
from yellowbrick.classifier import ClassificationReport
from yellowbrick.datasets import load_occupancy

# Specify the target classes
classes = ["Not CKD", "CKD"]

visualizer = ClassificationReport(gnb, classes=classes, support=True)
visualizer.fit(os_data_X1, os_data_y)
visualizer.score(X_test1, y_test)    # Evaluate the model on the test data
visualizer.show()
```



SVM

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV

# defining parameter range
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf','sigmoid','poly','linear']}

grid = GridSearchCV(SVC(), param_grid, refit = True)

# fitting the model for grid search
grid.fit(os_data_X1,  os_data_y.values.ravel())
```

```
GridSearchCV(estimator=SVC(),
             param_grid={'C': [0.1, 1, 10, 100, 1000],
                         'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                         'kernel': ['rbf', 'sigmoid', 'poly', 'linear']})
```

```
best_parameters = grid.best_params_
best_parameters
```

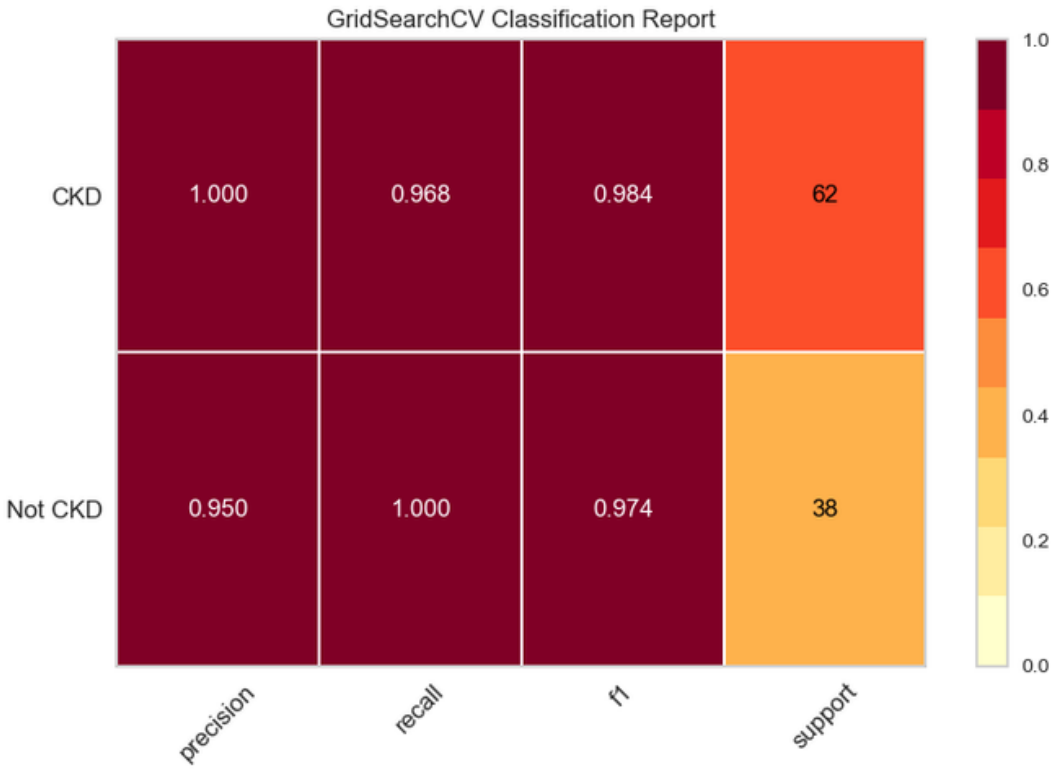
```
{'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
```

```
# make predictions on test set
y_pred1=grid.predict(X_test1)
```

```
# compute and print accuracy score
print('SVM model accuracy score with default hyperparameters: {0:0.4f}'. format
```

```
SVM model accuracy score with default hyperparameters: 0.9800
```

```
from sklearn.model_selection import TimeSeriesSplit
import yellowbrick
from yellowbrick.classifier import ClassificationReport
from yellowbrick.datasets import load_occupancy

# Specify the target classes
classes = ["Not CKD", "CKD"]

visualizer = ClassificationReport(grid, classes=classes, support=True)
visualizer.fit(os_data_X1, os_data_y)
visualizer.score(X_test1, y_test)    # Evaluate the model on the test data
visualizer.show()
```



Creating and testing the models with feature selection

KNN (with feature selection)

```
]: from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
leaf_size = list(range(2,20))# Par défaut c'est 30
n_neighbors = list(range(1,25))
p=[1,2] #p = 1 manhattan_distance/p=2 euclidean_distance
#Convert to dictionary
hyperparameters = dict(leaf_size=leaf_size, n_neighbors=n_neighbors, p=p)
#Create new KNN object
knn_2 = KNeighborsClassifier()
#Use GridSearch
clf = GridSearchCV(knn_2, hyperparameters, cv=10, n_jobs=-1)
#n_jobs: number of CPU'sfor execution.
#cv: number of folds of the cross validation

#Fit the model
best_model = clf.fit(os_data_X, os_data_y.values.ravel())
#Print The value of best Hyperparameters
print('Best leaf_size:', best_model.best_estimator_.get_params()['leaf_size'])
print('Best p:', best_model.best_estimator_.get_params()['p'])
print('Best n_neighbors:', best_model.best_estimator_.get_params()['n_neighbors'])
```

```
Best leaf_size: 2
Best p: 1
Best n_neighbors: 2
```

```
# Afficher les paramètres qui donnent les meilleurs performances
best_parameters = clf.best_params_
print(best_parameters)
```

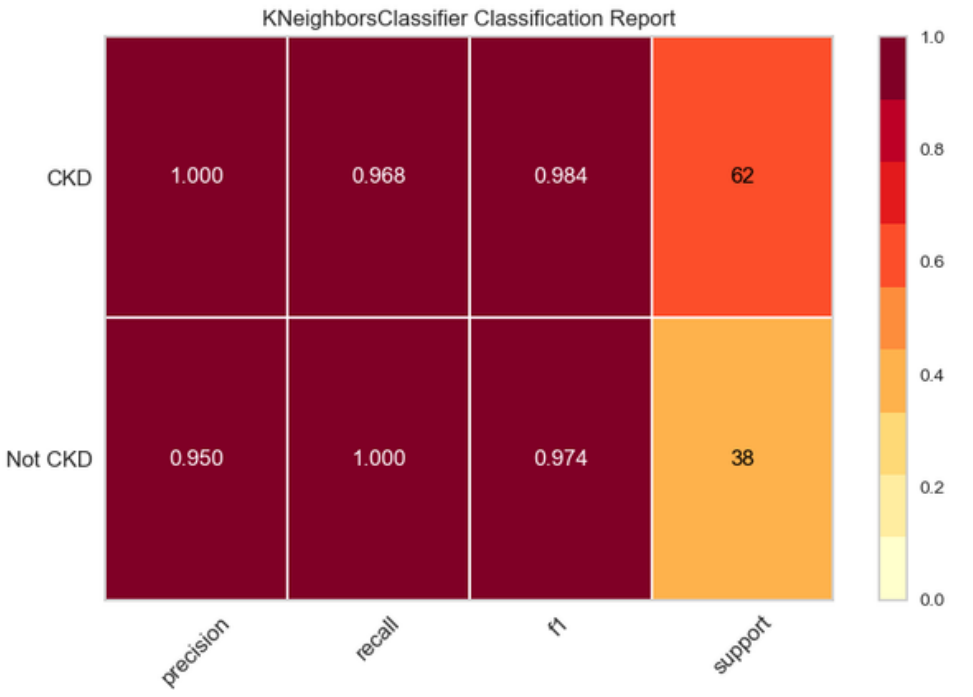
```
{'leaf_size': 2, 'n_neighbors': 2, 'p': 1}
```

```
knn = KNeighborsClassifier(n_neighbors=5,p=2,leaf_size=2 )
knn_model = knn.fit(os_data_X, os_data_y.values.ravel())
y_pred_knn = knn_model.predict(X_test)
```

```
print('Accuracy of K-NN classifier on training set: {:.2f}'
     .format(knn.score(os_data_X, os_data_y)))
print('Accuracy of K-NN classifier on test set: {:.2f}'
     .format(knn.score(X_test, y_test)))
```

```
Accuracy of K-NN classifier on training set: 0.98
Accuracy of K-NN classifier on test set: 0.98
```

```
from sklearn.model_selection import TimeSeriesSplit
import yellowbrick
from yellowbrick.classifier import ClassificationReport
from yellowbrick.datasets import load_occupancy

# Specify the target classes
classes = ["Not CKD", "CKD"]

visualizer = ClassificationReport(knn_model, classes=classes, support=True)
visualizer.fit(os_data_X, os_data_y)
visualizer.score(X_test, y_test)    # Evaluate the model on the test data
visualizer.show()
```



Naive Bayes (with feature selection)

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(os_data_X, os_data_y.values.ravel())
```

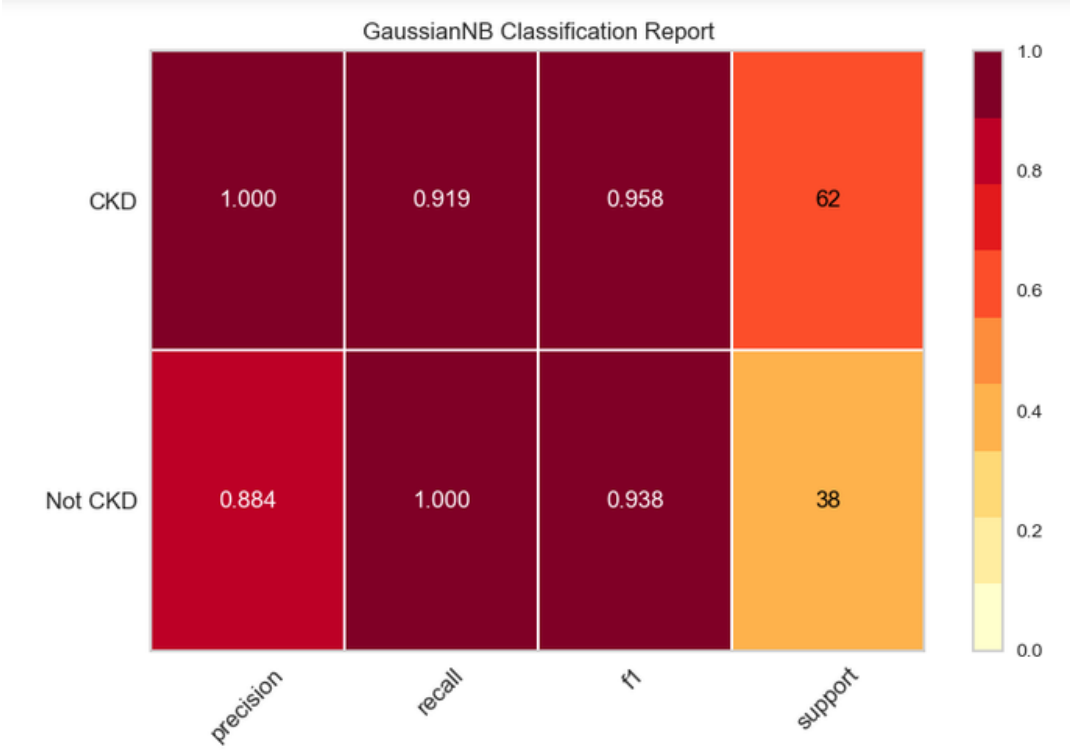
```
GaussianNB()
```

```
# making predictions on the testing set
y_pred = gnb.predict(X_test)
```

```
# comparing actual response values (y_test) with predicted response values (y_pred)
from sklearn import metrics
print("Gaussian Naive Bayes model accuracy:", metrics.accuracy_score(y_test, y_pred))
```

```
Gaussian Naive Bayes model accuracy: 0.95
```

```
#from sklearn.metrics import classification_report
#print(classification_report(y_test, y_pred))
from sklearn.model_selection import TimeSeriesSplit
import yellowbrick
from yellowbrick.classifier import ClassificationReport
from yellowbrick.datasets import load_occupancy

# Specify the target classes
classes = ["Not CKD", "CKD"]

visualizer = ClassificationReport(gnb, classes=classes, support=True)
visualizer.fit(os_data_X, os_data_y)
visualizer.score(X_test, y_test)    # Evaluate the model on the test data
```



SVM (with feature selection)

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV

# defining parameter range
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf','sigmoid','poly','linear']}

grid = GridSearchCV(SVC(), param_grid, refit = True)

# fitting the model for grid search
grid.fit(os_data_X,  os_data_y.values.ravel())
```

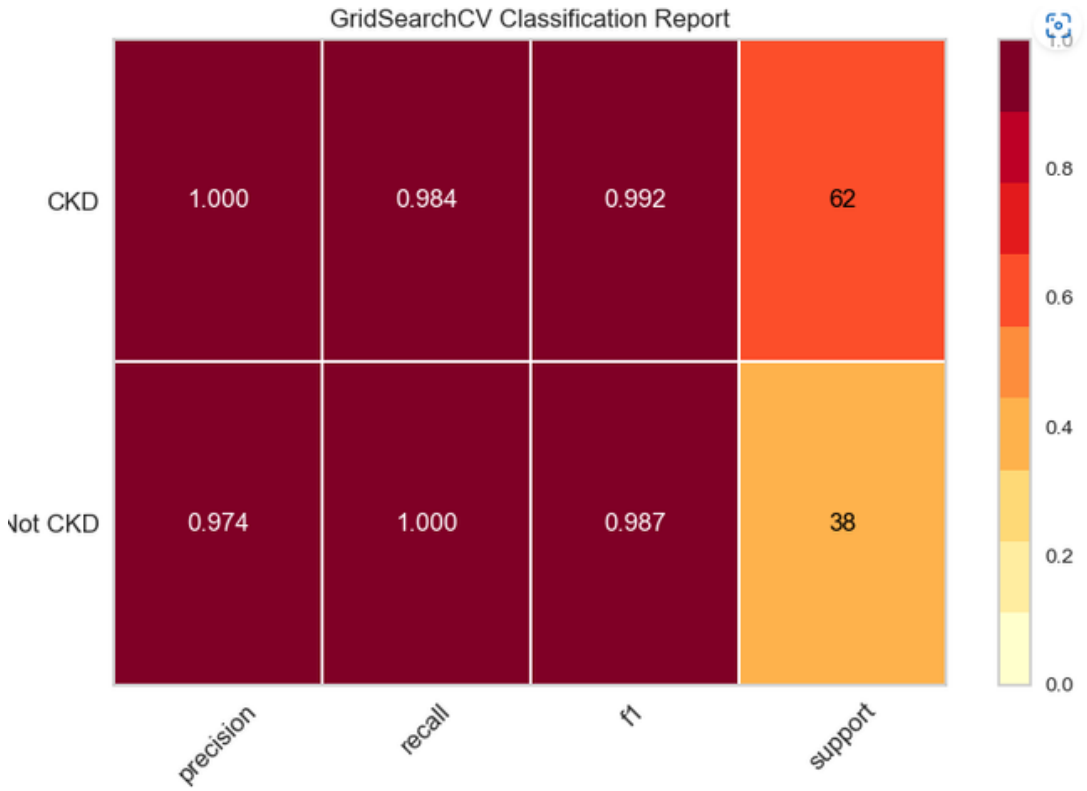
```
GridSearchCV(estimator=SVC(),
             param_grid={'C': [0.1, 1, 10, 100, 1000],
                         'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                         'kernel': ['rbf', 'sigmoid', 'poly', 'linear']})
```

```
# make predictions on test set
y_pred1=grid.predict(X_test)
```

```
# compute and print accuracy score
print('SVM model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred1)))
```

```
SVM model accuracy score: 0.9900
```

```
from sklearn.model_selection import TimeSeriesSplit
import yellowbrick
from yellowbrick.classifier import ClassificationReport
from yellowbrick.datasets import load_occupancy

# Specify the target classes
classes = ["Not CKD", "CKD"]

visualizer = ClassificationReport(grid, classes=classes, support=True)
visualizer.fit(os_data_X, os_data_y)
visualizer.score(X_test, y_test)    # Evaluate the model on the test data
visualizer.show()
```



Creating and testing the models with adaboost (after feature selection)

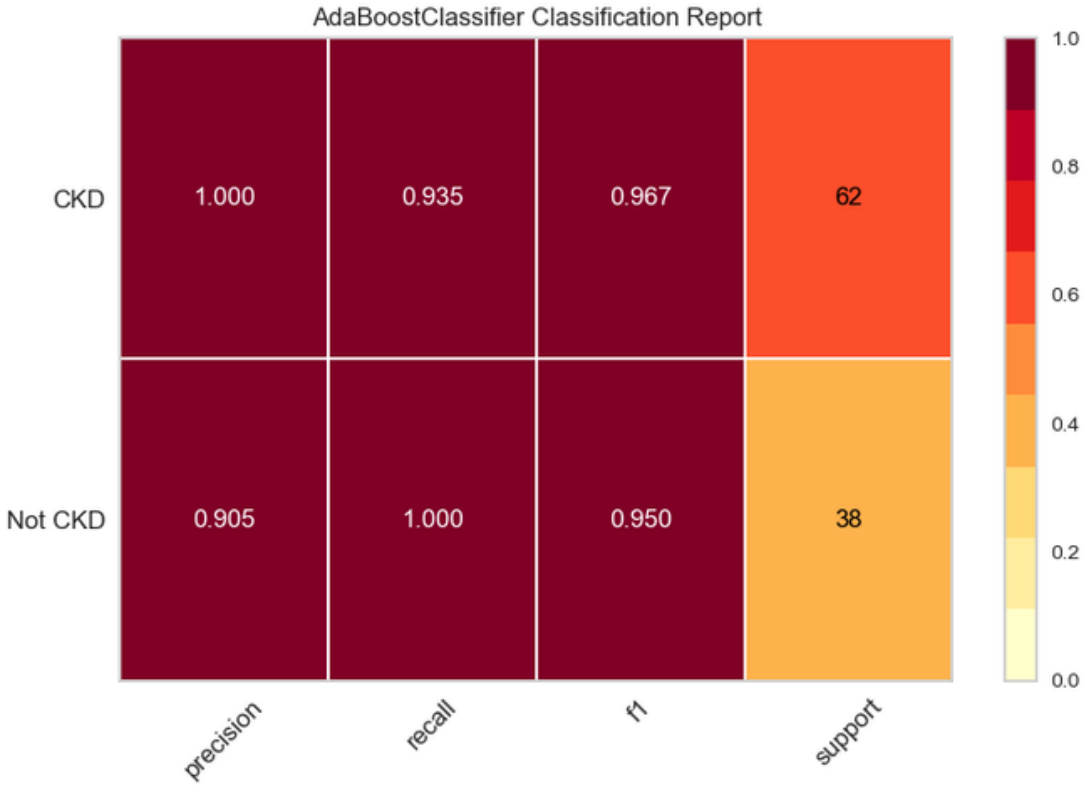
SVM With AdaBoost (after feature selection)

```
from sklearn.ensemble import AdaBoostClassifier
abcl = AdaBoostClassifier(base_estimator=SVC(),algorithm='SAMME',n_estimators=100,random_state=100)
abcl = abcl.fit(os_data_X, os_data_y.values.ravel())
y_predict = abcl.predict(X_test)
```

```
from sklearn import metrics
#Evaluate the model using accuracy and confusion matrix
acc_AB=metrics.accuracy_score(y_test, y_predict)
print('Accuracy using AdaBoosting: ',acc_AB)
print('Confusion Matrix: \n',metrics.confusion_matrix(y_test, y_predict))
```

```
Accuracy using AdaBoosting:  0.96
Confusion Matrix:
 [[38  0]
 [ 4 58]]
```

```
from sklearn.model_selection import TimeSeriesSplit
import yellowbrick
from yellowbrick.classifier import ClassificationReport
from yellowbrick.datasets import load_occupancy

# Specify the target classes
classes = ["Not CKD", "CKD"]
```



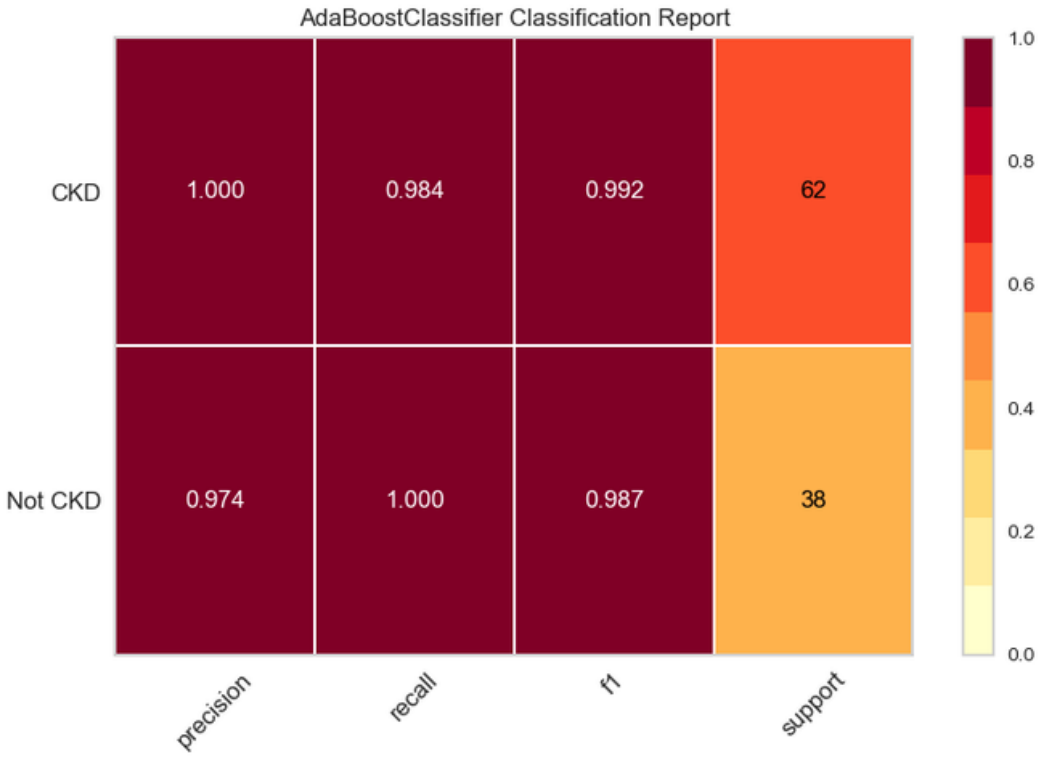
Naive-Bayes With AdaBoost (after feature selection)

```
from sklearn.ensemble import AdaBoostClassifier
abcl = AdaBoostClassifier(base_estimator=GaussianNB(),n_estimators=100,random_state=100)
abcl = abcl.fit(os_data_X, os_data_y.values.ravel())
y_predict = abcl.predict(X_test)
```

```
from sklearn import metrics
#Evaluate the model using accuracy and confusion matrix
acc_AB=metrics.accuracy_score(y_test, y_predict)
print('Accuracy using AdaBoosting: ',acc_AB)
print('Confusion Matrix: \n',metrics.confusion_matrix(y_test, y_predict))
```

```
Accuracy using AdaBoosting:  0.99
Confusion Matrix:
 [[38  0]
 [ 1 61]]
```

```
from sklearn.model_selection import TimeSeriesSplit
import yellowbrick
from yellowbrick.classifier import ClassificationReport
from yellowbrick.datasets import load_occupancy

# Specify the target classes
classes = ["Not CKD", "CKD"]

visualizer = ClassificationReport(abcl, classes=classes, support=True)
visualizer.fit(os_data_X, os_data_y)
visualizer.score(X_test, y_test)    # Evaluate the model on the test data
visualizer.show()
```



3- Article 3 :

- **Logistic regression**

Binary outcomes are modeled using the statistical method of logistic regression, which is well known in the field. Different learning methods are used to execute logistic regression in statistical research. A variant of the neural network method was used to create the LR algorithm. This method resembles neural networks in many ways, but it is simpler to set up and use.

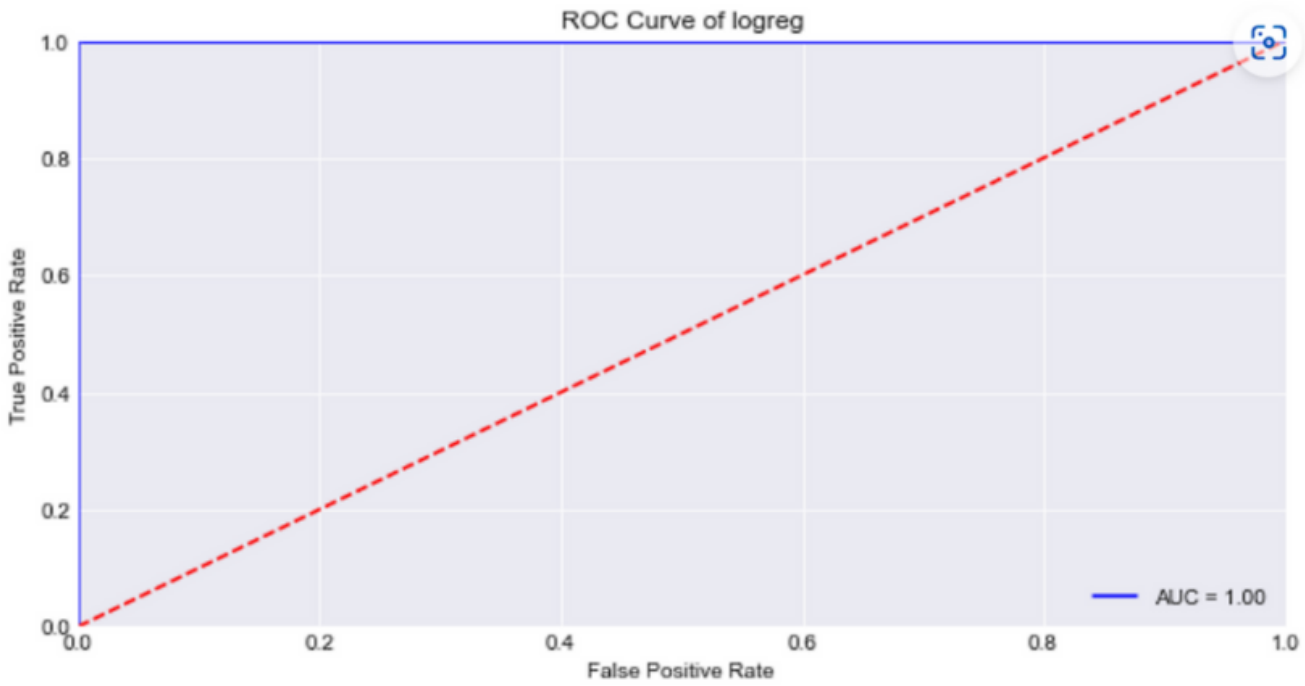
Logistic Regression

Entrée [87]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, auc
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=5)
logreg=LogisticRegression(random_state=0)
logreg.fit(X_train,y_train)
```

Out[87]:
```
LogisticRegression(random_state=0)
```

AUC-ROC CURVE

```
y_pred=logreg.predict(X_test)
from sklearn.metrics import plot_confusion_matrix, classification_report
plot_confusion_matrix(logreg,X_test,y_test)
print(classification_report(y_test,y_pred,digits=8))
```



Score Logistic regression

```
print(f"Training Accuracy of Logistic Regression is {accuracy_score(y_train, logreg.predict(X_train))}")
logreg_acc = accuracy_score(y_test, logreg.predict(X_test))
print(f"Test Accuracy of Logistic Regression is {logreg_acc} \n")
```

```
Training Accuracy of Logistic Regression is 0.99375
Test Accuracy of Logistic Regression is 0.975
```

- **XGBOOST**

Binary outcomes are modeled using the statistical method of logistic regression, which is well known in the field. Different learning methods are used to execute logistic regression in statistical research. A variant of the neural network method was used to create the LR algorithm. This method resembles neural networks in many ways, but it is simpler to set up and use.

XGBOOST

```
from xgboost import XGBClassifier

xgb = XGBClassifier(objective = 'binary:logistic', learning_rate = 0.5, max_depth = 5, n_estimators = 150)
xgb.fit(X_train, y_train)

# accuracy score, confusion matrix and classification report of xgboost

xgb_acc = accuracy_score(y_test, xgb.predict(X_test))

print(f"Training Accuracy of XgBoost is {accuracy_score(y_train, xgb.predict(X_train))}")
print(f"Test Accuracy of XgBoost is {xgb_acc} \n")

print(f"Confusion Matrix :- \n{confusion_matrix(y_test, xgb.predict(X_test))}\n")
print(f"Classification Report :- \n {classification_report(y_test, xgb.predict(X_test))}")
```

```
Training Accuracy of XgBoost is 1.0
Test Accuracy of XgBoost is 0.9875

Confusion Matrix :-
[[43  1]
 [ 0 36]]

Classification Report :-
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        44
           1       0.97      1.00      0.99        36

    accuracy                           0.99        80
   macro avg       0.99      0.99      0.99        80
weighted avg       0.99      0.99      0.99        80
```

- **StackingClassifier**

```
from sklearn.ensemble import StackingClassifier
model = StackingClassifier([('SVM', svmg),
                            ('LogisticRegression', logreg),
                            ('KNN', knn)],
                             final_estimator= SVC(kernel="sigmoid"))

model.fit(X_train, y_train)
st=model.score(X_test, y_test)
print (f"Training Accuracy of StackingClassifier is {accuracy_score(y_train, abc.predict(X_train))}")
```

IV- : Evaluation :

1-Article 1 :

the dataset was divide into 75% training and 25% testing and validation. the dataset was processed to remove outliers and replace missing numerical and nominal values using mean and mode statistical measures, respectively. the RFE algorithm was applied to select the most strongly representative features of CKD.
Selected features were fed into classification algorithms: SVM, KNN, decision tree, and random forest.
The parameters of all classifiers were tuned to perform the best classification, so all algorithms reached promising results. the random forest algorithm outperformed all other algorithms, achieving an accuracy, precision, recall, and F1-score of 99% for all measures

Accuracy

Proposed model (random forest)	**0.99**
Proposed model (decision tree)	**0.99**
Proposed model (KNN)	**0.96**
Proposed model (SVM)	**0.9800**

2-Article 2 :

2-1- RESULT AND DISCUSSION

Parameter	Classifiers					
	NB		KNN		SVM	
	1 st Method : Base					
Accuracy	0.98		0.96		0.98	
Precision	0.95		0.90		0.95	
Recall	0. 968		0.94		0.968	
F-measure	CKD=0.984	Not CKD=0.974	CKD=0.967	Not CKd=0.95	CKD=0.984	Not CKD=0.974
	2nd Method : CFS					
Accuracy	↓0.95		↑0.98		↑0.99	
Precision	↓0.884		↑0.95		↑0.974	
Recall	↓0.919		↑0.968		↑0.984	
F-measure	↓CKD=0.958	↓Not CKD=0.938	↑CKD=0.984	↑Not CKD=0.974	↑CKD=0.992	↑Not CKD=0.987
	3rd Method : CFS+ Adaboost					
Accuracy	↑↑0.99				↓↓0.96	
Precision	↑↑0.974				↑↓0.90	
Recall	↑↑0.984				↓↓0.935	
F-measure	↑↑CKD=0.99	↑↑Not CKD=0.987			↓↓CKD=0.967	↓↓Not CKD=0.95

The classification is conducted by three different methods, the first is classification by base classifier without a feature selection method and ensemble learning, the second is the result of the classification with feature selection but without the ensemble learning, the third is the result of the classification from selected features and ensemble learning.

The accuracy on 1st method using base classifier are quite high, the lowest accuracy is 0.96. Accuracy rate in kNN is 0.96, both Naive Bayes and SVM accuracy rate is 0.98.

The 2nd method with CFS feature selection was able to increase the accuracy of base classifier in KNN and SVM but it causes a decrease in Naive Bayes.Accuracy rate in kNN and SVM are increased respectively by 0.02, 0.01 . Accuracy rate in Naive Bayes is decreased respectively by 0.03.

On the third method, features are selected by CFS and the classification is conducted with ensemble learning AdaBoost. Accuracy rate in Naive Bayes is 0.99 and SVM is 0.96. Highest accuracy rate is achieved on third method with Naive Bayes as a base classifier.

On base classifier, precision rate of Naive Bayes is 0.95, SVM is 0.95 and kNN is 0.90. Precision rate is increased on second method, kNN is 0.95 and SVM is 0.974 but the Precision rate in Naive Bayes is 0.884 so it decreased.

The third method using CFS and AdaBoost also managed to increase the value of precision on the Naive Bayes became 0.974 but a decrease in SVM which is 0.9 .

The recall rate is increased in Naive Bayes and decreased in SVM. Classification on first method using Naive Bayes generates recall rate of 0.968, kNN is 0.94and SVM is 0.968. Selected features with CFS is decreased recall rate in Naive Bayes by 0,049, increased in kNN by 0,028 and SVM by 0,016.

In third method, the recall rate from Naive Bayes and SVM consecutively are 0.984, 0.935 .

The proposed method, namely AdaBoost and CFS was able to improve the classification results on the base classifier Naive Bayes . The improvement of classification result can be seen in the four test parameters i.e. accuracy; precision; recall and f-measure. The best results of classification is obtained from the second method with SVM and KNN as base classifier with the difference in the accuracy and precision. Accuracy rate in SVM is 0.99 and KNN is 0.98 and precision rate in SVM is 0.974 and KNN is 0.95 .

	Model	Score
2	SVM	0.99
0	KNN	0.98
1	Naive BAyes	0.95

2-2 Conclusion

This study was designed to diagnose chronic kidney disease based on 24 attribute which includes symptoms, signs and risk factors of chronic kidney disease. The initial features were selected using the method of Correlation-based Feature Selection (CFS). On the classification stage, AdaBoost was used for enhancing classification result. Three classifier, namely k-Nearest Neighbour (kNN), Support Vector Machine (SVM) and Naive Bayes were used to examining the effect of CFS and AdaBoost in enhancing classification result. There were three different classification method conducted. The first, classification conducted by only the base classifier. In the second method, classification was conducted after features was selected by CFS. In the third method, selected features were trained by AdaBoost learning. Classification was evaluated by using four parameters, namely accuracy; precision; recall and f-measure. Based on four parameters of evaluation, CFS and AdaBoost were successful in improving chronic kidney disease diagnosis.

There were increment in all classification methods. The best result was achieved by SVM classifier with 0.99 of accuracy rate.

3- Notebook 3 :

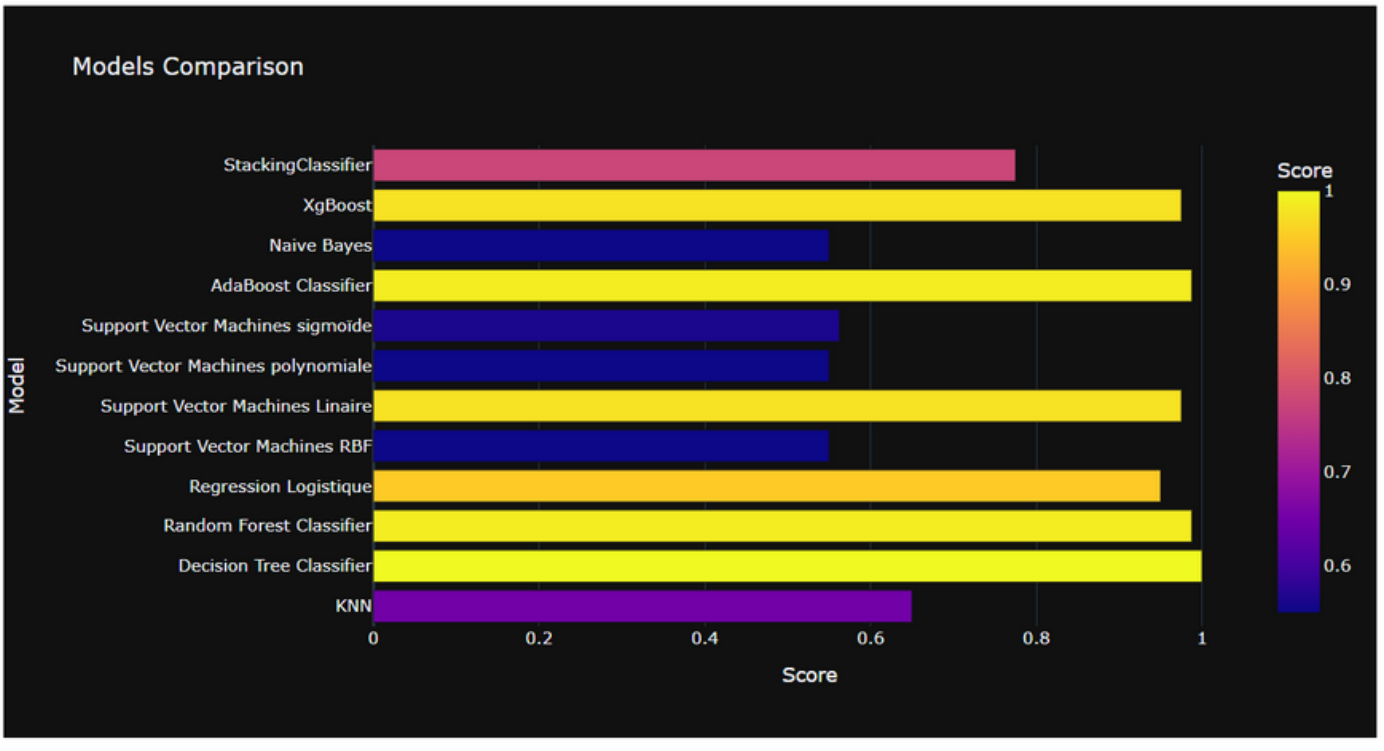
- **Algorithms Comparison:**

The comparison will help us classify the algorithms accuracy scores

Toggle Pandas/Lux

	Model	Score
1	Decision Tree Classifier	1.0000
2	Random Forest Classifier	0.9875
8	AdaBoost Classifier	0.9875
5	Support Vector Machines Linaire	0.9750
10	XgBoost	0.9750
3	Regression Logistique	0.9500
11	StackingClassifier	0.7750
0	KNN	0.6500
7	Support Vector Machines sigmoïde	0.5625
4	Support Vector Machines RBF	0.5500
6	Support Vector Machines polynomiale	0.5500
9	Naive Bayes	0.5500

```
Entrée [162]: px.bar(data_frame = models, x = 'Score', y = 'Model', color = 'Score', template = 'plotly_dark',
                     title = 'Models Comparison')
```



V.OVERALL EVALUATION

According to the findings of the study, the decision tree approach and logistic regression can be used to predict chronic kidney disease more accurately.

According to the study, their precision was 96.25 percent, and their accuracy was 97 percent. Compared to prior research, the accuracy percent of the models used in this investigation is considerably higher, indicating that the models used in this study are more reliable than those used in previous studies.

When cross validation measurements are used in the prediction of chronic kidney disease, the LR method outperforms the other processes. Future research may build on this work by developing a web application that incorporates these algorithms and using a bigger dataset than the one utilized in this study. This will aid in the achievement of improved outcomes as well as the accuracy and efficiency with which healthcare practitioners can anticipate kidney issues. This will enhance the dependability of the framework as well as the framework's presentation. The hope is that it would encourage people to seek early treatment for chronic renal disease and to make improvements in their lives.