**Early Prediction for Chronic Kidney Disease Detection: A**

**Progressive Approach to Health Management**

Abstract

Every year, an increasing number of patients are diagnosed with late stages of renal disease. Chronic Kidney Disease, also known as Chronic Renal Disease, is characterized by abnormal kidney function or a breakdown of renal function that progresses over months or years. Chronic kidney disease is often found during screening of persons who are known to be at risk for kidney issues, such as those with high blood pressure or diabetes, and those with a blood family who has chronic kidney disease (CKD). As a result, early prognosis is critical in battling the disease and providing effective therapy. Only early identification and continuous monitoring can avoid serious kidney damage or renal failure. Machine Learning (ML) plays a significant part in the healthcare system, and it may efficiently aid and help with decision support in medical institutions. The primary goals of this research are to design and suggest a machine learning method for predicting CKD. Random Forest (LR), Artificial Neural Network (ANN), and Decision Tree are three master teaching methodologies investigated (DT). The components are built using chronic kidney disease datasets, and the outcomes of these models are compared to select the optimal model for prediction.

Introduction

Chronic Kidney Disease (CKD) is a major medical problem and can be cured if treated in the early

stages. Usually, people are not aware that medical tests we take for different purposes could contain

valuable information concerning kidney diseases. Consequently, attributes of various medical tests

are investigated to distinguish which attributes may contain helpful information about the disease. The

information says that it helps us to measure the severity of the problem, the predicted survival of the

patient after the illness, the pattern of the disease and work for curing the disease.

In today world as we know most of the people are facing so many disease and as this can be cured

if we treat people in early stages this project can use a pre trained model to predict the Chronic Kidney

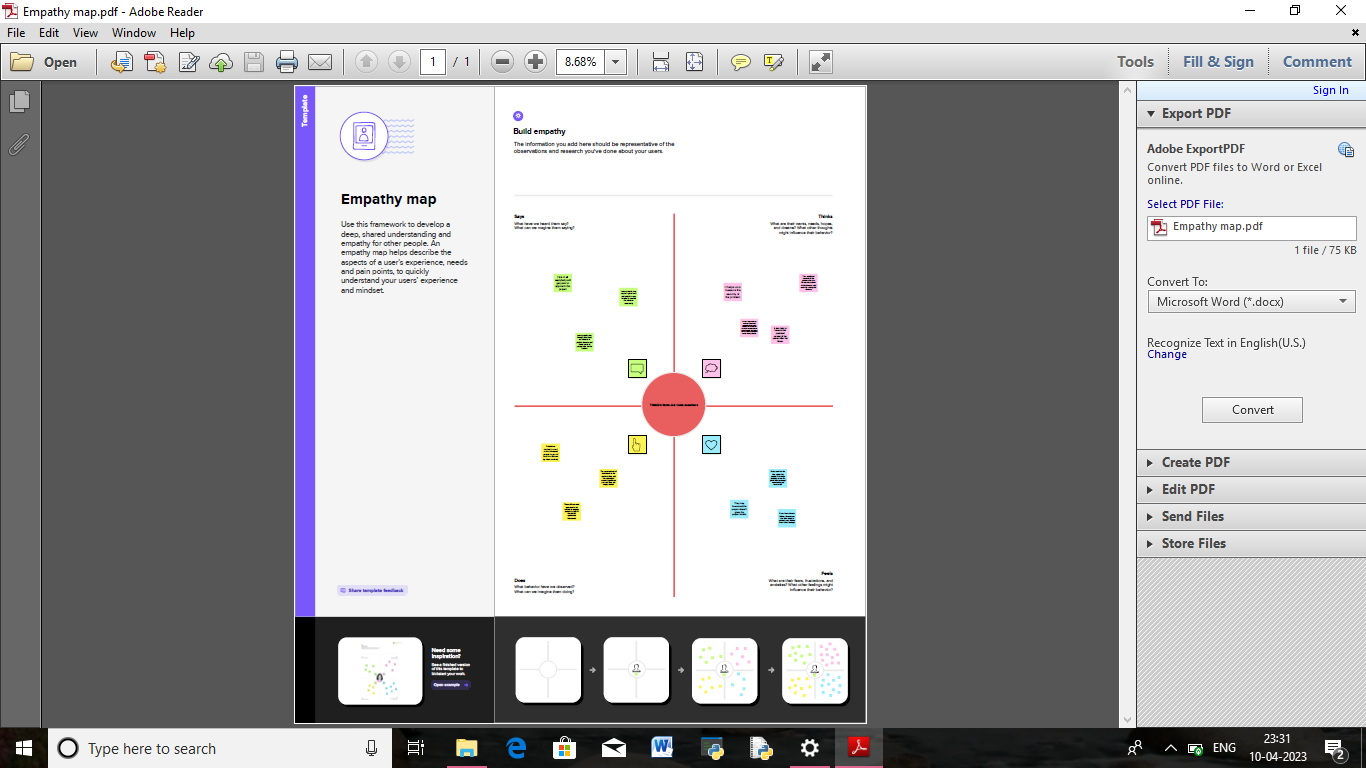
Disease which can help in treatments of peoples who are suffer from this disease.

Purpose

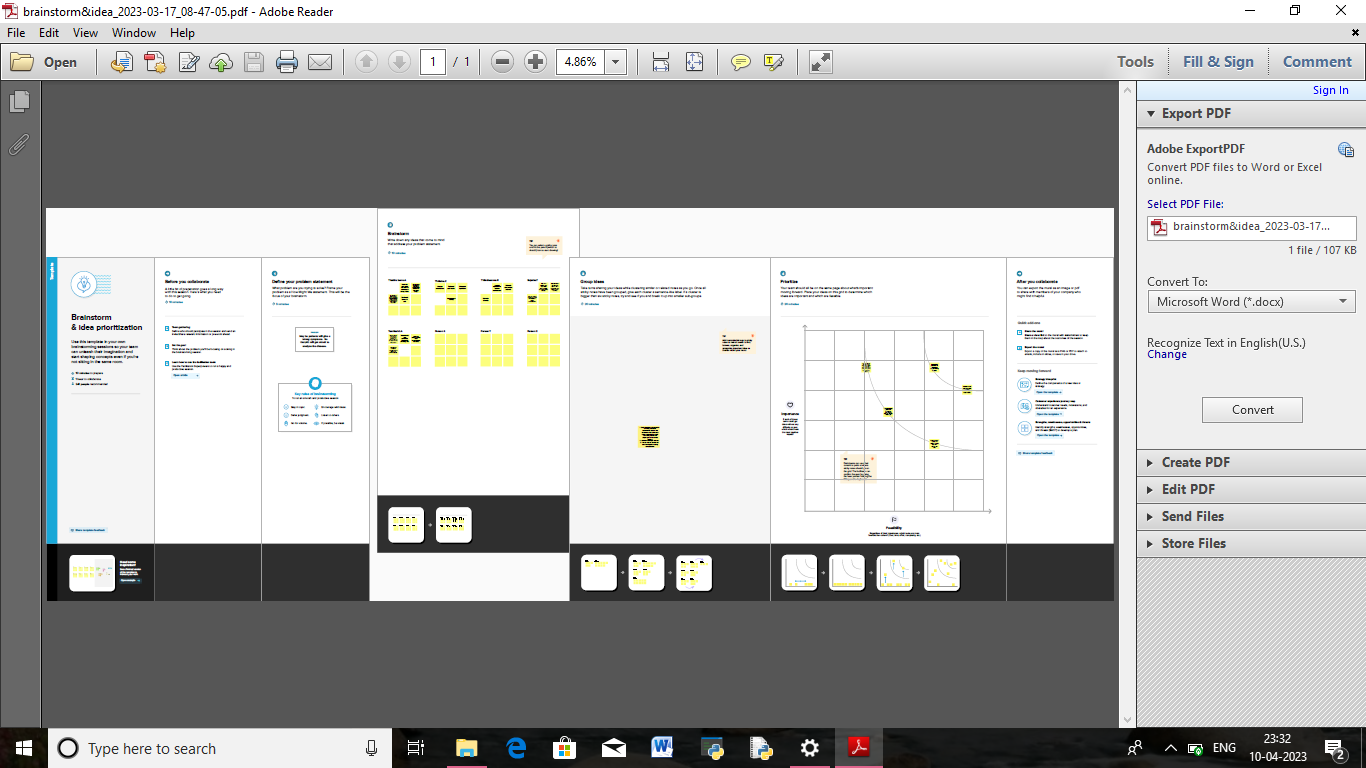
The rationale for testing asymptomatic people for CKD is that earlier detection might allow for the implementation of therapeutic interventions and avoidance of inappropriate exposure to nephrotoxic agents, both of which may slow the progression of CKD to end-stage kidney disease.

Problem Definition & Design Thinking

Empathy Map



Ideation & Brainstorming Map



Result

The application uses ANN and Naive Bayes Algorithms for classification. The application has Admin module which is the main module to maintain the application. After admin’s successful login he can add doctors and receptionists. The receptionist will add the training dataset (old patient) and register’s the new patient. Doctor can analyze whether a patient have CKD or not and also determine the CKD stage if patient having CKD. Also, doctors have an option to upload treatment details for particular patient. The patient can view his treatment details by logging in to the application.

Advantages & Disadvantages

Advantages

* The early detection of CKD allows patients to receive timely treatment, slowing the disease's progression. Due to its rapid recognition performance and accuracy, machine learning models can effectively assist physicians in achieving this goal.
* Your kidneys act like a filter to remove wastes and extra fluid from your body. Your kidneys filter about 200 quarts of blood each day to make about 1 to 2 quarts of urine. The urine contains wastes and extra fluid. This prevents buildup of wastes and fluid to keep your body healthy.

Disadvantage

* Having CKD increases the chances of having heart disease and stroke.
* Managing high blood pressure, blood sugar, and cholesterol levels—all factors that increase the risk for heart disease and stroke—is very important for people with CKD.

Application

Predictive analytics using machine learning helps detect fraudulent activities in the financial sector. Fraudulent transactions are identified by training machine learning algorithms with past datasets. The models find risky patterns in these datasets and learn to predict and deter fraud.

Conclusion

* This project is a medical sector application which helps the medical practitioners in predicting the CKD disease based on the CKD parameters. It is automation for CKD disease prediction and **it identifies** the disease, its stages in an efficient and economically manner
* It is successfully accomplished by applying the ANN for classification. This classification technique comes under data mining technology. This algorithm takes CKD parameters as input and predicts the disease based on old CKD patient’s data.

Future scope

The work will be considered as basement for the healthcare system for CKD patients. Also extensionto this work is that implementation of deep learning since deep learning provides high-quality performance than machine learning algorithm.

Appenix

Source Code

Milestone 2:

import pandas as pd

import numpy as np

from collections import Counter as c

import matplotlib.pyplot as plt

import seaborn as sns

import missingno as msno

from sklearn.metrics import accuracy\_score,confusion\_matrix

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

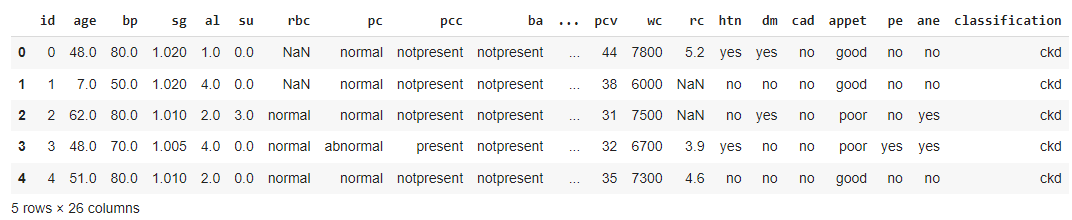
from sklearn.linear\_model import LogisticRegression

import pickle

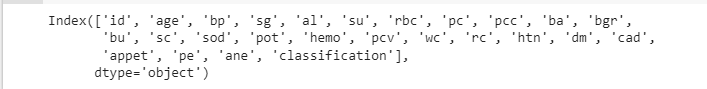
Read the data set

data=pd.read\_csv("/content/kidney\_disease.csv")

data.head()

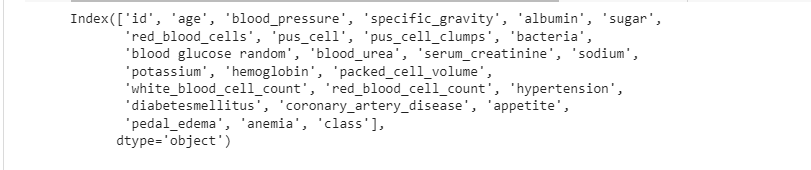


data.columns



data.columns=['id','age','blood\_pressure','specific\_gravity','albumin','sugar','red\_blood\_cells','pus\_cell','pus\_cell\_clumps','bacteria','blood glucose random','blood\_urea','serum\_creatinine','sodium','potassium','hemoglobin','packed\_cell\_volume','white\_blood\_cell\_count','red\_blood\_cell\_count','hypertension','diabetesmellitus','coronary\_artery\_disease','appetite','pedal\_edema','anemia','class']

data.columns



data.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 400 entries, 0 to 399

Data columns (total 26 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 id 400 non-null int64

1 age 391 non-null float64

2 blood\_pressure 388 non-null float64

3 specific\_gravity 353 non-null float64

4 albumin 354 non-null float64

5 sugar 351 non-null float64

6 red\_blood\_cells 248 non-null object

7 pus\_cell 335 non-null object

8 pus\_cell\_clumps 396 non-null object

9 bacteria 396 non-null object

10 blood glucose random 356 non-null float64

11 blood\_urea 381 non-null float64

12 serum\_creatinine 383 non-null float64

13 sodium 313 non-null float64

14 potassium 312 non-null float64

15 hemoglobin 348 non-null float64

16 packed\_cell\_volume 330 non-null object

17 white\_blood\_cell\_count 295 non-null object

18 red\_blood\_cell\_count 270 non-null object

19 hypertension 398 non-null object

20 diabetesmellitus 398 non-null object

21 coronary\_artery\_disease 398 non-null object

22 appetite 399 non-null object

23 pedal\_edema 399 non-null object

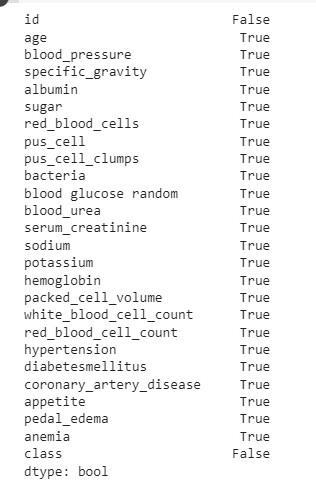
24 anemia 399 non-null object

25 class 400 non-null object

dtypes: float64(11), int64(1), object(14)

memory usage: 81.4+ KB

data.isnull().any()



data['blood glucose random'].fillna(data['blood glucose random'].mode()[0],inplace=True)

data['blood\_pressure'].fillna(data['blood\_pressure'].mean(),inplace=True)

data['blood\_urea'].fillna(data['blood\_urea'].mean(),inplace=True)

data['hemoglobin'].fillna(data['hemoglobin'].mean(),inplace=True)

data['potassium'].fillna(data['potassium'].mean(),inplace=True)

data['packed\_cell\_volume'].fillna(data['packed\_cell\_volume'].mode()[0],inplace=True)

data['red\_blood\_cell\_count'].fillna(data['red\_blood\_cell\_count'].mode()[0],inplace=True)

data['serum\_creatinine'].fillna(data['serum\_creatinine'].mean(),inplace=True)

data['sodium'].fillna(data['sodium'].mean(),inplace=True)

data['white\_blood\_cell\_count'].fillna(data['white\_blood\_cell\_count'].mode()[0],inplace=True)

data['age'].fillna(data['age'].mean(),inplace=True)

data['hypertension'].fillna(data['hypertension'].mode()[0],inplace=True)

data['pus\_cell\_clumps'].fillna(data['pus\_cell\_clumps'].mode()[0],inplace=True)

data['appetite'].fillna(data['appetite'].mode()[0],inplace=True)

data['albumin'].fillna(data['albumin'].mean(),inplace=True)

data['pus\_cell'].fillna(data['pus\_cell'].mode()[0],inplace=True)

data['red\_blood\_cells'].fillna(data['red\_blood\_cells'].mode()[0],inplace=True)

data['coronary\_artery\_disease'].fillna(data['coronary\_artery\_disease'].mode()[0],inplace=True)

data['bacteria'].fillna(data['bacteria'].mode()[0],inplace=True)

data['anemia'].fillna(data['anemia'].mode()[0],inplace=True)

data['sugar'].fillna(data['sugar'].mean(),inplace=True)

data['diabetesmellitus'].fillna(data['diabetesmellitus'].mode()[0],inplace=True)

data['pedal\_edema'].fillna(data['pedal\_edema'].mode()[0],inplace=True)

data['specific\_gravity'].fillna(data['specific\_gravity'].mean(),inplace=True)

catcols=set(data.dtype[data.dtypes=='0'].index.values)

print(catcols)

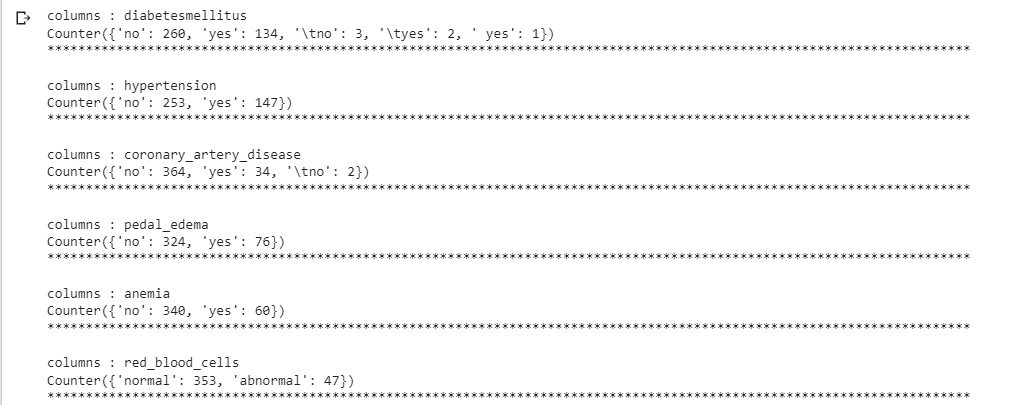
{'diabetesmellitus', 'hypertension', 'coronary\_artery\_disease', 'pedal\_edema', 'anemia', 'red\_blood\_cells', 'pus\_cell\_clumps', 'white\_blood\_cell\_count', 'packed\_cell\_volume', 'red\_blood\_cell\_count', 'appetite', 'class', 'bacteria', 'pus\_cell'}

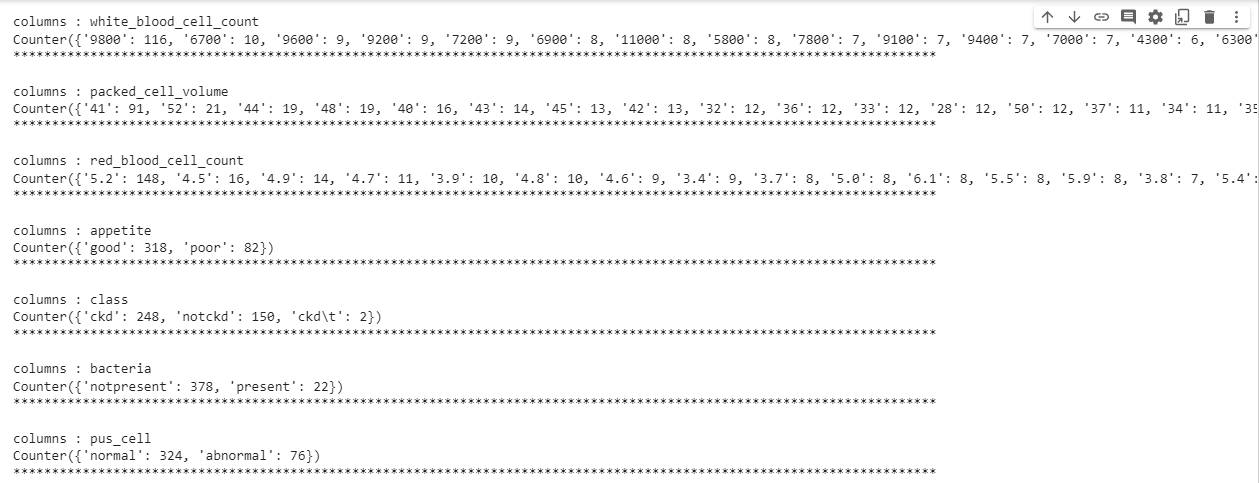
for i in catcols:

  print("columns :",i)

  print(c(data[i]))

  print('\*'\*120+'\n')





catcols.remove('red\_blood\_cell\_count')

catcols.remove('packed\_cell\_volume')

catcols.remove('white\_blood\_cell\_count')

print(catcols)

{'diabetesmellitus', 'hypertension', 'coronary\_artery\_disease', 'pedal\_edema', 'anemia', 'red\_blood\_cells', 'pus\_cell\_clumps', 'appetite', 'class', 'bacteria', 'pus\_cell'}

catcols={'anemia','pedal\_edema','appetite','bacteria','class','coronary\_artery\_disease','diabetesmellitus','hypertension','pus\_cell','pus\_cell\_clumps','red\_blood\_cells'}

from sklearn.preprocessing import LabelEncoder

for i in catcols:

  print("LABEL ENCODING OF:",i)

  LEi = LabelEncoder()

  print(c(data[i]))

  data[i] = LEi.fit\_transform(data[i])

  print(c(data[i]))

  print("\*"\*100)

LABEL ENCODING OF: diabetesmellitus

Counter({'no': 260, 'yes': 134, '\tno': 3, '\tyes': 2, ' yes': 1})

Counter({3: 260, 4: 134, 0: 3, 1: 2, 2: 1})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LABEL ENCODING OF: coronary\_artery\_disease

Counter({'no': 364, 'yes': 34, '\tno': 2})

Counter({1: 364, 2: 34, 0: 2})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LABEL ENCODING OF: hypertension

Counter({'no': 253, 'yes': 147})

Counter({0: 253, 1: 147})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LABEL ENCODING OF: pedal\_edema

Counter({'no': 324, 'yes': 76})

Counter({0: 324, 1: 76})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LABEL ENCODING OF: anemia

Counter({'no': 340, 'yes': 60})

Counter({0: 340, 1: 60})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LABEL ENCODING OF: red\_blood\_cells

Counter({'normal': 353, 'abnormal': 47})

Counter({1: 353, 0: 47})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LABEL ENCODING OF: pus\_cell\_clumps

Counter({'notpresent': 358, 'present': 42})

Counter({0: 358, 1: 42})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LABEL ENCODING OF: class

Counter({'ckd': 248, 'notckd': 150, 'ckd\t': 2})

Counter({0: 248, 2: 150, 1: 2})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LABEL ENCODING OF: appetite

Counter({'good': 318, 'poor': 82})

Counter({0: 318, 1: 82})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LABEL ENCODING OF: bacteria

Counter({'notpresent': 378, 'present': 22})

Counter({0: 378, 1: 22})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LABEL ENCODING OF: pus\_cell

Counter({'normal': 324, 'abnormal': 76})

Counter({1: 324, 0: 76})

contcols=set(data.dtypes[data.dtypes!='O'].index.values)

print(contcols)

{'hypertension', 'anemia', 'specific\_gravity', 'potassium', 'bacteria', 'coronary\_artery\_disease', 'pus\_cell\_clumps', 'sugar', 'blood\_pressure', 'appetite', 'hemoglobin', 'blood\_urea', 'pedal\_edema', 'blood glucose random', 'red\_blood\_cells', 'serum\_creatinine', 'id', 'diabetesmellitus', 'sodium', 'albumin', 'age', 'class', 'pus\_cell'}

for i in contcols:

  print("Continous Columns :",i)

  print(c(data[i]))

  print('\*'\*120+'\n')

Continous Columns : hypertension

Counter({0: 253, 1: 147})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Continous Columns : anemia

Counter({0: 340, 1: 60})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Continous Columns : specific\_gravity

Counter({1.02: 106, 1.01: 84, 1.025: 81, 1.015: 75, 1.0174079320113314: 47, 1.005: 7})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Continous Columns : potassium

Counter({4.62724358974359: 88, 5.0: 30, 3.5: 30, 4.9: 27, 4.7: 17, 4.8: 16, 4.0: 14, 4.2: 14, 4.1: 14, 3.8: 14, 3.9: 14, 4.4: 14, 4.5: 13, 3.7: 12, 4.3: 12, 3.6: 8, 4.6: 7, 3.4: 5, 5.2: 5, 5.7: 4, 5.3: 4, 3.2: 3, 5.5: 3, 2.9: 3, 5.4: 3, 6.3: 3, 3.3: 3, 2.5: 2, 5.8: 2, 5.9: 2, 5.6: 2, 3.0: 2, 6.5: 2, 6.4: 1, 6.6: 1, 39.0: 1, 7.6: 1, 47.0: 1, 5.1: 1, 2.8: 1, 2.7: 1})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Continous Columns : bacteria

Counter({0: 378, 1: 22})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Continous Columns : coronary\_artery\_disease

Counter({1: 364, 2: 34, 0: 2})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Continous Columns : pus\_cell\_clumps

Counter({0: 358, 1: 42})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Continous Columns : sugar

Counter({0.0: 290, 0.45014245014245013: 49, 2.0: 18, 3.0: 14, 4.0: 13, 1.0: 13, 5.0: 3})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Continous Columns : blood\_pressure

Counter({80.0: 116, 70.0: 112, 60.0: 71, 90.0: 53, 100.0: 25, 76.46907216494846: 12, 50.0: 5, 110.0: 3, 140.0: 1, 180.0: 1, 120.0: 1})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Continous Columns : appetite

Counter({0: 318, 1: 82})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Continous Columns : hemoglobin

Counter({12.526436781609195: 52, 15.0: 16, 10.9: 8, 9.8: 7, 11.1: 7, 13.0: 7, 13.6: 7, 11.3: 6, 10.3: 6, 12.0: 6, 13.9: 6, 15.4: 5, 11.2: 5, 10.8: 5, 9.7: 5, 12.6: 5, 7.9: 5, 10.0: 5, 14.0: 5, 14.3: 5, 14.8: 5, 12.2: 4, 12.4: 4, 12.5: 4, 15.2: 4, 9.1: 4, 11.9: 4, 13.5: 4, 16.1: 4, 14.1: 4, 13.2: 4, 13.8: 4, 13.7: 4, 13.4: 4, 17.0: 4, 15.5: 4, 15.8: 4, 9.6: 3, 11.6: 3, 9.5: 3, 9.4: 3, 12.7: 3, 9.9: 3, 10.1: 3, 8.6: 3, 11.0: 3, 15.6: 3, 8.1: 3, 8.3: 3, 10.4: 3, 11.8: 3, 11.4: 3, 11.5: 3, 15.9: 3, 14.5: 3, 16.2: 3, 14.4: 3, 14.2: 3, 16.3: 3, 16.5: 3, 15.7: 3, 16.4: 3, 14.9: 3, 15.3: 3, 17.8: 3, 12.1: 2, 9.3: 2, 10.2: 2, 10.5: 2, 6.0: 2, 11.7: 2, 8.0: 2, 12.3: 2, 8.7: 2, 13.1: 2, 8.8: 2, 13.3: 2, 14.6: 2, 16.9: 2, 16.0: 2, 14.7: 2, 16.6: 2, 16.7: 2, 16.8: 2, 15.1: 2, 17.1: 2, 17.2: 2, 17.4: 2, 5.6: 1, 7.6: 1, 7.7: 1, 12.9: 1, 6.6: 1, 7.5: 1, 4.8: 1, 7.1: 1, 9.2: 1, 6.2: 1, 8.2: 1, 6.1: 1, 8.4: 1, 9.0: 1, 10.6: 1, 10.7: 1, 5.5: 1, 5.8: 1, 6.8: 1, 8.5: 1, 7.3: 1, 12.8: 1, 6.3: 1, 3.1: 1, 17.3: 1, 17.7: 1, 17.5: 1, 17.6: 1})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Continous Columns : blood\_urea

Counter({57.425721784776904: 19, 46.0: 15, 25.0: 13, 19.0: 11, 40.0: 10, 18.0: 9, 50.0: 9, 15.0: 9, 48.0: 9, 26.0: 8, 27.0: 8, 32.0: 8, 49.0: 8, 36.0: 7, 28.0: 7, 20.0: 7, 17.0: 7, 38.0: 7, 16.0: 7, 30.0: 7, 44.0: 7, 31.0: 6, 45.0: 6, 39.0: 6, 29.0: 6, 24.0: 6, 37.0: 6, 22.0: 6, 23.0: 6, 53.0: 5, 55.0: 5, 33.0: 5, 66.0: 5, 35.0: 5, 42.0: 5, 47.0: 4, 51.0: 4, 34.0: 4, 68.0: 4, 41.0: 4, 60.0: 3, 107.0: 3, 80.0: 3, 96.0: 3, 52.0: 3, 106.0: 3, 125.0: 3, 56.0: 2, 54.0: 2, 72.0: 2, 86.0: 2, 90.0: 2, 87.0: 2, 155.0: 2, 153.0: 2, 77.0: 2, 89.0: 2, 111.0: 2, 73.0: 2, 98.0: 2, 82.0: 2, 132.0: 2, 58.0: 2, 10.0: 2, 162.0: 1, 148.0: 1, 180.0: 1, 163.0: 1, 75.0: 1, 65.0: 1, 103.0: 1, 70.0: 1, 202.0: 1, 114.0: 1, 164.0: 1, 142.0: 1, 391.0: 1, 92.0: 1, 139.0: 1, 85.0: 1, 186.0: 1, 217.0: 1, 88.0: 1, 118.0: 1, 50.1: 1, 71.0: 1, 21.0: 1, 219.0: 1, 166.0: 1, 208.0: 1, 176.0: 1, 145.0: 1, 165.0: 1, 322.0: 1, 235.0: 1, 76.0: 1, 113.0: 1, 1.5: 1, 146.0: 1, 133.0: 1, 137.0: 1, 67.0: 1, 115.0: 1, 223.0: 1, 98.6: 1, 158.0: 1, 94.0: 1, 74.0: 1, 150.0: 1, 61.0: 1, 57.0: 1, 95.0: 1, 191.0: 1, 93.0: 1, 241.0: 1, 64.0: 1, 79.0: 1, 215.0: 1, 309.0: 1})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Continous Columns : pedal\_edema

Counter({0: 324, 1: 76})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Continous Columns : blood glucose random

Counter({99.0: 54, 100.0: 9, 93.0: 9, 107.0: 8, 117.0: 6, 140.0: 6, 92.0: 6, 109.0: 6, 131.0: 6, 130.0: 6, 70.0: 5, 114.0: 5, 95.0: 5, 123.0: 5, 124.0: 5, 102.0: 5, 132.0: 5, 104.0: 5, 125.0: 5, 122.0: 5, 121.0: 4, 106.0: 4, 76.0: 4, 91.0: 4, 129.0: 4, 133.0: 4, 94.0: 4, 88.0: 4, 118.0: 4, 139.0: 4, 111.0: 4, 113.0: 4, 120.0: 4, 119.0: 4, 74.0: 3, 108.0: 3, 171.0: 3, 137.0: 3, 79.0: 3, 150.0: 3, 112.0: 3, 127.0: 3, 219.0: 3, 172.0: 3, 89.0: 3, 128.0: 3, 214.0: 3, 105.0: 3, 78.0: 3, 103.0: 3, 82.0: 3, 97.0: 3, 81.0: 3, 138.0: 2, 490.0: 2, 208.0: 2, 98.0: 2, 204.0: 2, 207.0: 2, 144.0: 2, 253.0: 2, 141.0: 2, 86.0: 2, 360.0: 2, 163.0: 2, 158.0: 2, 165.0: 2, 169.0: 2, 210.0: 2, 101.0: 2, 153.0: 2, 213.0: 2, 424.0: 2, 303.0: 2, 192.0: 2, 80.0: 2, 110.0: 2, 96.0: 2, 85.0: 2, 83.0: 2, 75.0: 2, 423.0: 1, 410.0: 1, 380.0: 1, 157.0: 1, 263.0: 1, 173.0: 1, 156.0: 1, 264.0: 1, 159.0: 1, 270.0: 1, 162.0: 1, 246.0: 1, 182.0: 1, 146.0: 1, 425.0: 1, 250.0: 1, 415.0: 1, 251.0: 1, 280.0: 1, 295.0: 1, 298.0: 1, 226.0: 1, 143.0: 1, 115.0: 1, 297.0: 1, 233.0: 1, 294.0: 1, 323.0: 1, 90.0: 1, 308.0: 1, 224.0: 1, 268.0: 1, 256.0: 1, 84.0: 1, 288.0: 1, 273.0: 1, 242.0: 1, 148.0: 1, 160.0: 1, 307.0: 1, 220.0: 1, 447.0: 1, 309.0: 1, 22.0: 1, 261.0: 1, 215.0: 1, 234.0: 1, 352.0: 1, 239.0: 1, 184.0: 1, 252.0: 1, 230.0: 1, 341.0: 1, 255.0: 1, 238.0: 1, 248.0: 1, 241.0: 1, 269.0: 1, 201.0: 1, 203.0: 1, 463.0: 1, 176.0: 1, 116.0: 1, 134.0: 1, 87.0: 1})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Continous Columns : red\_blood\_cells

Counter({1: 353, 0: 47})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Continous Columns : serum\_creatinine

Counter({1.2: 40, 1.1: 24, 1.0: 23, 0.5: 23, 0.7: 22, 0.9: 22, 0.6: 18, 0.8: 17, 3.072454308093995: 17, 2.2: 10, 1.5: 9, 1.7: 9, 1.3: 8, 1.6: 8, 1.8: 7, 1.4: 7, 2.5: 7, 2.8: 7, 1.9: 6, 2.7: 5, 2.1: 5, 2.0: 5, 3.2: 5, 3.3: 5, 3.9: 4, 7.3: 4, 4.0: 3, 2.4: 3, 3.4: 3, 2.9: 3, 5.3: 3, 2.3: 3, 7.2: 2, 4.6: 2, 4.1: 2, 5.2: 2, 6.3: 2, 3.0: 2, 6.1: 2, 6.7: 2, 5.6: 2, 6.5: 2, 4.4: 2, 6.0: 2, 3.8: 1, 24.0: 1, 9.6: 1, 76.0: 1, 7.7: 1, 10.8: 1, 5.9: 1, 3.25: 1, 9.7: 1, 6.4: 1, 32.0: 1, 8.5: 1, 15.0: 1, 3.6: 1, 10.2: 1, 11.5: 1, 12.2: 1, 9.2: 1, 13.8: 1, 16.9: 1, 7.1: 1, 18.0: 1, 13.0: 1, 48.1: 1, 14.2: 1, 16.4: 1, 2.6: 1, 7.5: 1, 4.3: 1, 18.1: 1, 11.8: 1, 9.3: 1, 6.8: 1, 13.5: 1, 12.8: 1, 11.9: 1, 12.0: 1, 13.4: 1, 15.2: 1, 13.3: 1, 0.4: 1})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Continous Columns : id

Counter({0: 1, 1: 1, 2: 1, 3: 1, 4: 1, 5: 1, 6: 1, 7: 1, 8: 1, 9: 1, 10: 1, 11: 1, 12: 1, 13: 1, 14: 1, 15: 1, 16: 1, 17: 1, 18: 1, 19: 1, 20: 1, 21: 1, 22: 1, 23: 1, 24: 1, 25: 1, 26: 1, 27: 1, 28: 1, 29: 1, 30: 1, 31: 1, 32: 1, 33: 1, 34: 1, 35: 1, 36: 1, 37: 1, 38: 1, 39: 1, 40: 1, 41: 1, 42: 1, 43: 1, 44: 1, 45: 1, 46: 1, 47: 1, 48: 1, 49: 1, 50: 1, 51: 1, 52: 1, 53: 1, 54: 1, 55: 1, 56: 1, 57: 1, 58: 1, 59: 1, 60: 1, 61: 1, 62: 1, 63: 1, 64: 1, 65: 1, 66: 1, 67: 1, 68: 1, 69: 1, 70: 1, 71: 1, 72: 1, 73: 1, 74: 1, 75: 1, 76: 1, 77: 1, 78: 1, 79: 1, 80: 1, 81: 1, 82: 1, 83: 1, 84: 1, 85: 1, 86: 1, 87: 1, 88: 1, 89: 1, 90: 1, 91: 1, 92: 1, 93: 1, 94: 1, 95: 1, 96: 1, 97: 1, 98: 1, 99: 1, 100: 1, 101: 1, 102: 1, 103: 1, 104: 1, 105: 1, 106: 1, 107: 1, 108: 1, 109: 1, 110: 1, 111: 1, 112: 1, 113: 1, 114: 1, 115: 1, 116: 1, 117: 1, 118: 1, 119: 1, 120: 1, 121: 1, 122: 1, 123: 1, 124: 1, 125: 1, 126: 1, 127: 1, 128: 1, 129: 1, 130: 1, 131: 1, 132: 1, 133: 1, 134: 1, 135: 1, 136: 1, 137: 1, 138: 1, 139: 1, 140: 1, 141: 1, 142: 1, 143: 1, 144: 1, 145: 1, 146: 1, 147: 1, 148: 1, 149: 1, 150: 1, 151: 1, 152: 1, 153: 1, 154: 1, 155: 1, 156: 1, 157: 1, 158: 1, 159: 1, 160: 1, 161: 1, 162: 1, 163: 1, 164: 1, 165: 1, 166: 1, 167: 1, 168: 1, 169: 1, 170: 1, 171: 1, 172: 1, 173: 1, 174: 1, 175: 1, 176: 1, 177: 1, 178: 1, 179: 1, 180: 1, 181: 1, 182: 1, 183: 1, 184: 1, 185: 1, 186: 1, 187: 1, 188: 1, 189: 1, 190: 1, 191: 1, 192: 1, 193: 1, 194: 1, 195: 1, 196: 1, 197: 1, 198: 1, 199: 1, 200: 1, 201: 1, 202: 1, 203: 1, 204: 1, 205: 1, 206: 1, 207: 1, 208: 1, 209: 1, 210: 1, 211: 1, 212: 1, 213: 1, 214: 1, 215: 1, 216: 1, 217: 1, 218: 1, 219: 1, 220: 1, 221: 1, 222: 1, 223: 1, 224: 1, 225: 1, 226: 1, 227: 1, 228: 1, 229: 1, 230: 1, 231: 1, 232: 1, 233: 1, 234: 1, 235: 1, 236: 1, 237: 1, 238: 1, 239: 1, 240: 1, 241: 1, 242: 1, 243: 1, 244: 1, 245: 1, 246: 1, 247: 1, 248: 1, 249: 1, 250: 1, 251: 1, 252: 1, 253: 1, 254: 1, 255: 1, 256: 1, 257: 1, 258: 1, 259: 1, 260: 1, 261: 1, 262: 1, 263: 1, 264: 1, 265: 1, 266: 1, 267: 1, 268: 1, 269: 1, 270: 1, 271: 1, 272: 1, 273: 1, 274: 1, 275: 1, 276: 1, 277: 1, 278: 1, 279: 1, 280: 1, 281: 1, 282: 1, 283: 1, 284: 1, 285: 1, 286: 1, 287: 1, 288: 1, 289: 1, 290: 1, 291: 1, 292: 1, 293: 1, 294: 1, 295: 1, 296: 1, 297: 1, 298: 1, 299: 1, 300: 1, 301: 1, 302: 1, 303: 1, 304: 1, 305: 1, 306: 1, 307: 1, 308: 1, 309: 1, 310: 1, 311: 1, 312: 1, 313: 1, 314: 1, 315: 1, 316: 1, 317: 1, 318: 1, 319: 1, 320: 1, 321: 1, 322: 1, 323: 1, 324: 1, 325: 1, 326: 1, 327: 1, 328: 1, 329: 1, 330: 1, 331: 1, 332: 1, 333: 1, 334: 1, 335: 1, 336: 1, 337: 1, 338: 1, 339: 1, 340: 1, 341: 1, 342: 1, 343: 1, 344: 1, 345: 1, 346: 1, 347: 1, 348: 1, 349: 1, 350: 1, 351: 1, 352: 1, 353: 1, 354: 1, 355: 1, 356: 1, 357: 1, 358: 1, 359: 1, 360: 1, 361: 1, 362: 1, 363: 1, 364: 1, 365: 1, 366: 1, 367: 1, 368: 1, 369: 1, 370: 1, 371: 1, 372: 1, 373: 1, 374: 1, 375: 1, 376: 1, 377: 1, 378: 1, 379: 1, 380: 1, 381: 1, 382: 1, 383: 1, 384: 1, 385: 1, 386: 1, 387: 1, 388: 1, 389: 1, 390: 1, 391: 1, 392: 1, 393: 1, 394: 1, 395: 1, 396: 1, 397: 1, 398: 1, 399: 1})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Continous Columns : diabetesmellitus

Counter({3: 260, 4: 134, 0: 3, 1: 2, 2: 1})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Continous Columns : sodium

Counter({137.52875399361022: 87, 135.0: 40, 140.0: 25, 141.0: 22, 139.0: 21, 142.0: 20, 138.0: 20, 137.0: 19, 136.0: 17, 150.0: 17, 147.0: 13, 145.0: 11, 132.0: 10, 146.0: 10, 131.0: 9, 144.0: 9, 133.0: 8, 130.0: 7, 134.0: 6, 143.0: 4, 127.0: 3, 124.0: 3, 114.0: 2, 125.0: 2, 128.0: 2, 122.0: 2, 113.0: 2, 120.0: 2, 111.0: 1, 104.0: 1, 4.5: 1, 129.0: 1, 163.0: 1, 126.0: 1, 115.0: 1})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Continous Columns : albumin

Counter({0.0: 199, 1.0169491525423728: 46, 1.0: 44, 2.0: 43, 3.0: 43, 4.0: 24, 5.0: 1})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Continous Columns : age

Counter({60.0: 19, 65.0: 17, 48.0: 12, 50.0: 12, 55.0: 12, 47.0: 11, 62.0: 10, 45.0: 10, 54.0: 10, 59.0: 10, 56.0: 10, 61.0: 9, 51.48337595907928: 9, 70.0: 9, 46.0: 9, 34.0: 9, 68.0: 8, 73.0: 8, 64.0: 8, 71.0: 8, 57.0: 8, 63.0: 7, 72.0: 7, 67.0: 7, 30.0: 7, 42.0: 6, 69.0: 6, 35.0: 6, 44.0: 6, 43.0: 6, 33.0: 6, 51.0: 5, 52.0: 5, 53.0: 5, 75.0: 5, 76.0: 5, 58.0: 5, 41.0: 5, 66.0: 5, 24.0: 4, 40.0: 4, 39.0: 4, 80.0: 4, 23.0: 4, 74.0: 3, 38.0: 3, 17.0: 3, 8.0: 3, 32.0: 3, 37.0: 3, 25.0: 3, 29.0: 3, 21.0: 2, 15.0: 2, 5.0: 2, 12.0: 2, 49.0: 2, 19.0: 2, 36.0: 2, 20.0: 2, 28.0: 2, 7.0: 1, 82.0: 1, 11.0: 1, 26.0: 1, 81.0: 1, 14.0: 1, 27.0: 1, 83.0: 1, 4.0: 1, 3.0: 1, 6.0: 1, 90.0: 1, 78.0: 1, 2.0: 1, 22.0: 1, 79.0: 1})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

contcols.remove('specific\_gravity')

contcols.remove('albumin')

contcols.remove('sugar')

print(contcols)

{'hypertension', 'anemia', 'potassium', 'bacteria', 'coronary\_artery\_disease', 'pus\_cell\_clumps', 'blood\_pressure', 'appetite', 'hemoglobin', 'blood\_urea', 'pedal\_edema', 'blood glucose random', 'red\_blood\_cells', 'serum\_creatinine', 'id', 'diabetesmellitus', 'sodium', 'age', 'class', 'pus\_cell'}

contcols.add('red\_blood\_cell\_count')

contcols.add('packed\_cell\_volume')

contcols.add('white\_blood\_cell\_count')

print(contcols)

{'hypertension', 'anemia', 'white\_blood\_cell\_count', 'potassium', 'bacteria', 'coronary\_artery\_disease', 'pus\_cell\_clumps', 'blood\_pressure', 'appetite', 'hemoglobin', 'blood\_urea', 'pedal\_edema', 'blood glucose random', 'red\_blood\_cells', 'serum\_creatinine', 'packed\_cell\_volume', 'red\_blood\_cell\_count', 'id', 'diabetesmellitus', 'sodium', 'age', 'class', 'pus\_cell'}

catcols.add('specific\_gravity')

catcols.add('albumin')

catcols.add('sugar')

print(catcols)

{'diabetesmellitus', 'coronary\_artery\_disease', 'hypertension', 'pedal\_edema', 'anemia', 'albumin', 'red\_blood\_cells', 'pus\_cell\_clumps', 'sugar', 'specific\_gravity', 'class', 'appetite', 'bacteria', 'pus\_cell'}

**data['coronary\_artery\_disease'] = data.coronary\_artery\_disease.replace('\tno','no')**

**c(data['coronary\_artery\_disease'])**

Counter({1: 364, 2: 34, 0: 2})

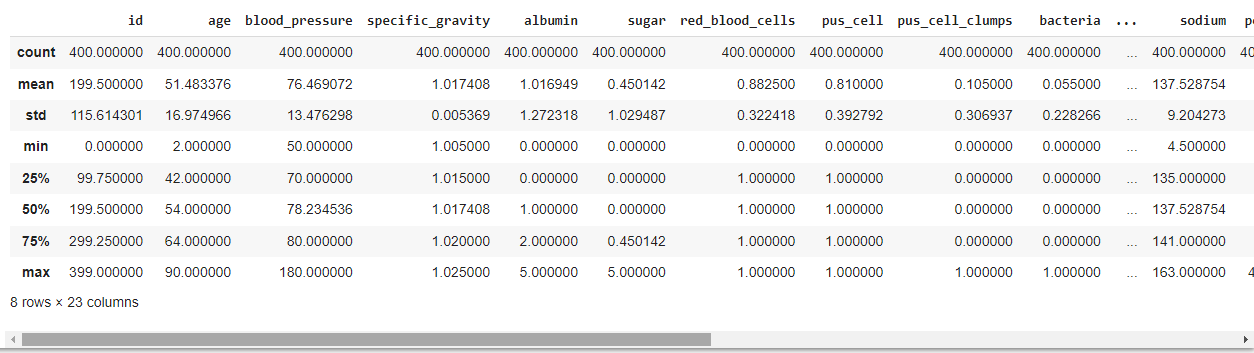
data['diabetesmellitus']=data.diabetesmellitus.replace('\tno','no')

c(data['diabetesmellitus'])

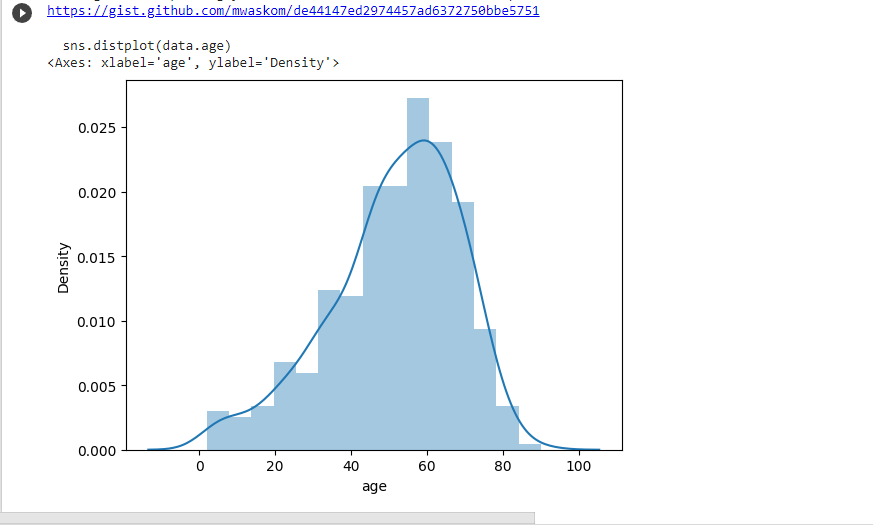
Counter({4: 134, 3: 260, 2: 1, 0: 3, 1: 2})

Milestone 3

data.describe()



sns.distplot(data.age)



import matplotlib.pyplot as plt

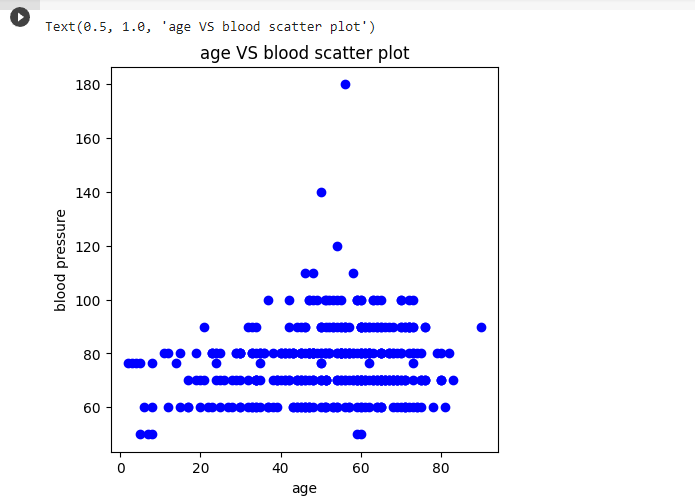
fig=plt.figure(figsize=(5,5))

plt.scatter(data['age'],data['blood\_pressure'],color='blue')

plt.xlabel('age')

plt.ylabel('blood pressure')

plt.title("age VS blood scatter plot")



plt.figure(figsize=(20,15), facecolor='white')

plotnumber = 1

for column in contcols:

  if plotnumber<=11:

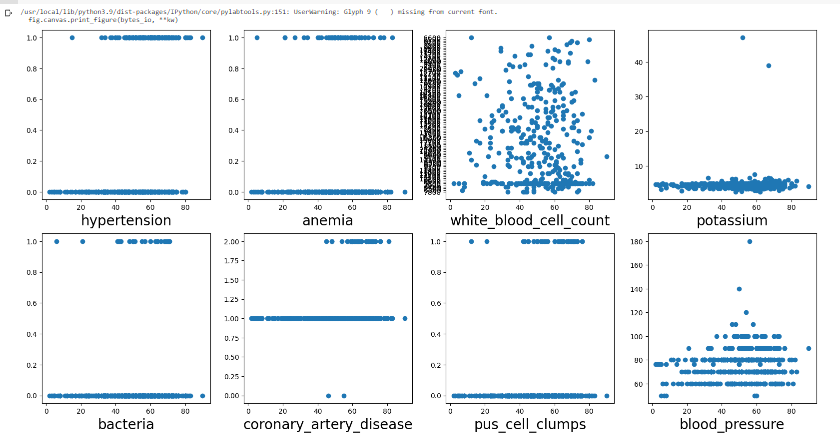
    ax = plt.subplot(3,4,plotnumber)

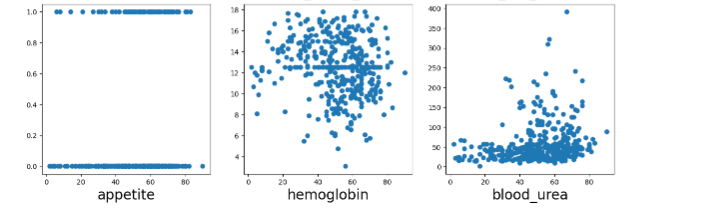
    plt.scatter(data['age'],data[column])

    plt.xlabel(column,fontsize=20)

  plotnumber+=1

plt.show()





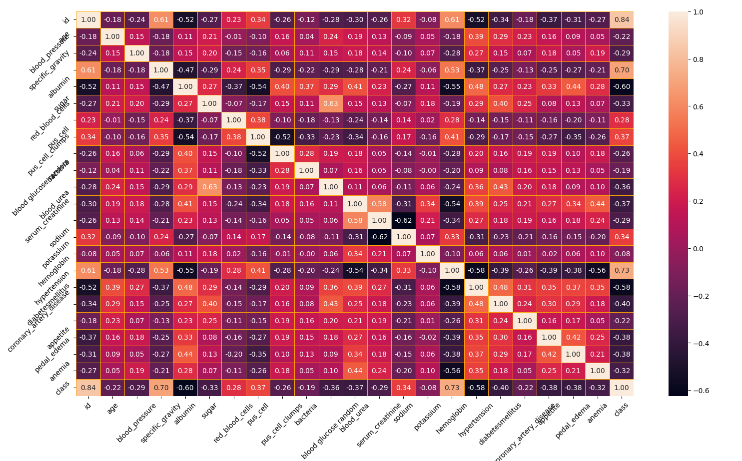
f,ax=plt.subplots(figsize=(18,10))

sns.heatmap(data.corr(),annot=True,fmt=".2f",ax=ax,linewidths= 0.5,linecolor="orange")

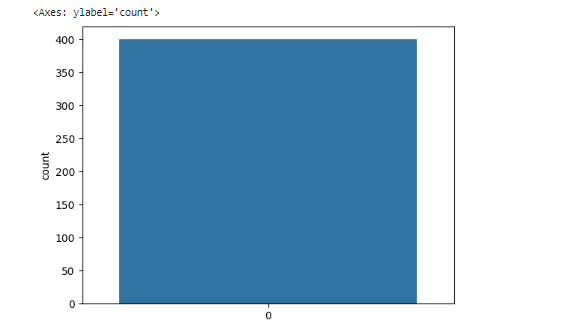
plt.xticks(rotation=45)

plt.yticks(rotation=45)

plt.show()



sns.countplot(data['class'])



from sklearn.preprocessing import StandardScaler

sc=StandardScaler()

selcols=['red\_blood\_cells','pus\_cell','blood glucose random','blood\_urea','pedal\_edema','anemia','diabetesmellitus','coronary\_artery\_disease']

x=pd.DataFrame(data,columns=selcols)

y=pd.DataFrame(data,columns=['class'])

print(x.shape)

print(y.shape)

(400, 8)

(400, 1)

from sklearn.model\_selection import train\_test\_split

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.2,random\_state=2)

Milestone 4

import tensorflow

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

classification = Sequential()

classification.add(Dense(30,activation='relu'))

classification.add(Dense(128,activation='relu'))

classification.add(Dense(64,activation='relu'))

classification.add(Dense(32,activation='relu'))

classification.add(Dense(1,activation='sigmoid'))

classification.compile(optimizer='adam',loss='binary\_crossentropy',

metrics=['accuracy'])

classification.fit(x\_train,y\_train,batch\_size=10,

validation\_split=0.2,epochs=100)

Epoch 1/100

26/26 [==============================] - 3s 19ms/step - loss: 1.9170 - accuracy: 0.2422 - val\_loss: 0.5725 - val\_accuracy: 0.4688

Epoch 2/100

26/26 [==============================] - 0s 6ms/step - loss: 1.5366 - accuracy: 0.2305 - val\_loss: 0.6411 - val\_accuracy: 0.2188

Epoch 3/100

26/26 [==============================] - 0s 6ms/step - loss: 0.5509 - accuracy: 0.3164 - val\_loss: 0.5074 - val\_accuracy: 0.2031

Epoch 4/100

26/26 [==============================] - 0s 6ms/step - loss: 0.3913 - accuracy: 0.2266 - val\_loss: 0.4190 - val\_accuracy: 0.2031

Epoch 5/100

26/26 [==============================] - 0s 6ms/step - loss: 0.4187 - accuracy: 0.2734 - val\_loss: 1.0563 - val\_accuracy: 0.1875

Epoch 6/100

26/26 [==============================] - 0s 6ms/step - loss: 0.6825 - accuracy: 0.2891 - val\_loss: 1.0657 - val\_accuracy: 0.1875

Epoch 7/100

26/26 [==============================] - 0s 6ms/step - loss: 0.6412 - accuracy: 0.2656 - val\_loss: 0.5097 - val\_accuracy: 0.2344

Epoch 8/100

26/26 [==============================] - 0s 6ms/step - loss: 0.5948 - accuracy: 0.2461 - val\_loss: 0.8240 - val\_accuracy: 0.1875

Epoch 9/100

26/26 [==============================] - 0s 5ms/step - loss: 0.2572 - accuracy: 0.2930 - val\_loss: 0.2896 - val\_accuracy: 0.2500

Epoch 10/100

26/26 [==============================] - 0s 6ms/step - loss: 0.3208 - accuracy: 0.2656 - val\_loss: 1.1173 - val\_accuracy: 0.1875

Epoch 11/100

26/26 [==============================] - 0s 5ms/step - loss: 0.1900 - accuracy: 0.2656 - val\_loss: 0.5179 - val\_accuracy: 0.2031

Epoch 12/100

26/26 [==============================] - 0s 5ms/step - loss: -0.0150 - accuracy: 0.2930 - val\_loss: 0.5630 - val\_accuracy: 0.2031

Epoch 13/100

26/26 [==============================] - 0s 6ms/step - loss: 0.3870 - accuracy: 0.2383 - val\_loss: 0.3150 - val\_accuracy: 0.2812

Epoch 14/100

26/26 [==============================] - 0s 5ms/step - loss: 0.0302 - accuracy: 0.3008 - val\_loss: 0.2780 - val\_accuracy: 0.2344

Epoch 15/100

26/26 [==============================] - 0s 5ms/step - loss: -0.2033 - accuracy: 0.2305 - val\_loss: 0.1754 - val\_accuracy: 0.3125

Epoch 16/100

26/26 [==============================] - 0s 6ms/step - loss: -0.3200 - accuracy: 0.3008 - val\_loss: 0.3586 - val\_accuracy: 0.2188

Epoch 17/100

26/26 [==============================] - 0s 6ms/step - loss: -0.3454 - accuracy: 0.2656 - val\_loss: 0.7526 - val\_accuracy: 0.1875

Epoch 18/100

26/26 [==============================] - 0s 5ms/step - loss: -0.2979 - accuracy: 0.2969 - val\_loss: 0.0144 - val\_accuracy: 0.4375

Epoch 19/100

26/26 [==============================] - 0s 5ms/step - loss: 0.2707 - accuracy: 0.3164 - val\_loss: -0.1211 - val\_accuracy: 0.2656

Epoch 20/100

26/26 [==============================] - 0s 6ms/step - loss: -0.6757 - accuracy: 0.2695 - val\_loss: 0.5293 - val\_accuracy: 0.2344

Epoch 21/100

26/26 [==============================] - 0s 6ms/step - loss: -1.1019 - accuracy: 0.3047 - val\_loss: -0.2022 - val\_accuracy: 0.2812

Epoch 22/100

26/26 [==============================] - 0s 5ms/step - loss: -1.7570 - accuracy: 0.2969 - val\_loss: 0.5081 - val\_accuracy: 0.2188

Epoch 23/100

26/26 [==============================] - 0s 4ms/step - loss: -1.9639 - accuracy: 0.2852 - val\_loss: -0.4449 - val\_accuracy: 0.2969

Epoch 24/100

26/26 [==============================] - 0s 4ms/step - loss: -2.9232 - accuracy: 0.2930 - val\_loss: -0.2926 - val\_accuracy: 0.3438

Epoch 25/100

26/26 [==============================] - 0s 4ms/step - loss: -3.9678 - accuracy: 0.3164 - val\_loss: 0.4535 - val\_accuracy: 0.2344

Epoch 26/100

26/26 [==============================] - 0s 4ms/step - loss: -6.0380 - accuracy: 0.2891 - val\_loss: -1.1717 - val\_accuracy: 0.3438

Epoch 27/100

26/26 [==============================] - 0s 4ms/step - loss: -8.9568 - accuracy: 0.3398 - val\_loss: -1.0845 - val\_accuracy: 0.2344

Epoch 28/100

26/26 [==============================] - 0s 4ms/step - loss: -11.3656 - accuracy: 0.3047 - val\_loss: -5.0146 - val\_accuracy: 0.2656

Epoch 29/100

26/26 [==============================] - 0s 4ms/step - loss: -21.1796 - accuracy: 0.3125 - val\_loss: -8.4466 - val\_accuracy: 0.3750

Epoch 30/100

26/26 [==============================] - 0s 4ms/step - loss: -32.7403 - accuracy: 0.3359 - val\_loss: 4.2639 - val\_accuracy: 0.2031

Epoch 31/100

26/26 [==============================] - 0s 4ms/step - loss: -47.5475 - accuracy: 0.2930 - val\_loss: -20.2726 - val\_accuracy: 0.3438

Epoch 32/100

26/26 [==============================] - 0s 4ms/step - loss: -35.1881 - accuracy: 0.3242 - val\_loss: 13.0977 - val\_accuracy: 0.2031

Epoch 33/100

26/26 [==============================] - 0s 5ms/step - loss: -95.2867 - accuracy: 0.3008 - val\_loss: -51.0965 - val\_accuracy: 0.3750

Epoch 34/100

26/26 [==============================] - 0s 4ms/step - loss: -167.9017 - accuracy: 0.3008 - val\_loss: -87.6063 - val\_accuracy: 0.3438

Epoch 35/100

26/26 [==============================] - 0s 4ms/step - loss: -271.6215 - accuracy: 0.3164 - val\_loss: -101.2619 - val\_accuracy: 0.3438

Epoch 36/100

26/26 [==============================] - 0s 4ms/step - loss: -390.8019 - accuracy: 0.3125 - val\_loss: -184.3765 - val\_accuracy: 0.3281

Epoch 37/100

26/26 [==============================] - 0s 4ms/step - loss: -676.4297 - accuracy: 0.3203 - val\_loss: -208.3688 - val\_accuracy: 0.2969

Epoch 38/100

26/26 [==============================] - 0s 4ms/step - loss: -674.3459 - accuracy: 0.3047 - val\_loss: -285.3145 - val\_accuracy: 0.4531

Epoch 39/100

26/26 [==============================] - 0s 4ms/step - loss: -1415.9741 - accuracy: 0.3281 - val\_loss: -602.9527 - val\_accuracy: 0.3281

Epoch 40/100

26/26 [==============================] - 0s 4ms/step - loss: -2186.7444 - accuracy: 0.3125 - val\_loss: -626.8016 - val\_accuracy: 0.2500

Epoch 41/100

26/26 [==============================] - 0s 4ms/step - loss: -2038.4053 - accuracy: 0.3047 - val\_loss: 224.6841 - val\_accuracy: 0.2188

Epoch 42/100

26/26 [==============================] - 0s 5ms/step - loss: -4294.9565 - accuracy: 0.3203 - val\_loss: -773.7548 - val\_accuracy: 0.2344

Epoch 43/100

26/26 [==============================] - 0s 4ms/step - loss: -5427.2393 - accuracy: 0.3086 - val\_loss: -1853.7975 - val\_accuracy: 0.2812

Epoch 44/100

26/26 [==============================] - 0s 4ms/step - loss: -6827.4507 - accuracy: 0.2930 - val\_loss: -2729.7163 - val\_accuracy: 0.4062

Epoch 45/100

26/26 [==============================] - 0s 4ms/step - loss: -7563.3521 - accuracy: 0.3477 - val\_loss: -4681.4121 - val\_accuracy: 0.3438

Epoch 46/100

26/26 [==============================] - 0s 4ms/step - loss: -11600.1230 - accuracy: 0.3008 - val\_loss: -5234.6426 - val\_accuracy: 0.3906

Epoch 47/100

26/26 [==============================] - 0s 4ms/step - loss: -14210.9482 - accuracy: 0.3125 - val\_loss: -6712.8218 - val\_accuracy: 0.3281

Epoch 48/100

26/26 [==============================] - 0s 4ms/step - loss: -19682.2637 - accuracy: 0.3398 - val\_loss: -560.5317 - val\_accuracy: 0.2188

Epoch 49/100

26/26 [==============================] - 0s 4ms/step - loss: -17378.3965 - accuracy: 0.2891 - val\_loss: -9983.1055 - val\_accuracy: 0.3281

Epoch 50/100

26/26 [==============================] - 0s 4ms/step - loss: -26196.3574 - accuracy: 0.3281 - val\_loss: -12007.9863 - val\_accuracy: 0.3281

Epoch 51/100

26/26 [==============================] - 0s 5ms/step - loss: -31911.7598 - accuracy: 0.3125 - val\_loss: -12461.4434 - val\_accuracy: 0.4219

Epoch 52/100

26/26 [==============================] - 0s 4ms/step - loss: -41101.1367 - accuracy: 0.3398 - val\_loss: -16585.0547 - val\_accuracy: 0.3125

Epoch 53/100

26/26 [==============================] - 0s 4ms/step - loss: -48644.7617 - accuracy: 0.3203 - val\_loss: -18620.9883 - val\_accuracy: 0.3125

Epoch 54/100

26/26 [==============================] - 0s 4ms/step - loss: -58008.2969 - accuracy: 0.2852 - val\_loss: -25147.3516 - val\_accuracy: 0.3438

Epoch 55/100

26/26 [==============================] - 0s 4ms/step - loss: -69741.4141 - accuracy: 0.3281 - val\_loss: -31293.5098 - val\_accuracy: 0.3438

Epoch 56/100

26/26 [==============================] - 0s 4ms/step - loss: -82555.2578 - accuracy: 0.3281 - val\_loss: -30401.0469 - val\_accuracy: 0.2812

Epoch 57/100

26/26 [==============================] - 0s 4ms/step - loss: -97163.6406 - accuracy: 0.2930 - val\_loss: -41751.3516 - val\_accuracy: 0.3125

Epoch 58/100

26/26 [==============================] - 0s 4ms/step - loss: -115082.7422 - accuracy: 0.3164 - val\_loss: -48524.1133 - val\_accuracy: 0.3438

Epoch 59/100

26/26 [==============================] - 0s 4ms/step - loss: -122187.7734 - accuracy: 0.3359 - val\_loss: -60370.7695 - val\_accuracy: 0.3750

Epoch 60/100

26/26 [==============================] - 0s 4ms/step - loss: -144334.0625 - accuracy: 0.2969 - val\_loss: -71876.5156 - val\_accuracy: 0.3125

Epoch 61/100

26/26 [==============================] - 0s 4ms/step - loss: -159226.1562 - accuracy: 0.3320 - val\_loss: -63945.5625 - val\_accuracy: 0.2969

Epoch 62/100

26/26 [==============================] - 0s 4ms/step - loss: -187995.3281 - accuracy: 0.3047 - val\_loss: -81384.0859 - val\_accuracy: 0.2969

Epoch 63/100

26/26 [==============================] - 0s 4ms/step - loss: -221094.8125 - accuracy: 0.3125 - val\_loss: -95184.6328 - val\_accuracy: 0.3906

Epoch 64/100

26/26 [==============================] - 0s 5ms/step - loss: -235971.0312 - accuracy: 0.3125 - val\_loss: -88253.0938 - val\_accuracy: 0.4219

Epoch 65/100

26/26 [==============================] - 0s 4ms/step - loss: -261264.7031 - accuracy: 0.3125 - val\_loss: -102569.3047 - val\_accuracy: 0.4219

Epoch 66/100

26/26 [==============================] - 0s 4ms/step - loss: -295232.5312 - accuracy: 0.3320 - val\_loss: -131853.0625 - val\_accuracy: 0.3438

Epoch 67/100

26/26 [==============================] - 0s 4ms/step - loss: -346577.7500 - accuracy: 0.3203 - val\_loss: -150731.5469 - val\_accuracy: 0.3438

Epoch 68/100

26/26 [==============================] - 0s 5ms/step - loss: -378144.5938 - accuracy: 0.3203 - val\_loss: -173653.5000 - val\_accuracy: 0.3438

Epoch 69/100

26/26 [==============================] - 0s 4ms/step - loss: -437493.9062 - accuracy: 0.2969 - val\_loss: -185766.8281 - val\_accuracy: 0.4062

Epoch 70/100

26/26 [==============================] - 0s 4ms/step - loss: -476310.2188 - accuracy: 0.3398 - val\_loss: -205862.0469 - val\_accuracy: 0.2969

Epoch 71/100

26/26 [==============================] - 0s 5ms/step - loss: -512368.0938 - accuracy: 0.2969 - val\_loss: -248088.7344 - val\_accuracy: 0.3750

Epoch 72/100

26/26 [==============================] - 0s 4ms/step - loss: -592021.3125 - accuracy: 0.3281 - val\_loss: -258929.8750 - val\_accuracy: 0.2969

Epoch 73/100

26/26 [==============================] - 0s 4ms/step - loss: -630349.8750 - accuracy: 0.2930 - val\_loss: -294555.1250 - val\_accuracy: 0.4062

Epoch 74/100

26/26 [==============================] - 0s 4ms/step - loss: -712402.6250 - accuracy: 0.3047 - val\_loss: -323334.4062 - val\_accuracy: 0.3438

Epoch 75/100

26/26 [==============================] - 0s 4ms/step - loss: -801394.6875 - accuracy: 0.3398 - val\_loss: -327954.4375 - val\_accuracy: 0.2969

Epoch 76/100

26/26 [==============================] - 0s 4ms/step - loss: -861479.0000 - accuracy: 0.3047 - val\_loss: -346356.1562 - val\_accuracy: 0.2969

Epoch 77/100

26/26 [==============================] - 0s 4ms/step - loss: -934881.6875 - accuracy: 0.3164 - val\_loss: -435822.5625 - val\_accuracy: 0.3281

Epoch 78/100

26/26 [==============================] - 0s 4ms/step - loss: -1051332.7500 - accuracy: 0.3125 - val\_loss: -502227.8125 - val\_accuracy: 0.3281

Epoch 79/100

26/26 [==============================] - 0s 4ms/step - loss: -1146265.2500 - accuracy: 0.3086 - val\_loss: -526691.1250 - val\_accuracy: 0.3281

Epoch 80/100

26/26 [==============================] - 0s 4ms/step - loss: -1269717.7500 - accuracy: 0.3164 - val\_loss: -534240.1250 - val\_accuracy: 0.2969

Epoch 81/100

26/26 [==============================] - 0s 4ms/step - loss: -1367591.5000 - accuracy: 0.2930 - val\_loss: -630709.0625 - val\_accuracy: 0.3281

Epoch 82/100

26/26 [==============================] - 0s 4ms/step - loss: -1376455.7500 - accuracy: 0.3594 - val\_loss: -570607.4375 - val\_accuracy: 0.2969

Epoch 83/100

26/26 [==============================] - 0s 5ms/step - loss: -1475926.5000 - accuracy: 0.3320 - val\_loss: -733813.8750 - val\_accuracy: 0.3281

Epoch 84/100

26/26 [==============================] - 0s 4ms/step - loss: -1736667.2500 - accuracy: 0.2930 - val\_loss: -786050.1875 - val\_accuracy: 0.3281

Epoch 85/100

26/26 [==============================] - 0s 4ms/step - loss: -1801545.3750 - accuracy: 0.3125 - val\_loss: -843021.3125 - val\_accuracy: 0.3281

Epoch 86/100

26/26 [==============================] - 0s 4ms/step - loss: -2006606.0000 - accuracy: 0.3555 - val\_loss: -761527.0000 - val\_accuracy: 0.2969

Epoch 87/100

26/26 [==============================] - 0s 4ms/step - loss: -1988779.5000 - accuracy: 0.2969 - val\_loss: -917298.3125 - val\_accuracy: 0.4062

Epoch 88/100

26/26 [==============================] - 0s 4ms/step - loss: -2198979.5000 - accuracy: 0.2891 - val\_loss: -997340.6250 - val\_accuracy: 0.4062

Epoch 89/100

26/26 [==============================] - 0s 5ms/step - loss: -2370956.7500 - accuracy: 0.3125 - val\_loss: -1072312.0000 - val\_accuracy: 0.3594

Epoch 90/100

26/26 [==============================] - 0s 4ms/step - loss: -2573623.7500 - accuracy: 0.3281 - val\_loss: -1115827.0000 - val\_accuracy: 0.3125

Epoch 91/100

26/26 [==============================] - 0s 4ms/step - loss: -2788516.0000 - accuracy: 0.3281 - val\_loss: -1196266.7500 - val\_accuracy: 0.3125

Epoch 92/100

26/26 [==============================] - 0s 4ms/step - loss: -2952012.5000 - accuracy: 0.3047 - val\_loss: -1324442.0000 - val\_accuracy: 0.3125

Epoch 93/100

26/26 [==============================] - 0s 4ms/step - loss: -3128123.0000 - accuracy: 0.3516 - val\_loss: -1426936.6250 - val\_accuracy: 0.3125

Epoch 94/100

26/26 [==============================] - 0s 4ms/step - loss: -3381758.5000 - accuracy: 0.3164 - val\_loss: -1401845.7500 - val\_accuracy: 0.2969

Epoch 95/100

26/26 [==============================] - 0s 4ms/step - loss: -3559427.0000 - accuracy: 0.2891 - val\_loss: -1621283.5000 - val\_accuracy: 0.3594

Epoch 96/100

26/26 [==============================] - 0s 4ms/step - loss: -3747214.0000 - accuracy: 0.3398 - val\_loss: -1494436.6250 - val\_accuracy: 0.2969

Epoch 97/100

26/26 [==============================] - 0s 4ms/step - loss: -3934591.7500 - accuracy: 0.3086 - val\_loss: -1859112.1250 - val\_accuracy: 0.3281

Epoch 98/100

26/26 [==============================] - 0s 4ms/step - loss: -4253708.0000 - accuracy: 0.3242 - val\_loss: -1956884.5000 - val\_accuracy: 0.3281

Epoch 99/100

26/26 [==============================] - 0s 4ms/step - loss: -4342637.5000 - accuracy: 0.3008 - val\_loss: -1859389.1250 - val\_accuracy: 0.4219

Epoch 100/100

26/26 [==============================] - 0s 4ms/step - loss: -4647518.5000 - accuracy: 0.3086 - val\_loss: -2204960.2500 - val\_accuracy: 0.3281

<keras.callbacks.History at 0x7fac27917af0>

from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n\_estimators=10,criterion='entropy')

rfc.fit(x\_train,y\_train)

C:\Users\ELCOT\Pictures\Screenshots\Screenshot (101).png

y\_predict = rfc.predict(x\_test)

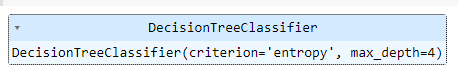
y\_predict\_train = rfc.predict(x\_train)

from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier(max\_depth=4,splitter='best',criterion=

'entropy')

dtc.fit(x\_train,y\_train)



y\_predict = dtc.predict(x\_test)

y\_predict

array([0, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0, 2, 2, 0, 0, 0, 2, 2, 0, 2, 2, 0, 2, 0, 2, 0, 0, 2, 0, 0, 2, 0, 0, 0, 0, 2, 0, 0, 2, 0, 2, 0, 0, 0, 2, 0, 2, 2, 2, 0, 0, 0, 2, 0, 0, 0, 2, 2, 0, 0, 2, 2, 0, 0, 0, 0, 2, 0, 2, 2, 0, 2, 2, 0, 0, 0, 2, 2, 2])

y\_predict\_train = dtc.predict(x\_train)

from sklearn.linear\_model import LogisticRegression

lgr = LogisticRegression()

lgr.fit(x\_train,y\_train)

C:\Users\pc\Pictures\Screenshots\Screenshot (2).png

from sklearn.metrics import accuracy\_score,classification\_report

y\_predict = lgr.predict(x\_test)

y\_pred = lgr.predict([[1,1,121.000000,36.0,0,0,1,0]])

print(y\_pred)

(y\_pred)

[2]

array([2])

y\_pred = dtc.predict([[1,1,121.000000,36.0,0,0,1,0]])

print(y\_pred)

(y\_pred)

[2]

array([2])

y\_pred = rfc.predict([[1,1,121.000000,36.0,0,0,1,0]])

print(y\_pred)

(y\_pred)

[2]

array([2])

classification.save("ckd.h5")

y\_pred = classification.predict(x\_test)

y\_pred

array([[1.], [1.], [0.], [1.], [1.], [1.], [0.], [0.], [1.], [0.], [0.], [1.], [1.], [1.], [0.], [0.], [0.], [1.], [1.], [0.], [1.], [1.], [0.], [1.], [0.], [1.], [0.], [1.], [1.], [1.], [0.], [1.], [1.], [0.], [0.], [1.], [1.], [1.], [1.], [1.], [1.], [1.], [0.], [0.], [0.], [1.], [0.], [1.], [1.], [1.], [1.], [0.], [0.], [1.], [0.], [1.], [0.], [1.], [1.], [0.], [0.], [1.], [1.], [0.], [1.], [1.], [1.], [1.], [1.], [1.], [1.], [0.], [1.], [1.], [1.], [0.], [0.], [1.], [1.], [1.]], dtype=float32)

y\_pred = (y\_pred > 0.5)

y\_pred

array([[ True], [ True], [False], [ True], [ True], [ True], [False], [False], [ True], [False], [False], [ True], [ True], [ True], [False], [False], [False], [ True], [ True], [False], [ True], [ True], [False], [ True], [False], [ True], [False], [ True], [ True], [ True], [False], [ True], [ True], [False], [False], [ True], [ True], [ True], [ True], [ True], [ True], [ True], [False], [False], [False], [ True], [False], [ True], [ True], [ True], [ True], [False], [False], [ True], [False], [ True], [False], [ True], [ True], [False], [False], [ True], [ True], [False], [ True], [ True], [ True], [ True], [ True], [ True], [ True], [False], [ True], [ True], [ True], [False], [False], [ True], [ True], [ True]])

def predict\_exit(sample\_value):

  sample\_value = np.array(sample\_value)

  sample\_value = sample\_value.reshape(1,-1)

  sample\_value = sc.transform(sample\_value)

  return classifier.predict(sample\_value)

test=classification.predict([[1,1,121.000000,36.0,0,0,1,0]])

if test==1:

     print('prediction: High chance of CKD!')

else:

    print('prediction: Low chance of CKD.')

prediction: High chance of CKD!

Milestone5

from sklearn import model\_selection

dfs = []

models = [

          ('LogReg', LogisticRegression()),

          ('RF', RandomForestClassifier()),

          ('DecisionTree', DecisionTreeClassifier()),

         ]

results = []

names = []

scoring = ['accuracy','precision\_weighted','recall\_weighted','f1\_weighted','roc\_auc']

target\_names = ['NO CKD','CKD','CKD']

for name, model in models:

        kfold = model\_selection.KFold(n\_splits=5, shuffle=True, random\_state=90210)

        cv\_results = model\_selection.cross\_validate(model, x\_train, y\_train, cv=kfold, scoring=scoring)

        clf = model.fit(x\_train, y\_train)

        y\_pred = clf.predict(x\_test)

        print(names)

        print(classification\_report(y\_test, y\_pred, target\_names=target\_names))

        results.append(cv\_results)

        names.append(names)

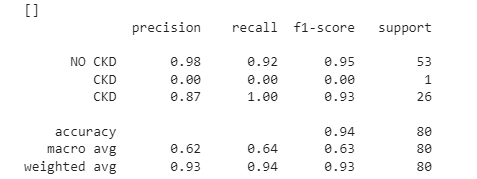
        this\_df = pd.DataFrame(cv\_results)

        this\_df['model'] = name

dfs.append(this\_df)

final = pd.concat(dfs, ignore\_index=True)

print(final)



from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_predict)

cm

array([[49, 0, 4], [ 1, 0, 0], [ 0, 0, 26]])

plt.figure(figsize=(8,6))

sns.heatmap(cm, cmap='Blues', annot=True, xticklabels=

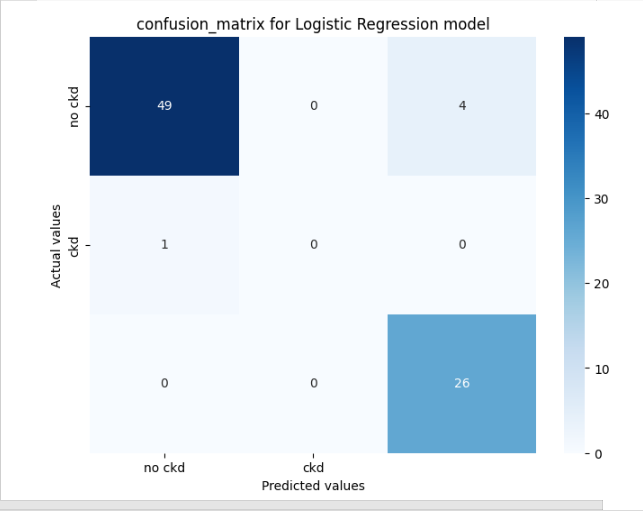
['no ckd', 'ckd'], yticklabels=['no ckd', 'ckd'])

plt.xlabel('Predicted values')

plt.ylabel('Actual values')

plt.title('confusion\_matrix for Logistic Regression model')

plt.show()



from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_predict)

cm

array([[49, 0, 4], [ 1, 0, 0], [ 0, 0, 26]])

plt.figure(figsize=(8,6))

sns.heatmap(cm, cmap='Blues', annot=True, xticklabels=

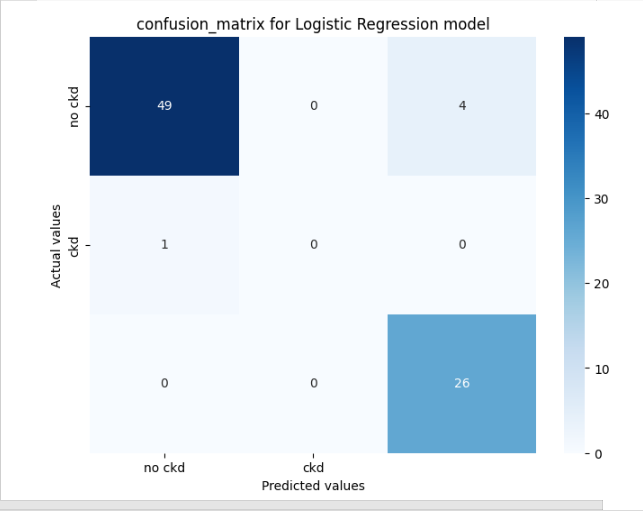
['no ckd','ckd'], yticklabels=['no ckd', 'ckd'])

plt.xlabel('Predicted values')

plt.ylabel('Actual values')

plt.title('Confusion Matrix for RandomForestClassifier')

plt.show()



from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_predict)

cm

array([[49, 0, 4], [ 1, 0, 0], [ 0, 0, 26]])

plt.figure(figsize=(8,6))

sns.heatmap(cm, cmap='Blues', annot=True, xticklabels=

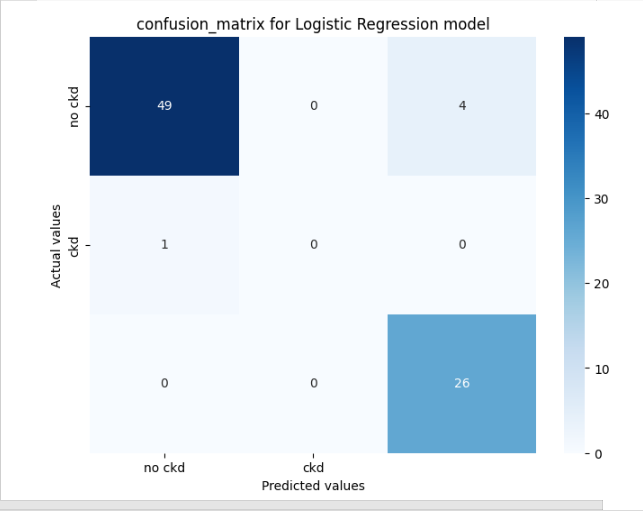
['no ckd','ckd'], yticklabels=['no ckd', 'ckd'])

plt.xlabel('Predicted values')

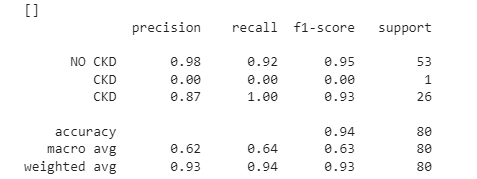
plt.ylabel('Actual values')

plt.title('Confusion Matrix for RandomForestClassifier')

plt.show()



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



from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_predict)

cm

array([[49, 0, 4], [ 1, 0, 0], [ 0, 0, 26]])

plt.figure(figsize=(8,6))

sns.heatmap(cm, cmap='Blues', annot=True, xticklabels=

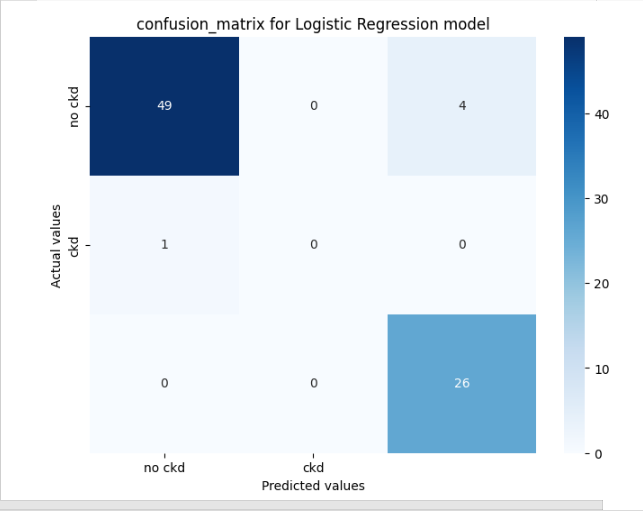
['no ckd','ckd'], yticklabels=['no ckd', 'ckd'])

plt.xlabel('Predicted values')

plt.ylabel('Actual values')

plt.title('Confusion Matrix for RandomForestClassifier')

plt.show()



bootstraps=[]

for model in list(set(final.model.values)):

    model\_df = final.loc[final.model == model]

    bootstrap = model\_df.sample(n=30, replace=True)

    bootstraps.append(bootstrap)

bootstrap\_df = pd.concat(bootstraps, ignore\_index=True)

results\_long = pd.melt(bootstrap\_df,id\_vars=['model'],var\_name='metrics',value\_name='values')

time\_metrics = ['fit\_time','score\_time']

results\_long\_nofit = results\_long.loc[~results\_long['metrics'].isin(time\_metrics)]

results\_long\_nofit = results\_long\_nofit.sort\_values(by='values')

results\_long\_fit = results\_long.loc[results\_long['metrics'].

isin(time\_metrics)]

results\_long\_fit = results\_long\_fit.sort\_values(by='values')

import matplotlib.pyplot as plt

import seaborn as sns

plt.figure(figsize=(20, 12))

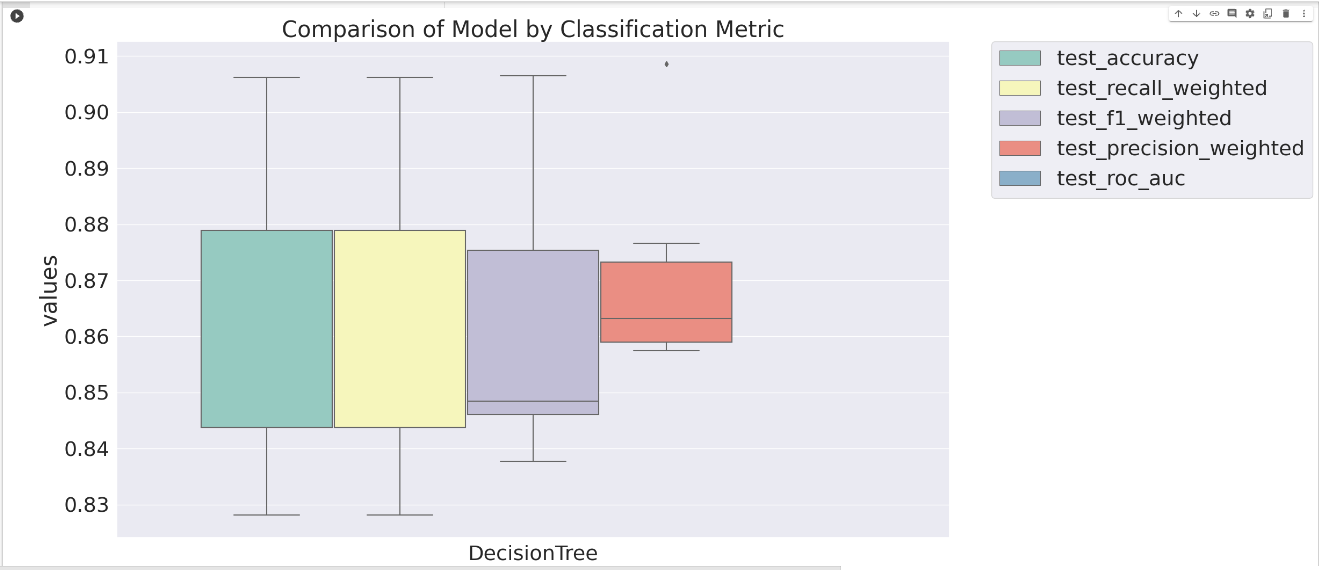
sns.set(font\_scale=2.5)

g=sns.boxplot(x="model", y="values", hue="metrics", data=results\_long\_nofit, palette="Set3")

plt.legend(bbox\_to\_anchor=(1.05,1),loc=2,borderaxespad=0.)

plt.title('Comparison of Model by Classification Metric')

plt.savefig('./benchmark\_models\_performance.png',dpi=300)



Milestone6

pickle.dump(lgr, open('ckd.pkl','wb'))

from flask import Flask, render\_template, request

import numpy as np

import pickle

app = Flask('\_name\_')

model = pickle.load(open('ckd.pkl', 'rb'))

@app.route('/')

def home():

    return render\_template('home.html')

@app.route('/prediction',methods=['POST', 'GET'])

def prediction():

    return render\_template('indexnew.html')

@app.route('/Home',methods=['POST','GET'])

def my\_home():

    return render\_template('home.html')

@app.route('/predict',methods=['POST'])

def predict():

    input\_features=[float(x) for x in request.form.values()]

    features\_values=[np.array(input\_features)]

    features\_name=['blood\_urea','blood glucose random', 'anemia',

                   'coronary\_artery\_disease', 'pus\_cell', 'red\_blood\_cells',

                   'diabetesmellitus', 'pedal\_edema']

    df=pd.DataFrame(features\_value, columns=fetures\_name)

    output=model.predict(df)