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. Support Vector Machine (SVR), Random Forest (LR), Artificial Neural Network (ANN), and Decision Tree are four 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.

Keywords: Chronic Kidney Disease (CKD), Machine Learning (ML), Support Vector Machine (SVR), Random Forest (LR), Artificial Neural Network (ANN), Decision Tree (DT).

1.INTRODUCTION

OVERVIEW:

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 todays 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 pretrained model to predict the Chronic Kidney Disease which can help in treatments of peoples who are suffer from this disease.

A kind of artificial intelligence is machine learning (ML) (AI). Its heart is algorithmic procedures, which allow the machine to solve issues without the need for specialist computer programming. The widespread use of ML in the medical industry promotes medical innovation, lowers medical expenses, and improves medical quality. However, further research on using ML to solve clinical problems in nephrology is needed. Understanding the aim and technique of ML application, as well as the present state of its use in nephrology, is required to properly address and overcome these issues. Machine learning has previously been used to identify human body state, evaluate disease-related aspects, and diagnose a variety of disorders. The term machine learning (ML) is very popular these days, and a lot of clinical prediction model studies have employed this type of technology. While the capacity to capture vast volumes of information on individual patients is transforming the healthcare business, the enormous volume of data being gathered is impractical for humans to comprehend. Machine learning allows healthcare practitioners to advance toward individualized care, often known as precision medicine, by automatically finding patterns and reasoning about data. The integration of machine learning, health informatics, and predictive analytics provides prospects to alter clinical decision support systems and assist improve patient outcomes.

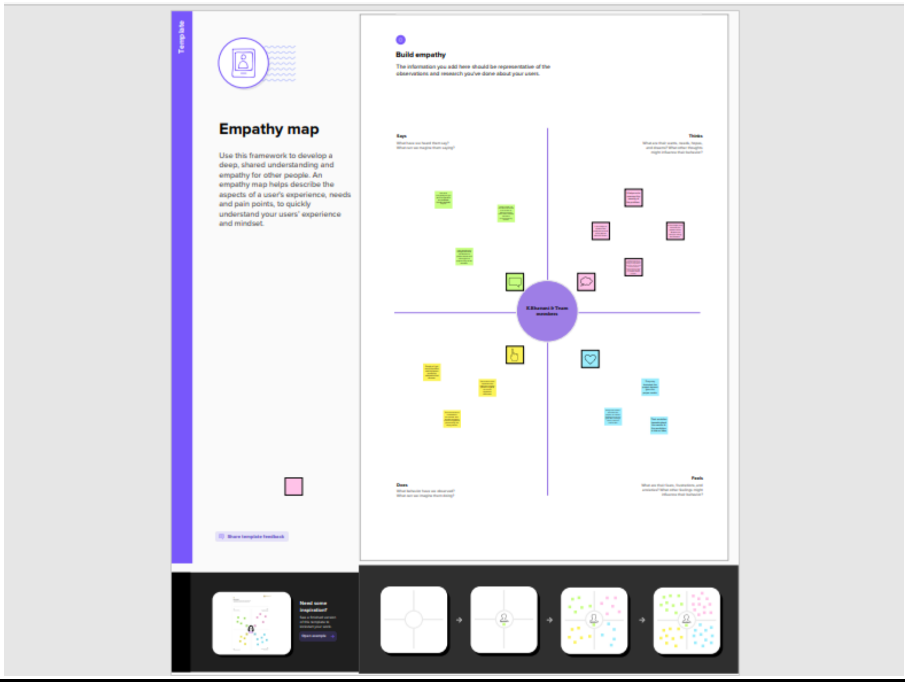
Chronic Kidney Disease refers to the kidneys' inability to fulfil their normal blood filtration role and other functions (CKD). The term "chronic" refers to the progressive deterioration of kidney cells over time. This is a severe renal failure in which the kidney no longer filters blood and there is a significant fluid accumulation in the body. This causes an abnormally high level of potassium and calcium salts in the body. High quantities of these salts in the body cause a variety of additional problems. The primary function of the kidneys is to filter excess water and wastes from the blood. This mechanism must work properly to balance the salts and minerals in our bodies. The proper salt balance is required to manage blood pressure, activate hormones, and create red blood cells, among other things. A high calcium concentration causes bone problems and cystic ovaries in women. CKD can also cause a sudden sickness or an allergy to specific medications. Acute is the medical term for this condition.

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.

1. PROBLEM DEFINITION & DESIGN THINKING

EMPATHY MAP:



Template

Empathy map

Use this framework to develop a deep, shared understanding and empathy for other people. An empathy map helps describe the aspects of a user's experience, needs and pain points, to quickly understand your users' experience and mindset.

Build empathy

The information you add here should be representative of the observations and research you've done about your users.

Says

Thinks

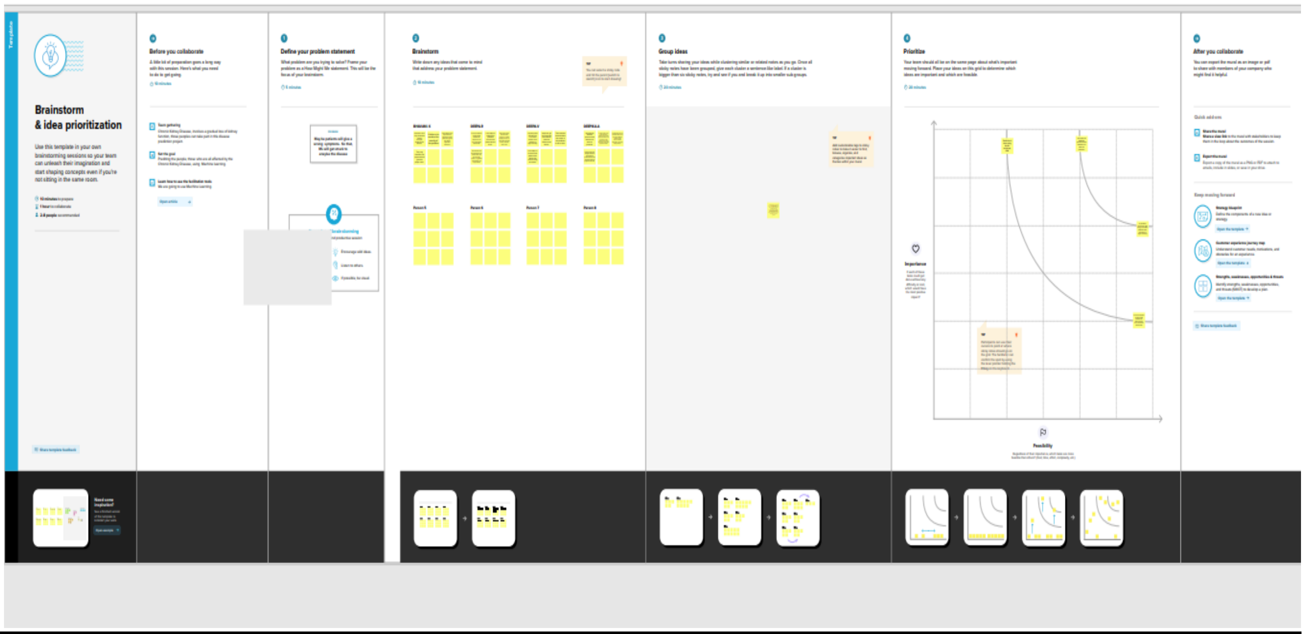
Does

Feels

Share template feedback

Need some inspiration?

IDEATION & BRAINSTROMING MAP:



Template

Brainstorm & idea prioritization

Before you collaborate

Define your problem statement

Brainstorm

Group ideas

Prioritize

Importance

Feasibility

After you collaborate

2. RESULT

The application uses KNN 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. When multiple patient data is analyzed, we got 97% of accuracy using KNN and 91% of accuracy using Naive Bayes. Graph Comparison of KNN.

3. ADVANTAGES

- ❖ Early diagnosis of CKD in people with diabetes will likely prevent CKD development, progression to ESKD and death, in addition to improving quality of life and substantially reducing healthcare cost.

- ❖ Early detection of CKD allows proper management that could **slow down CKD progression, prevent cardiovascular and other comorbidities and enable timely initiation of dialysis**. Screening for CKD could be best managed by partnership between primary care physicians and nephrologists.

- ❖ 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.

DISADVANTAGES:

- ❖ There are also some of the disadvantages such as data ownership problems, privacy and security related issues for human data administration etc.

5. APPLICATIONS

- 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.

- Random Forest and Artificial Neural Network have been used. They have extracted 20 out of 25 features and applied RF and ANN. RF has been identified with the highest accuracy of 97.12%.

5.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 KNN and Naïve Bayes alogorithms 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.

6.FUTURE SCOPE

This work will be considered as basement for the healthcare system for CKD patients. Also extension to this work is that implementation of deep leaning since deep learning provides high-quality performance than Machine Learning algorithm.

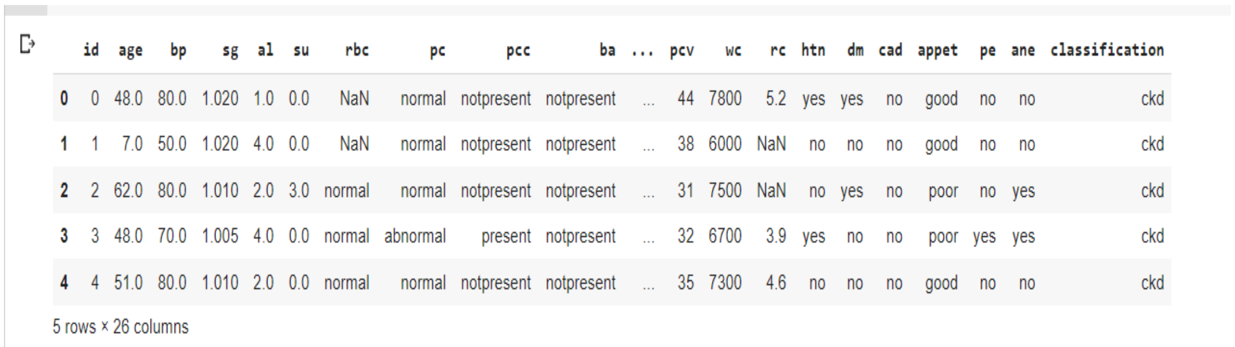
8.APPENDIX

Milestone 2: Data Collection & Preparation

Activity 1:collect the dataset

```
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
```

```
data=pd.read_csv("/content/kidney_disease.csv")
data.head()
```



	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	...	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	...	38	6000	NaN	no	no	no	good	no	no	ckd
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31	7500	NaN	no	yes	no	poor	no	yes	ckd
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	...	35	7300	4.6	no	no	no	good	no	no	ckd

5 rows × 26 columns

data.columns

```
Index(['id', 'age', 'bp', 'sg', 'al', 'su', 'rbc', 'pc', 'pcc', 'ba', 'bgr', 'bu',
'sc', 'sod', 'pot', 'hemo', 'pcv', 'wc', 'rc', 'htn', 'dm', 'cad', 'appet', 'pe',
'ane', 'classification'], dtype='object')
```

```
data.columns=['id','age','blood_pressure','specific_gravity','albumin','sugar','red_bl
ood_cells','pus_cell','pus_cell_clumps','bacteria','blood glucose random','blood_urea'
,'serum_creatinine','sodium','potassium','hemoglobin','packed_cell_volume','white_bloo
d_cell_count','red_blood_cell_count','hypertension','diabetesmellitus','coronary_arter
y_disease','appetite','pedal_edema','anemia','class']
data.columns
```

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

```
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()
```

```
id False
age True
blood_pressure True
specific_gravity True
albumin True
sugar True
red_blood_cells True
pus_cell True
pus_cell_clumps True
bacteria True
blood glucose random True
blood_urea True
serum_creatinine True
sodium True
```

```
potassium True
hemoglobin True
packed_cell_volume True
white_blood_cell_count True
red_blood_cell_count True
hypertension True
diabetesmellitus True
coronary_artery_disease True
appetite True
pedal_edema True
anemia True
class False
dtype: bool
```

```
data['blood glucose random'].fillna(data['blood glucose random'].mode()[0],inplace=Tru
e)
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=Tru
e)
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],inpla
ce=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.dtypes[data.dtypes=='O'].index.values)
print(catcols)
```

```
{'pus_cell', 'red_blood_cell_count', 'red_blood_cells', 'bacteria', 'hypertension',
'coronary_artery_disease', 'white_blood_cell_count', 'class', 'anemia',
'packed_cell_volume', 'diabetesmellitus', 'appetite', 'pedal_edema',
'pus_cell_clumps'}
```

```
for i in catcols:
  print("columns :",i)
  print(c(data[i]))
  print('*'*120+'\n')

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

columns : red_blood_cell_count
Counter({'5.2': 148, '4.5': 16, '4.9': 14, '4.7': 11, '3.9': 10, '4.8': 10, '4.6': 9,
'3.4': 9, '3.7': 8, '5.0': 8, '6.1': 8, '5.5': 8, '5.9': 8, '3.8': 7, '5.4': 7, '5.8':
7, '5.3': 7, '4.3': 6, '4.2': 6, '5.6': 6, '4.4': 5, '3.2': 5, '4.1': 5, '6.2': 5,
'5.1': 5, '6.4': 5, '5.7': 5, '6.5': 5, '3.6': 4, '6.0': 4, '6.3': 4, '4.0': 3, '4':
3, '3.5': 3, '3.3': 3, '5': 2, '2.6': 2, '2.8': 2, '2.5': 2, '3.1': 2, '2.1': 2,
'2.9': 2, '2.7': 2, '3.0': 2, '2.3': 1, '8.0': 1, '3': 1, '2.4': 1, '\t?': 1})
***************************************************************************************
*********************************

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

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

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

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

columns : white_blood_cell_count
Counter({'9800': 116, '6700': 10, '9600': 9, '9200': 9, '7200': 9, '6900': 8, '11000':
8, '5800': 8, '7800': 7, '9100': 7, '9400': 7, '7000': 7, '4300': 6, '6300': 6,
'10700': 6, '10500': 6, '7500': 5, '8300': 5, '7900': 5, '8600': 5, '5600': 5,
'10200': 5, '5000': 5, '8100': 5, '9500': 5, '6000': 4, '6200': 4, '10300': 4, '7700':
4, '5500': 4, '10400': 4, '6800': 4, '6500': 4, '4700': 4, '7300': 3, '4500': 3,
'8400': 3, '6400': 3, '4200': 3, '7400': 3, '8000': 3, '5400': 3, '3800': 2, '11400':
2, '5300': 2, '8500': 2, '14600': 2, '7100': 2, '13200': 2, '9000': 2, '8200': 2,
'15200': 2, '12400': 2, '12800': 2, '8800': 2, '5700': 2, '9300': 2, '6600': 2,
'12100': 1, '12200': 1, '18900': 1, '21600': 1, '11300': 1, '\t6200': 1, '11800': 1,
'12500': 1, '11900': 1, '12700': 1, '13600': 1, '14900': 1, '16300': 1, '\t8400': 1,
'10900': 1, '2200': 1, '11200': 1, '19100': 1, '\t?': 1, '12300': 1, '16700': 1,
'2600': 1, '26400': 1, '4900': 1, '12000': 1, '15700': 1, '4100': 1, '11500': 1,
'10800': 1, '9900': 1, '5200': 1, '5900': 1, '9700': 1, '5100': 1})
***************************************************************************************
*********************************

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

columns : anemia
Counter({'no': 340, 'yes': 60})
***************************************************************************************
*********************************
```

```
columns : packed_cell_volume
Counter({'41': 91, '52': 21, '44': 19, '48': 19, '40': 16, '43': 14, '45': 13, '42':
13, '32': 12, '36': 12, '33': 12, '28': 12, '50': 12, '37': 11, '34': 11, '35': 9,
'29': 9, '30': 9, '46': 9, '31': 8, '39': 7, '24': 7, '26': 6, '38': 5, '47': 4, '49':
4, '53': 4, '51': 4, '54': 4, '27': 3, '22': 3, '25': 3, '23': 2, '19': 2, '16': 1,
'\t?': 1, '14': 1, '18': 1, '17': 1, '15': 1, '21': 1, '20': 1, '\t43': 1, '9': 1})
****************************************************************************************
**********************************

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

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

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

columns : pus_cell_clumps
Counter({'notpresent': 358, 'present': 42})
****************************************************************************************
**********************************
```

```
catcols.remove('red_blood_cell_count')
catcols.remove('packed_cell_volume')
catcols.remove('white_blood_cell_count')
print(catcols)
```

```
{'pus_cell', 'red_blood_cells', 'bacteria', 'hypertension', 'coronary_artery_disease',
'class', 'anemia', 'diabetesmellitus', 'appetite', 'pedal_edema', 'pus_cell_clumps'}
```

```
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: pus_cell
Counter({'normal': 324, 'abnormal': 76})
Counter({1: 324, 0: 76})
****************************************************************************************
**************
LABEL ENCODING OF: red_blood_cells
Counter({'normal': 353, 'abnormal': 47})
Counter({1: 353, 0: 47})
```

```
************************************************************************************
**************
LABEL ENCODING OF: bacteria
Counter({'notpresent': 378, 'present': 22})
Counter({0: 378, 1: 22})
************************************************************************************
**************
LABEL ENCODING OF: hypertension
Counter({'no': 253, 'yes': 147})
Counter({0: 253, 1: 147})
************************************************************************************
**************
LABEL ENCODING OF: coronary_artery_disease
Counter({'no': 364, 'yes': 34, '\tno': 2})
Counter({1: 364, 2: 34, 0: 2})
************************************************************************************
**************
LABEL ENCODING OF: class
Counter({'ckd': 248, 'notckd': 150, 'ckd\t': 2})
Counter({0: 248, 2: 150, 1: 2})
************************************************************************************
**************
LABEL ENCODING OF: anemia
Counter({'no': 340, 'yes': 60})
Counter({0: 340, 1: 60})
************************************************************************************
**************
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: appetite
Counter({'good': 318, 'poor': 82})
Counter({0: 318, 1: 82})
************************************************************************************
**************
LABEL ENCODING OF: pedal_edema
Counter({'no': 324, 'yes': 76})
Counter({0: 324, 1: 76})
************************************************************************************
**************
LABEL ENCODING OF: pus_cell_clumps
Counter({'notpresent': 358, 'present': 42})
Counter({0: 358, 1: 42})
```

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

```
{'id', 'albumin', 'blood_pressure', 'bacteria', 'hemoglobin', 'blood glucose random',
'class', 'red_blood_cells', 'pus_cell', 'hypertension', 'blood_urea', 'potassium',
'pedal_edema', 'anemia', 'serum_creatinine', 'age', 'diabetesmellitus',
'pus_cell_clumps', 'sugar', 'coronary_artery_disease', 'specific_gravity', 'sodium',
'appetite'}
```

```
for i in contcols:
  print("Continous Columns :",i)
  print(c(data[i]))
  print('*'*120+'\n')
```

```
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 : 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 : 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 : bacteria
Counter({0: 378, 1: 22})
*************************************************************************************
*********************************

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

Continous Columns : red_blood_cells
Counter({1: 353, 0: 47})
************************************************************************************
*********************************

Continous Columns : pus_cell
Counter({1: 324, 0: 76})
************************************************************************************
*********************************

Continous Columns : hypertension
Counter({0: 253, 1: 147})
************************************************************************************
*********************************

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

Continous Columns : anemia
Counter({0: 340, 1: 60})
*************************************************************************************
*********************************

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 : 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})
*************************************************************************************
*********************************

Continous Columns : diabetesmellitus
Counter({3: 260, 4: 134, 0: 3, 1: 2, 2: 1})
*************************************************************************************
*********************************

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

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 : 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 : appetite
Counter({0: 318, 1: 82})
*************************************************************************************
*********************************
```

```
contcols.remove('specific_gravity')
contcols.remove('albumin')
contcols.remove('sugar')
print(contcols)
```

```
{'id', 'blood_pressure', 'bacteria', 'hemoglobin', 'blood glucose random', 'class', 
'red_blood_cells', 'pus_cell', 'hypertension', 'blood_urea', 'potassium', 
'pedal_edema', 'anemia', 'serum_creatinine', 'age', 'diabetesmellitus', 
'pus_cell_clumps', 'coronary_artery_disease', 'sodium', 'appetite'}
```

```
contcols.add('red_blood_cell_count')
contcols.add('packed_cell_volume')
contcols.add('white_blood_cell_count')
print(contcols)
```

```
{'id', 'blood_pressure', 'bacteria', 'hemoglobin', 'blood glucose random', 'class', 
'red_blood_cells', 'pus_cell', 'red_blood_cell_count', 'hypertension', 'blood_urea', 
'packed_cell_volume', 'potassium', 'pedal_edema', 'anemia', 'serum_creatinine', 'age', 
'diabetesmellitus', 'pus_cell_clumps', 'coronary_artery_disease', 
'white_blood_cell_count', 'sodium', 'appetite'}
```

```
catcols.add('specific_gravity')
catcols.add('albumin')
catcols.add('sugar')
print(catcols)
```

```
{'pus_cell', 'albumin', 'red_blood_cells', 'bacteria', 'sugar', 'hypertension',
'coronary_artery_disease', 'class', 'specific_gravity', 'anemia', 'diabetesmellitus',
'appetite', 'pedal_edema', 'pus_cell_clumps'}
```

```
data['coronary_artery_disease'] = data.coronary_artery_disease.replace('\tno','no')
c(data['coronary_artery_disease'])
```

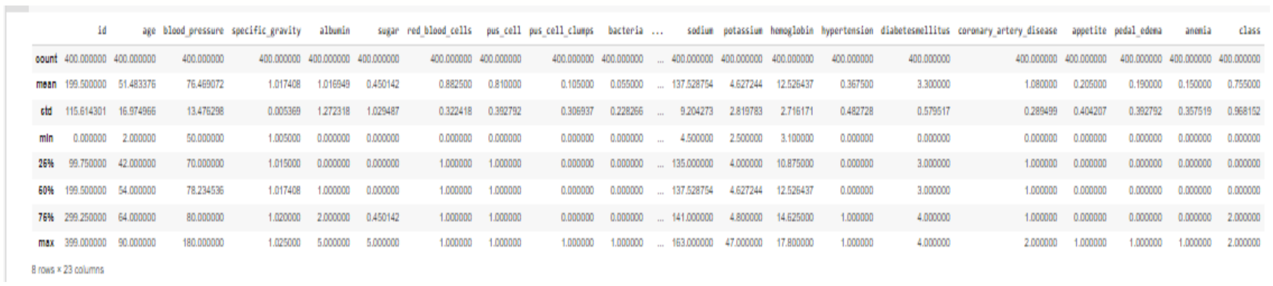
```
Counter({1: 364, 2: 34, 0: 2})
```

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

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

Task3

```
data.describe()
```



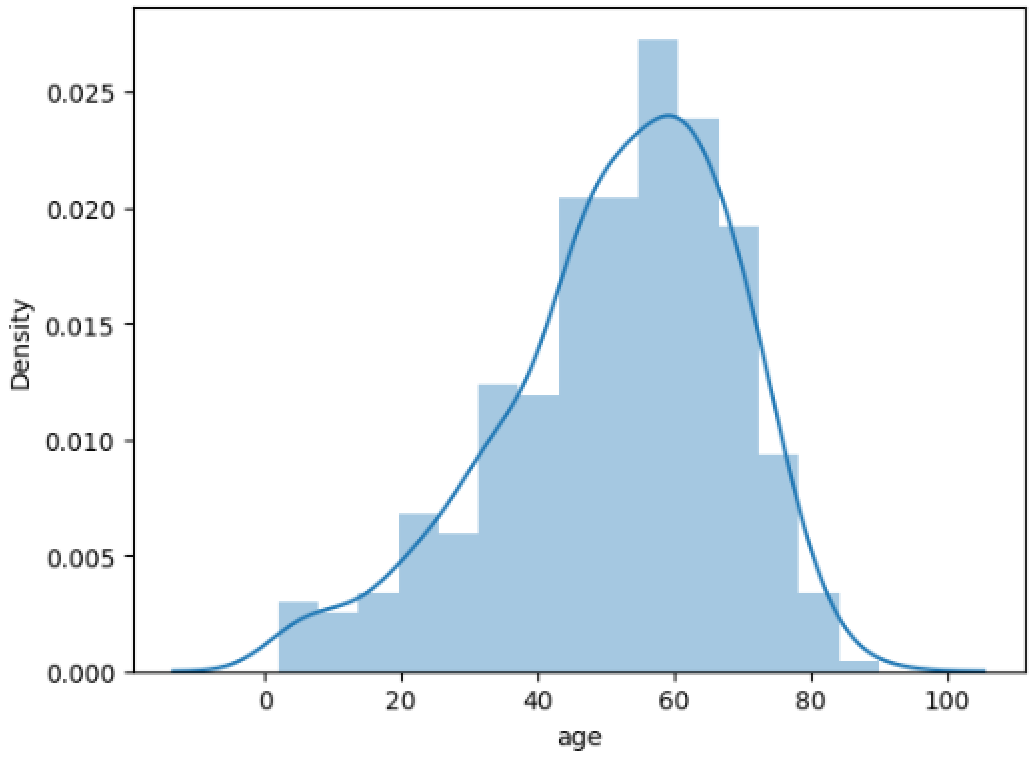
	id	age	blood_pressure	specific_gravity	albumin	sugar	red_blood_cells	pus_cell	pus_cell_clumps	bacteria	...	sodium	potassium	hemoglobin	hypertension	diabetesmellitus	coronary_artery_disease	appetite	pedal_edema	anemia	class
count	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	...	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000
mean	199.500000	51.483376	76.469072	1.017408	1.016949	0.450142	0.882500	0.810000	0.105000	0.055000	...	137.528754	4.627244	12.526437	0.367500	3.300000	1.080000	0.205000	0.190000	0.150000	0.755000
std	115.614301	16.974966	13.476298	0.005369	1.272318	1.029487	0.322418	0.392792	0.306937	0.228266	...	9.204273	2.819783	2.716171	0.482728	0.579517	0.289499	0.404207	0.392792	0.357519	0.968152
min	0.000000	2.000000	50.000000	1.005000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	4.500000	2.500000	3.100000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	99.750000	42.000000	70.000000	1.015000	0.000000	0.000000	1.000000	1.000000	0.000000	0.000000	...	135.000000	4.000000	10.875000	0.000000	3.000000	1.000000	0.000000	0.000000	0.000000	0.000000
50%	199.500000	54.000000	78.234536	1.017408	1.000000	0.000000	1.000000	1.000000	0.000000	0.000000	...	137.528754	4.627244	12.526437	0.000000	3.000000	1.000000	0.000000	0.000000	0.000000	0.000000
75%	299.250000	64.000000	80.000000	1.020000	2.000000	0.450142	1.000000	1.000000	0.000000	0.000000	...	141.000000	4.800000	14.625000	1.000000	4.000000	1.000000	0.000000	0.000000	0.000000	2.000000
max	399.000000	90.000000	180.000000	1.025000	5.000000	5.000000	1.000000	1.000000	1.000000	1.000000	...	163.000000	47.000000	17.800000	1.000000	4.000000	2.000000	1.000000	1.000000	1.000000	2.000000

8 rows × 23 columns

```
sns.distplot(data.age)
```

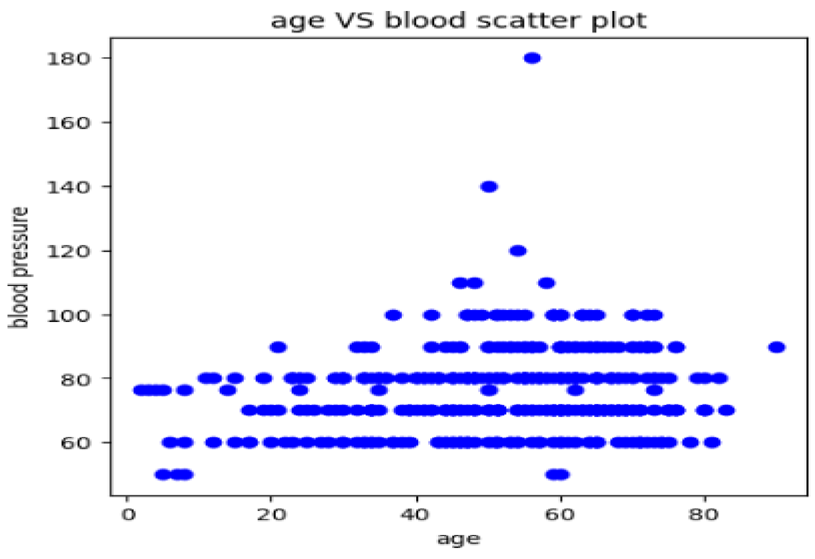
```
<ipython-input-21-868c85374ad7>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data.age)
<Axes: xlabel='age', ylabel='Density'>
```

[illegible]



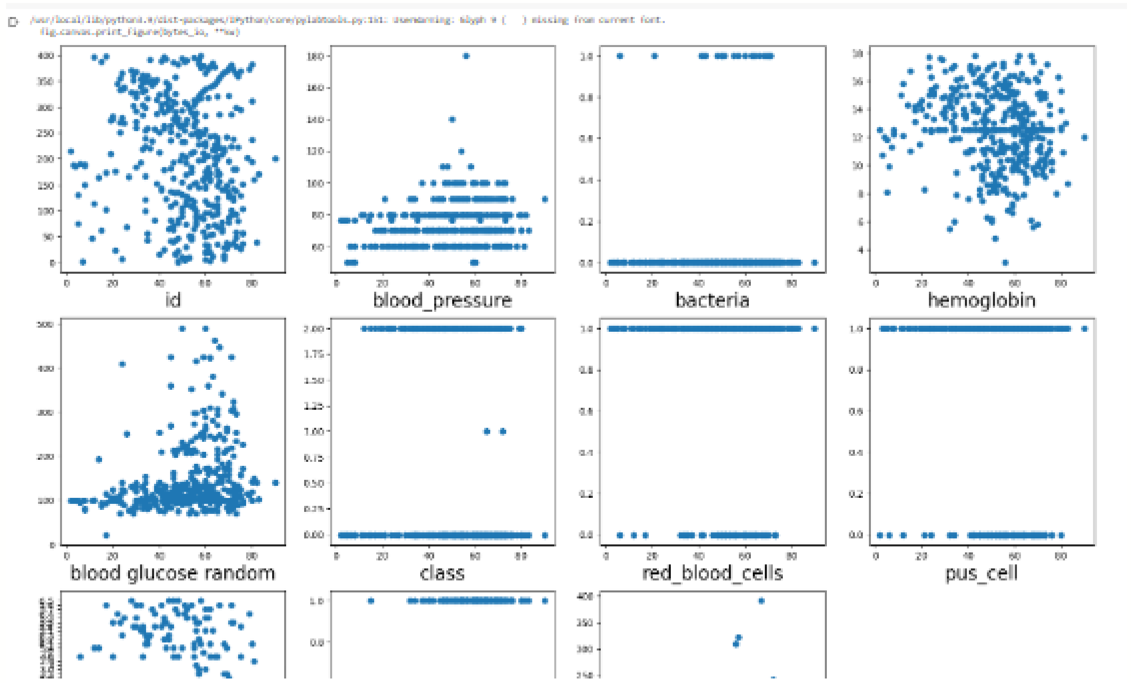
```
import matplotlib.pyplot as plt
fig=plt.figure(figsize=(5,5))
plt.scatter(data['age'],data['blood_pressure'],color='blue')
plt.xlabel('age')
plt.ylabel('blood pressure')
plt.title("age VS blood scatter plot")
```

```
Text(0.5, 1.0, 'age VS blood scatter plot')
```



```
plt.figure(figsize=(20,15), facecolor='white')
plotnumber = 1
```

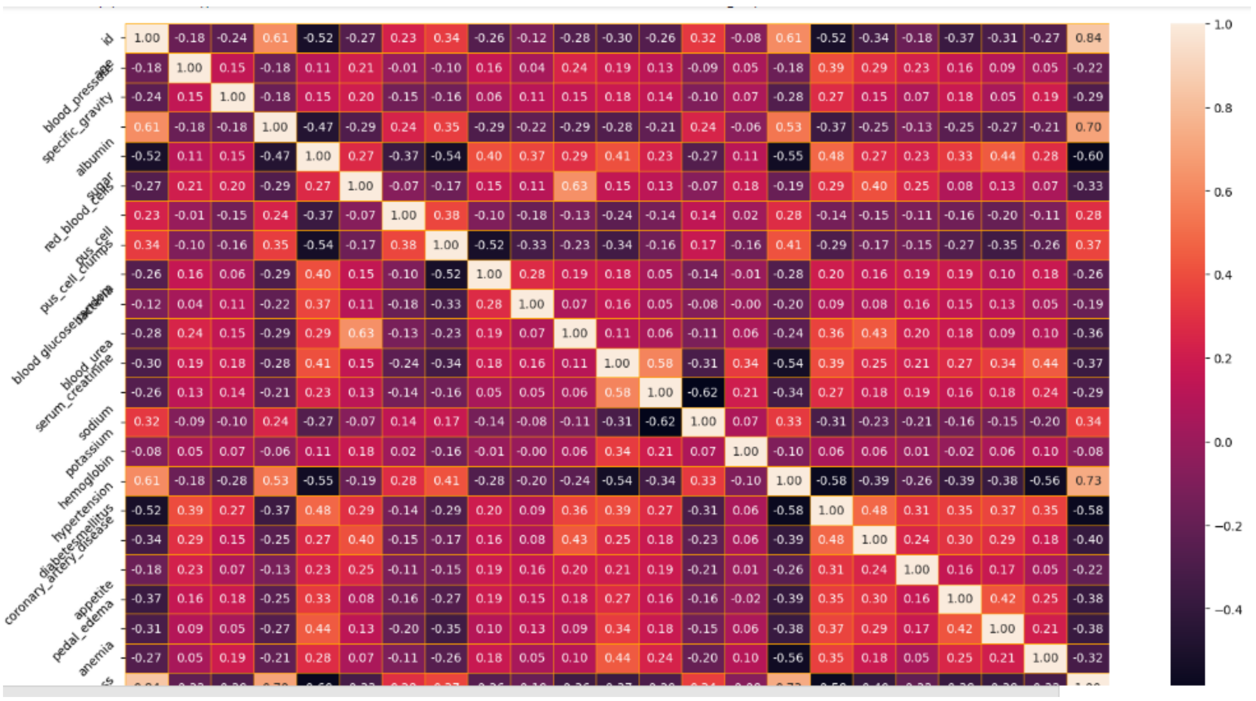
```
for column in contcols:
  if plotnumber<=11:
    ax = plt.subplot(3,4,plotnumber)
    plt.scatter(data['age'],data[column])
    plt.xlabel(column,fontsize=20)
  plotnumber+=1
plt.show()
```



```
/usr/local/lib/python3.9/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 9 (	) missing from current font.
  fig.canvas.print_figure(bytes_io, **kw)
```

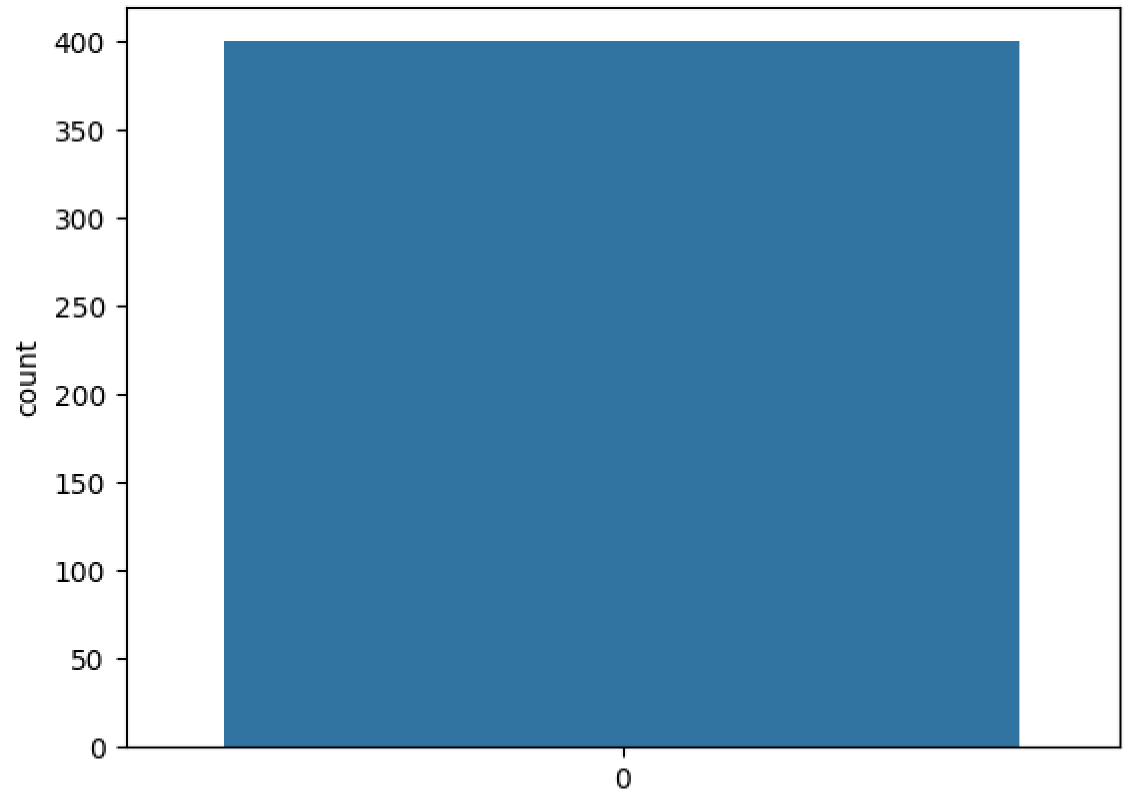
```
f,ax=plt.subplots(figsize=(18,10))
sns.heatmap(data.corr(),annot=True,fmt=".2f",ax=ax,linewidths=0.5,linecolor="orange")
plt.xticks(rotation=45)
plt.yticks(rotation=45)
plt.show()
```

```
<ipython-input-34-e8d9004f5c7e>:2: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it will
default to False. Select only valid columns or specify the value of
numeric_only to silence this warning.

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



```
sns.countplot(data['class'])
```

```
<Axes: ylabel='count'>
```



```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_bal=sc.fit_transform(x)


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)
```

Task4

```
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 [==============================] - 4s 14ms/step - loss: 2.2048 - accuracy: 0.2227 - val_loss:
1.4365 - val_accuracy: 0.6562
Epoch 2/100
26/26 [==============================] - 0s 3ms/step - loss: 0.9171 - accuracy: 0.3594 - val_loss: 0.5479
- val_accuracy: 0.1875
Epoch 3/100
26/26 [==============================] - 0s 3ms/step - loss: 0.5997 - accuracy: 0.1875 - val_loss: 0.6109
- val_accuracy: 0.1406
Epoch 4/100
26/26 [==============================] - 0s 4ms/step - loss: 0.4835 - accuracy: 0.2148 - val_loss: 0.5270
- val_accuracy: 0.1875
Epoch 5/100
26/26 [==============================] - 0s 4ms/step - loss: 0.5568 - accuracy: 0.2031 - val_loss: 0.6949
- val_accuracy: 0.6094
Epoch 6/100
26/26 [==============================] - 0s 4ms/step - loss: 0.8272 - accuracy: 0.2852 - val_loss: 0.9742
- val_accuracy: 0.6562
```

```
Epoch 7/100
26/26 [==============================] - 0s 4ms/step - loss: 0.4711 - accuracy: 0.2227 - val_loss: 0.4680
- val_accuracy: 0.1875
Epoch 8/100
26/26 [==============================] - 0s 4ms/step - loss: 0.4261 - accuracy: 0.2188 - val_loss: 0.4100
- val_accuracy: 0.2188
Epoch 9/100
26/26 [==============================] - 0s 3ms/step - loss: 0.3835 - accuracy: 0.2617 - val_loss: 0.4292
- val_accuracy: 0.2031
Epoch 10/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3873 - accuracy: 0.2500 - val_loss: 0.4394
- val_accuracy: 0.2031
Epoch 11/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2823 - accuracy: 0.2695 - val_loss: 0.5949
- val_accuracy: 0.1875
Epoch 12/100
26/26 [==============================] - 0s 4ms/step - loss: 0.4096 - accuracy: 0.2070 - val_loss: 0.6250
- val_accuracy: 0.4844
Epoch 13/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3747 - accuracy: 0.2852 - val_loss: 0.3442
- val_accuracy: 0.3281
Epoch 14/100
26/26 [==============================] - 0s 3ms/step - loss: 0.3358 - accuracy: 0.2695 - val_loss: 0.3940
- val_accuracy: 0.2656
Epoch 15/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2018 - accuracy: 0.2266 - val_loss: 0.3563
- val_accuracy: 0.4062
Epoch 16/100
26/26 [==============================] - 0s 4ms/step - loss: 0.1615 - accuracy: 0.2617 - val_loss: 0.3745
- val_accuracy: 0.3281
Epoch 17/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3019 - accuracy: 0.3203 - val_loss: 0.2356
- val_accuracy: 0.2812
Epoch 18/100
26/26 [==============================] - 0s 4ms/step - loss: -0.0043 - accuracy: 0.2812 - val_loss: 0.3077
- val_accuracy: 0.2969
Epoch 19/100
26/26 [==============================] - 0s 3ms/step - loss: -0.1773 - accuracy: 0.3164 - val_loss: 0.3292
- val_accuracy: 0.2344
Epoch 20/100
26/26 [==============================] - 0s 4ms/step - loss: 6.1474e-04 - accuracy: 0.2930 - val_loss:
0.3287 - val_accuracy: 0.2656
Epoch 21/100
26/26 [==============================] - 0s 3ms/step - loss: -0.2269 - accuracy: 0.2969 - val_loss: 0.4992
- val_accuracy: 0.2344
Epoch 22/100
26/26 [==============================] - 0s 4ms/step - loss: -0.6257 - accuracy: 0.2812 - val_loss: 0.1017
- val_accuracy: 0.3750
Epoch 23/100
26/26 [==============================] - 0s 4ms/step - loss: -0.6817 - accuracy: 0.3086 - val_loss: -
0.0235 - val_accuracy: 0.3594
Epoch 24/100
26/26 [==============================] - 0s 4ms/step - loss: -1.1833 - accuracy: 0.3047 - val_loss: 0.8737
- val_accuracy: 0.2188
Epoch 25/100
```

```
26/26 [==============================] - 0s 4ms/step - loss: -1.2722 - accuracy: 0.3047 - val_loss: 0.5743
- val_accuracy: 0.2812
Epoch 26/100
26/26 [==============================] - 0s 4ms/step - loss: -2.6032 - accuracy: 0.3086 - val_loss: -
0.6850 - val_accuracy: 0.3125
Epoch 27/100
26/26 [==============================] - 0s 4ms/step - loss: -3.4789 - accuracy: 0.2930 - val_loss: -
0.2767 - val_accuracy: 0.3125
Epoch 28/100
26/26 [==============================] - 0s 3ms/step - loss: -5.9341 - accuracy: 0.3242 - val_loss: -
2.1193 - val_accuracy: 0.3438
Epoch 29/100
26/26 [==============================] - 0s 4ms/step - loss: -10.2077 - accuracy: 0.3086 - val_loss: -
1.7894 - val_accuracy: 0.3281
Epoch 30/100
26/26 [==============================] - 0s 4ms/step - loss: -14.4810 - accuracy: 0.3203 - val_loss: -
3.2881 - val_accuracy: 0.2656
Epoch 31/100
26/26 [==============================] - 0s 3ms/step - loss: -25.3035 - accuracy: 0.2734 - val_loss: -
6.7795 - val_accuracy: 0.4375
Epoch 32/100
26/26 [==============================] - 0s 4ms/step - loss: -24.8979 - accuracy: 0.3750 - val_loss: -
5.8626 - val_accuracy: 0.2969
Epoch 33/100
26/26 [==============================] - 0s 4ms/step - loss: -56.2986 - accuracy: 0.2852 - val_loss: -
23.9756 - val_accuracy: 0.3281
Epoch 34/100
26/26 [==============================] - 0s 3ms/step - loss: -54.3976 - accuracy: 0.3320 - val_loss: -
2.8867 - val_accuracy: 0.4688
Epoch 35/100
26/26 [==============================] - 0s 3ms/step - loss: -120.3166 - accuracy: 0.3164 - val_loss: -
15.4850 - val_accuracy: 0.2344
Epoch 36/100
26/26 [==============================] - 0s 4ms/step - loss: -174.4860 - accuracy: 0.2891 - val_loss: -
71.7914 - val_accuracy: 0.4375
Epoch 37/100
26/26 [==============================] - 0s 3ms/step - loss: -237.4409 - accuracy: 0.3320 - val_loss: -
47.0410 - val_accuracy: 0.4531
Epoch 38/100
26/26 [==============================] - 0s 3ms/step - loss: -388.6144 - accuracy: 0.3086 - val_loss: -
107.9362 - val_accuracy: 0.2500
Epoch 39/100
26/26 [==============================] - 0s 4ms/step - loss: -554.7281 - accuracy: 0.3242 - val_loss: -
306.4272 - val_accuracy: 0.2969
Epoch 40/100
26/26 [==============================] - 0s 3ms/step - loss: -869.0827 - accuracy: 0.2891 - val_loss: -
389.8230 - val_accuracy: 0.3281
Epoch 41/100
26/26 [==============================] - 0s 4ms/step - loss: -1253.6703 - accuracy: 0.3086 - val_loss: -
551.7296 - val_accuracy: 0.3125
Epoch 42/100
26/26 [==============================] - 0s 3ms/step - loss: -1707.9587 - accuracy: 0.3359 - val_loss: -
814.2131 - val_accuracy: 0.3438
Epoch 43/100
26/26 [==============================] - 0s 4ms/step - loss: -2320.7683 - accuracy: 0.3086 - val_loss: -
701.7258 - val_accuracy: 0.2969
```

```
Epoch 44/100
26/26 [==============================] - 0s 4ms/step - loss: -3065.9426 - accuracy: 0.3086 - val_loss: -
731.8843 - val_accuracy: 0.2500
Epoch 45/100
26/26 [==============================] - 0s 4ms/step - loss: -3660.0698 - accuracy: 0.3008 - val_loss: -
1574.1968 - val_accuracy: 0.2812
Epoch 46/100
26/26 [==============================] - 0s 4ms/step - loss: -5680.1357 - accuracy: 0.3203 - val_loss: -
2293.3916 - val_accuracy: 0.3281
Epoch 47/100
26/26 [==============================] - 0s 4ms/step - loss: -6967.0845 - accuracy: 0.2969 - val_loss: -
3286.7412 - val_accuracy: 0.3750
Epoch 48/100
26/26 [==============================] - 0s 4ms/step - loss: -9047.5156 - accuracy: 0.3125 - val_loss: -
2247.7939 - val_accuracy: 0.2344
Epoch 49/100
26/26 [==============================] - 0s 4ms/step - loss: -12428.6309 - accuracy: 0.3281 - val_loss: -
5268.7109 - val_accuracy: 0.3281
Epoch 50/100
26/26 [==============================] - 0s 4ms/step - loss: -14076.1992 - accuracy: 0.3086 - val_loss: -
3762.0325 - val_accuracy: 0.2500
Epoch 51/100
26/26 [==============================] - 0s 3ms/step - loss: -18751.8379 - accuracy: 0.3008 - val_loss: -
8220.3857 - val_accuracy: 0.3281
Epoch 52/100
26/26 [==============================] - 0s 4ms/step - loss: -23258.8203 - accuracy: 0.3125 - val_loss: -
9964.2305 - val_accuracy: 0.3281
Epoch 53/100
26/26 [==============================] - 0s 4ms/step - loss: -27949.6230 - accuracy: 0.3164 - val_loss: -
12395.6992 - val_accuracy: 0.3438
Epoch 54/100
26/26 [==============================] - 0s 3ms/step - loss: -33556.9609 - accuracy: 0.2969 - val_loss: -
15524.3359 - val_accuracy: 0.3281
Epoch 55/100
26/26 [==============================] - 0s 3ms/step - loss: -41140.9453 - accuracy: 0.2852 - val_loss: -
19366.2051 - val_accuracy: 0.3750
Epoch 56/100
26/26 [==============================] - 0s 3ms/step - loss: -46924.2305 - accuracy: 0.3477 - val_loss: -
16395.5742 - val_accuracy: 0.2812
Epoch 57/100
26/26 [==============================] - 0s 3ms/step - loss: -56678.1484 - accuracy: 0.3477 - val_loss: -
19839.3164 - val_accuracy: 0.2656
Epoch 58/100
26/26 [==============================] - 0s 4ms/step - loss: -66784.3594 - accuracy: 0.3047 - val_loss: -
27607.7031 - val_accuracy: 0.3125
Epoch 59/100
26/26 [==============================] - 0s 4ms/step - loss: -80370.1562 - accuracy: 0.3047 - val_loss: -
30604.3105 - val_accuracy: 0.3125
Epoch 60/100
26/26 [==============================] - 0s 4ms/step - loss: -94207.0469 - accuracy: 0.3203 - val_loss: -
38359.4961 - val_accuracy: 0.3438
Epoch 61/100
26/26 [==============================] - 0s 3ms/step - loss: -107886.5938 - accuracy: 0.2969 - val_loss: -
41153.1094 - val_accuracy: 0.4219
Epoch 62/100
```

```
26/26 [==============================] - 0s 3ms/step - loss: -127458.7891 - accuracy: 0.3633 - val_loss: -
44060.1641 - val_accuracy: 0.2656
Epoch 63/100
26/26 [==============================] - 0s 4ms/step - loss: -140593.7812 - accuracy: 0.3086 - val_loss: -
60062.4531 - val_accuracy: 0.3438
Epoch 64/100
26/26 [==============================] - 0s 3ms/step - loss: -163717.9531 - accuracy: 0.3125 - val_loss: -
65308.0547 - val_accuracy: 0.2969
Epoch 65/100
26/26 [==============================] - 0s 4ms/step - loss: -182544.9844 - accuracy: 0.3086 - val_loss: -
81722.9844 - val_accuracy: 0.2969
Epoch 66/100
26/26 [==============================] - 0s 4ms/step - loss: -209554.3281 - accuracy: 0.3281 - val_loss: -
85782.1094 - val_accuracy: 0.3438
Epoch 67/100
26/26 [==============================] - 0s 3ms/step - loss: -232767.3281 - accuracy: 0.3008 - val_loss: -
99050.7734 - val_accuracy: 0.3438
Epoch 68/100
26/26 [==============================] - 0s 3ms/step - loss: -269942.0938 - accuracy: 0.2852 - val_loss: -
116290.5000 - val_accuracy: 0.4062
Epoch 69/100
26/26 [==============================] - 0s 3ms/step - loss: -273457.5938 - accuracy: 0.3555 - val_loss: -
139497.4219 - val_accuracy: 0.3438
Epoch 70/100
26/26 [==============================] - 0s 4ms/step - loss: -326605.8438 - accuracy: 0.3086 - val_loss: -
138202.0000 - val_accuracy: 0.3438
Epoch 71/100
26/26 [==============================] - 0s 4ms/step - loss: -361164.9688 - accuracy: 0.3203 - val_loss: -
163524.6719 - val_accuracy: 0.2969
Epoch 72/100
26/26 [==============================] - 0s 4ms/step - loss: -390278.5000 - accuracy: 0.2891 - val_loss: -
183546.9375 - val_accuracy: 0.3438
Epoch 73/100
26/26 [==============================] - 0s 4ms/step - loss: -450391.6875 - accuracy: 0.3789 - val_loss: -
130775.7812 - val_accuracy: 0.2656
Epoch 74/100
26/26 [==============================] - 0s 4ms/step - loss: -437421.1875 - accuracy: 0.2734 - val_loss: -
202538.4062 - val_accuracy: 0.3438
Epoch 75/100
26/26 [==============================] - 0s 5ms/step - loss: -521634.6562 - accuracy: 0.3047 - val_loss: -
243800.6875 - val_accuracy: 0.3438
Epoch 76/100
26/26 [==============================] - 0s 7ms/step - loss: -574653.6250 - accuracy: 0.3281 - val_loss: -
252712.0156 - val_accuracy: 0.2969
Epoch 77/100
26/26 [==============================] - 0s 6ms/step - loss: -634765.3125 - accuracy: 0.3164 - val_loss: -
289275.9375 - val_accuracy: 0.3125
Epoch 78/100
26/26 [==============================] - 0s 6ms/step - loss: -681481.3125 - accuracy: 0.2930 - val_loss: -
303045.1875 - val_accuracy: 0.3438
Epoch 79/100
26/26 [==============================] - 0s 5ms/step - loss: -741731.0000 - accuracy: 0.3633 - val_loss: -
331268.5938 - val_accuracy: 0.2969
Epoch 80/100
26/26 [==============================] - 0s 5ms/step - loss: -779055.8750 - accuracy: 0.2852 - val_loss: -
353191.1562 - val_accuracy: 0.3438
```

```
Epoch 81/100
26/26 [==============================] - 0s 5ms/step - loss: -900078.6250 - accuracy: 0.3203 - val_loss: -
351035.0625 - val_accuracy: 0.2812
Epoch 82/100
26/26 [==============================] - 0s 5ms/step - loss: -958793.8125 - accuracy: 0.3008 - val_loss: -
429064.1875 - val_accuracy: 0.3438
Epoch 83/100
26/26 [==============================] - 0s 5ms/step - loss: -969407.8125 - accuracy: 0.3125 - val_loss: -
464402.0000 - val_accuracy: 0.3438
Epoch 84/100
26/26 [==============================] - 0s 6ms/step - loss: -1115386.2500 - accuracy: 0.3398 - val_loss:
-435046.5938 - val_accuracy: 0.2969
Epoch 85/100
26/26 [==============================] - 0s 8ms/step - loss: -1187751.2500 - accuracy: 0.3008 - val_loss:
-521964.3750 - val_accuracy: 0.4219
Epoch 86/100
26/26 [==============================] - 0s 5ms/step - loss: -1278379.0000 - accuracy: 0.3320 - val_loss:
-564589.6250 - val_accuracy: 0.3125
Epoch 87/100
26/26 [==============================] - 0s 5ms/step - loss: -1400801.0000 - accuracy: 0.3203 - val_loss:
-620953.0000 - val_accuracy: 0.3125
Epoch 88/100
26/26 [==============================] - 0s 5ms/step - loss: -1390284.6250 - accuracy: 0.2969 - val_loss:
-704251.8125 - val_accuracy: 0.3438
Epoch 89/100
26/26 [==============================] - 0s 4ms/step - loss: -1585361.6250 - accuracy: 0.3242 - val_loss:
-650247.0000 - val_accuracy: 0.2969
Epoch 90/100
26/26 [==============================] - 0s 5ms/step - loss: -1721550.7500 - accuracy: 0.2969 - val_loss:
-735139.7500 - val_accuracy: 0.3594
Epoch 91/100
26/26 [==============================] - 0s 5ms/step - loss: -1783213.2500 - accuracy: 0.3125 - val_loss:
-812982.6875 - val_accuracy: 0.3125
Epoch 92/100
26/26 [==============================] - 0s 5ms/step - loss: -1845209.2500 - accuracy: 0.3086 - val_loss:
-847776.6250 - val_accuracy: 0.3438
Epoch 93/100
26/26 [==============================] - 0s 5ms/step - loss: -2043956.5000 - accuracy: 0.3203 - val_loss:
-887238.1250 - val_accuracy: 0.2969
Epoch 94/100
26/26 [==============================] - 0s 7ms/step - loss: -2156800.0000 - accuracy: 0.3281 - val_loss:
-990026.5000 - val_accuracy: 0.2969
Epoch 95/100
26/26 [==============================] - 0s 8ms/step - loss: -2288126.7500 - accuracy: 0.3164 - val_loss:
-1102977.1250 - val_accuracy: 0.3281
Epoch 96/100
26/26 [==============================] - 0s 5ms/step - loss: -2503164.2500 - accuracy: 0.3008 - val_loss:
-1122875.5000 - val_accuracy: 0.3125
Epoch 97/100
26/26 [==============================] - 0s 5ms/step - loss: -2564180.2500 - accuracy: 0.3125 - val_loss:
-1201146.8750 - val_accuracy: 0.3438
Epoch 98/100
26/26 [==============================] - 0s 4ms/step - loss: -2812798.0000 - accuracy: 0.3164 - val_loss:
-1261096.6250 - val_accuracy: 0.3125
Epoch 99/100
```

```
26/26 [==============================] - 0s 5ms/step - loss: -2896184.5000 - accuracy: 0.3242 - val_loss: -1327229.6250 - val_accuracy: 0.2969
Epoch 100/100
26/26 [==============================] - 0s 3ms/step - loss: -3152621.7500 - accuracy: 0.3008 - val_loss: -1469980.3750 - val_accuracy: 0.3281
<keras.callbacks.History at 0x7f7b84546070>
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=10,criterion='entropy')
```

```
rfc.fit(x_train,y_train)
```

```
<ipython-input-50-b87bb2ba9825>:1: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  rfc.fit(x_train,y_train)
```

☑ RandomForestClassifier

```
RandomForestClassifier(criterion='entropy', n_estimators=10)
```

```
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)
```

☑ DecisionTreeClassifier

```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

```
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)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/utils/validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was
```

```
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

☑ LogisticRegression

```
LogisticRegression()
```

```
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]
/usr/local/lib/python3.9/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but
LogisticRegression was fitted with feature names
  warnings.warn(
array([2])
```

```
y_pred = dtc.predict([[1,1,121.000000,36.0,0,0,1,0]])
print(y_pred)
(y_pred)
```

```
[2]
/usr/local/lib/python3.9/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but
DecisionTreeClassifier was fitted with feature names
  warnings.warn(
array([2])
```

```
y_pred = rfc.predict([[1,1,121.000000,36.0,0,0,1,0]])

print(y_pred)
(y_pred)
```

```
[2]
/usr/local/lib/python3.9/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
array([2])
```

```
classification.save("ckd.h5")
```

```
y_pred = classification.predict(x_test)
```

```
3/3 [==============================] - 0s 5ms/step
```

```
y_pred
```

```
array([[1.],
 [1.],
 [0.],
 [1.],
 [1.],
 [0.],
 [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.],
 [1.],
 [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], [False], [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], [ True], [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]])
```

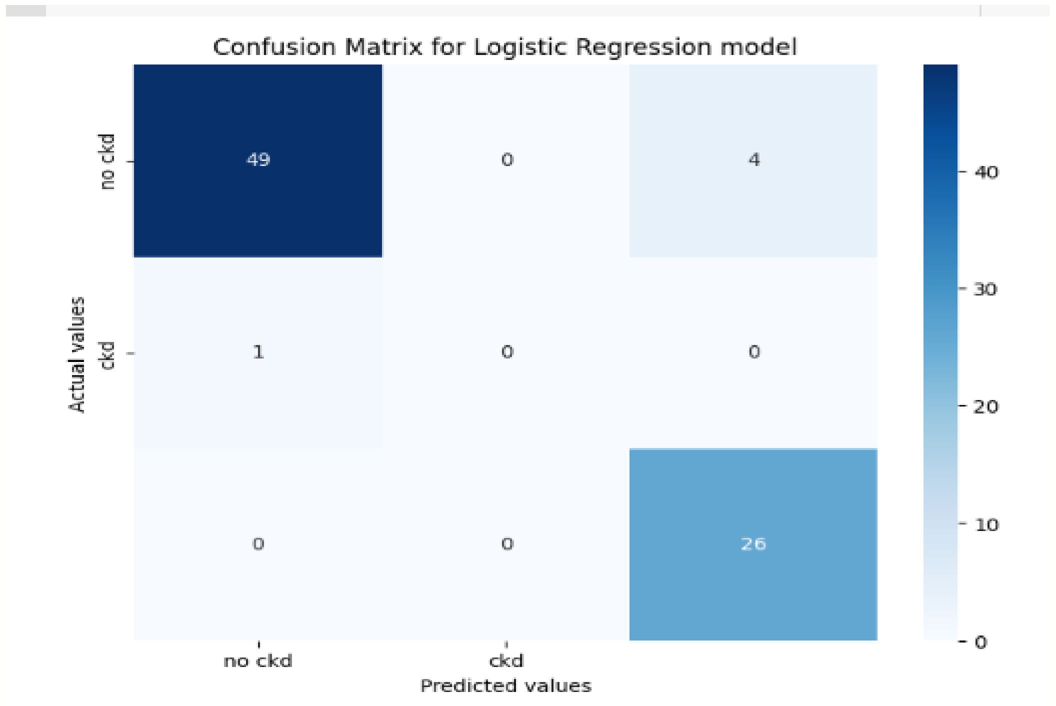
```
def predict_exit(sample_value):
  sample_value = np.array(sample_value)
  sample_value = sample_value.reshape(1,-1)
  sample_value = sc.transform(sample_value)
  return classifier.predict(sample_value)
```

```
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.')

1/1 [==============================] - 0s 139ms/step
prediction: High chance of CKD!
```

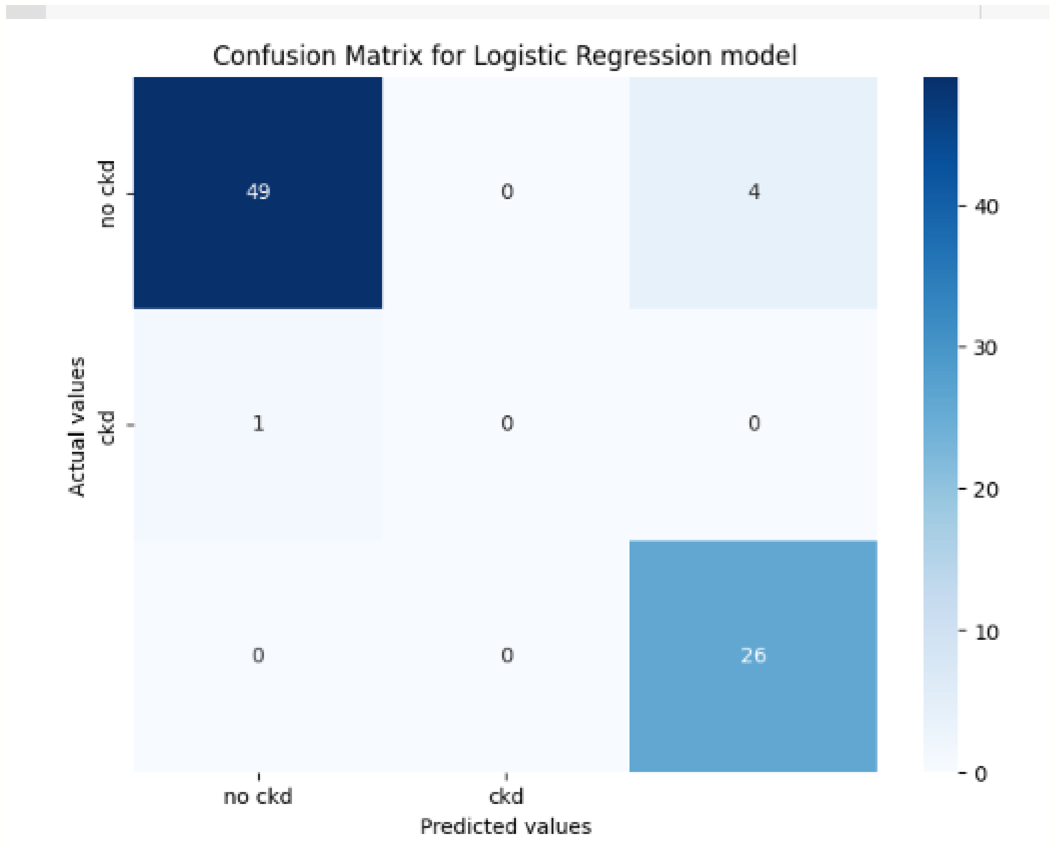
Task5

```
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, sco
ring=scoring)
    clf = model.fit(x_train, y_train)
    y_pred = clf.predict(x_test)
    print(name)
    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)
RF
              precision    recall  f1-score   support

      NO CKD       0.96      0.96      0.96        53
         CKD       0.00      0.00      0.00         1
         CKD       0.93      0.96      0.94        26

    accuracy                           0.95        80
   macro avg       0.63      0.64      0.64        80
weighted avg       0.94      0.95      0.94        80

DecisionTree
              precision    recall  f1-score   support

      NO CKD       0.94      0.96      0.95        53
         CKD       0.00      0.00      0.00         1
         CKD       0.92      0.92      0.92        26
    accuracy                           0.94        80
   macro avg       0.62      0.63      0.63        80
weighted avg       0.93      0.94      0.93        80

   fit_time  score_time  test_accuracy  test_precision_weighted  \
0  0.003376    0.028890       0.906250                 0.908578
1  0.003545    0.023921       0.890625                 0.876603
2  0.003815    0.025869       0.843750                 0.859001
3  0.003978    0.035339       0.843750                 0.863235
4  0.003999    0.023558       0.859375                 0.861819

   test_recall_weighted  test_f1_weighted  test_roc_auc         model
0              0.906250          0.906527           NaN  DecisionTree
1              0.890625          0.883523           NaN  DecisionTree
2              0.843750          0.850885           NaN  DecisionTree
3              0.843750          0.846096           NaN  DecisionTree
4              0.859375          0.854311           NaN  DecisionTree
```

```
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()
```

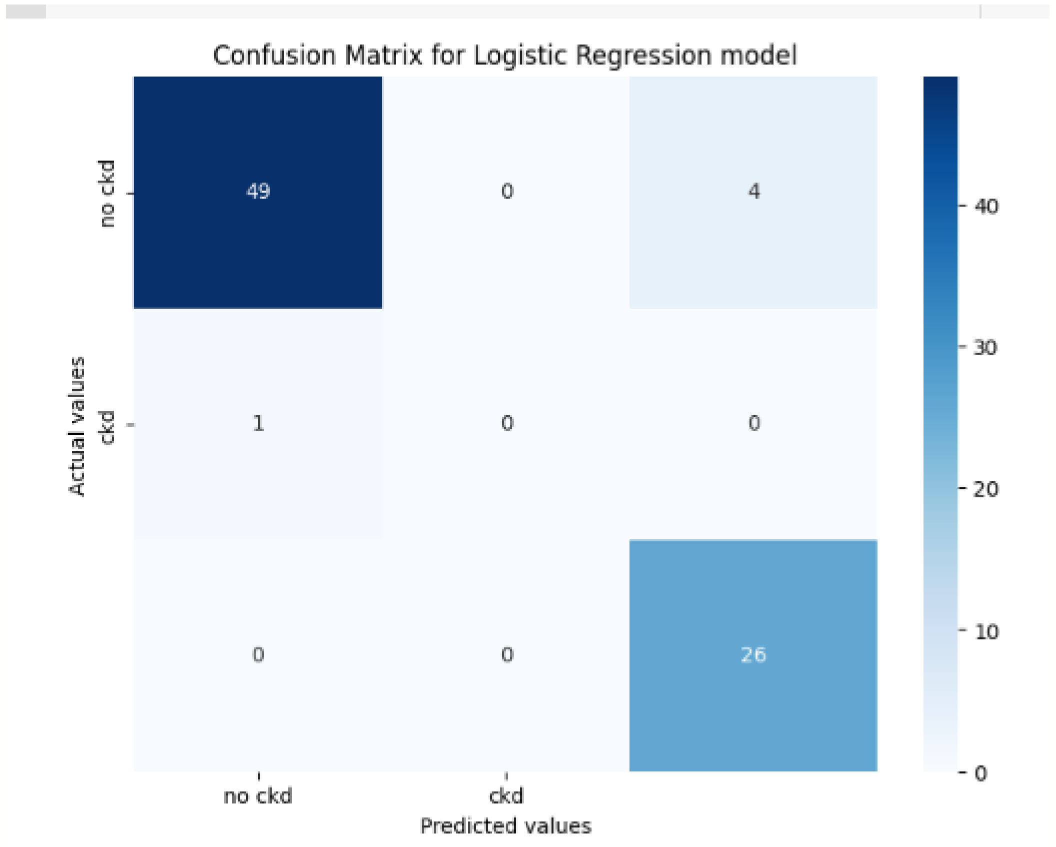


```
print(classification_report(y_test, y_pred))

    precision    recall  f1-score   support
```

```
           0       0.94      0.96      0.95        53
           1       0.00      0.00      0.00         1
           2       0.92      0.92      0.92        26

    accuracy                           0.94        80
   macro avg       0.62      0.63      0.63        80
weighted avg       0.93      0.94      0.93        80
```

```
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='v
alues')
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)
```

Task 6

```
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)
```

THANK YOU