ARIGNAR ANNA GOVERNMENT ARTS COLLEGE

VILLUPURAM – 605 602.

DEPARTMENT OF COMPUTER APPLICATIONS

MACHINE LEARNING WITH PYTHON

Project Title : Early Prediction for Chronic Kidney Disease Detection: A Progressive Approach to Health Management

Team Id : **NM2023TMID16896**

Team Leader : **DHINESHRAJ R**

Team member: VELAVAN S

Team member: ARIGARAN K

Team member: EZHUMALAI A

Team member: GOPI D

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.

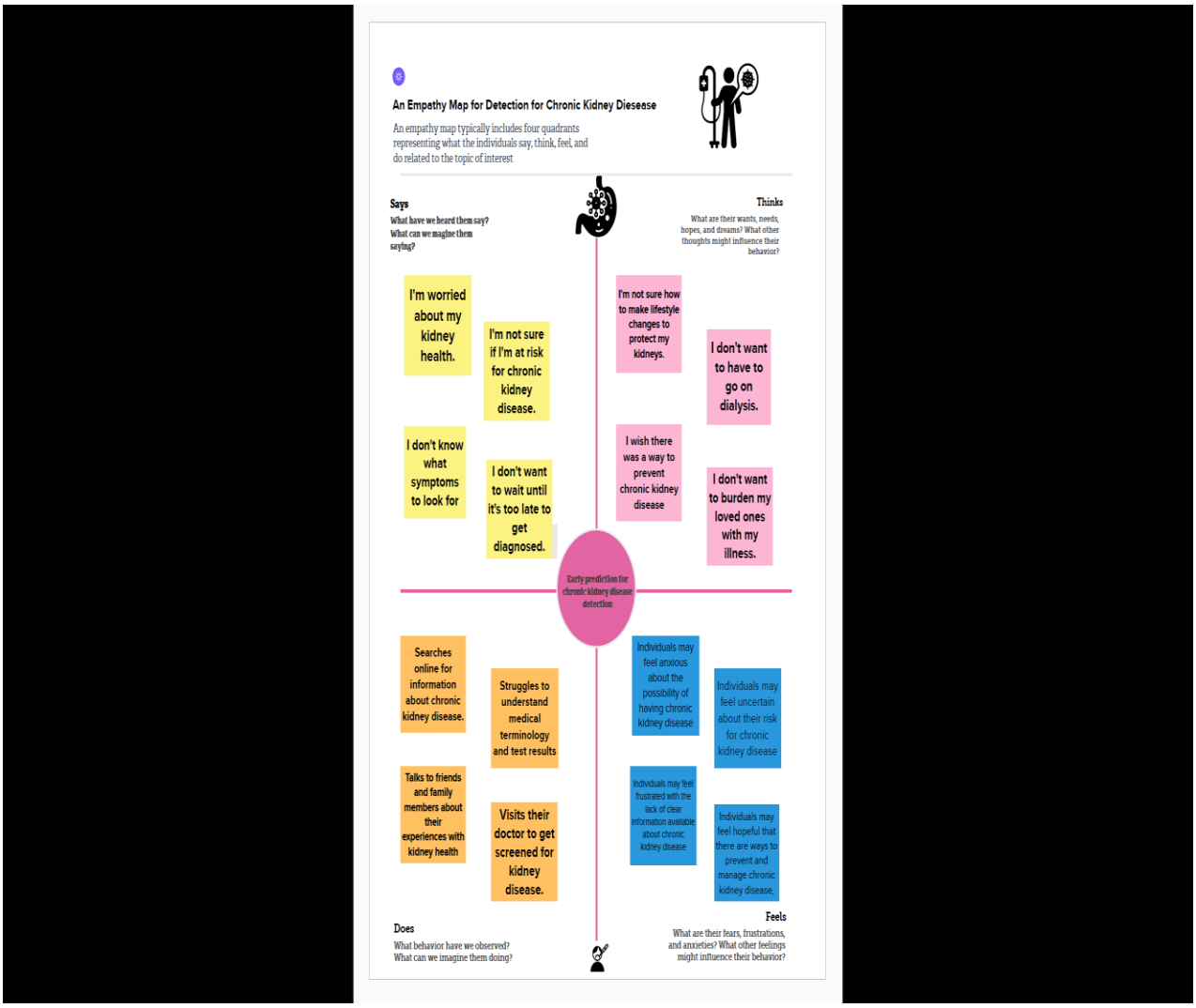
In today world as we know most of the people are facing so many disease and as this can be cured If, we treat people in early stages this project can use a pre trained model to predict the Chronic Kidney disease which can help in treatments of peoples who are suffer from this disease.

Purpose

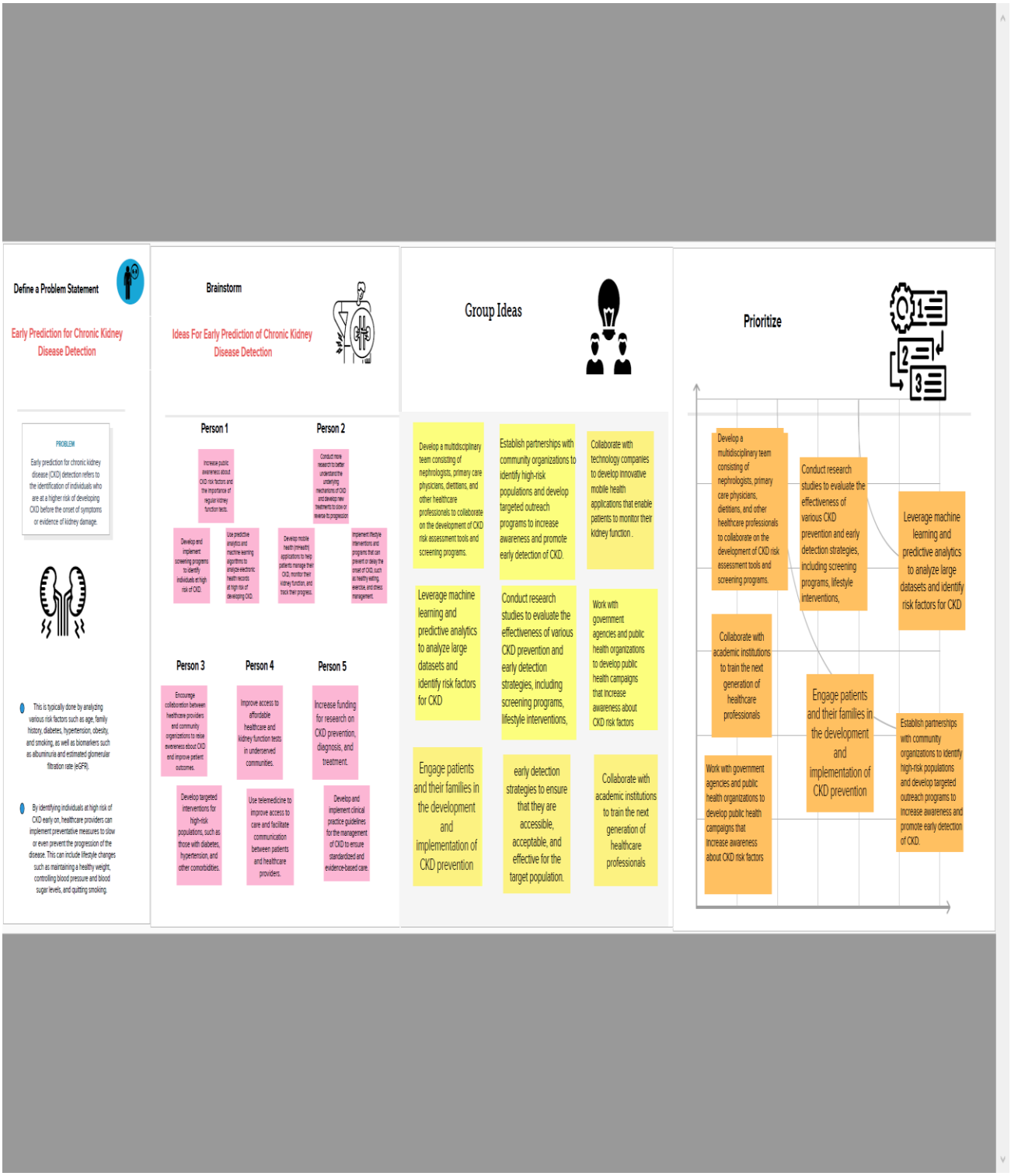
The rationale for testing asymptomatic people for CKD is that earlier detection might allow for the implementation of therapeutic interventions and avoidance of inappropriate exposure to nephrotoxic agents, both of which may slow the progression of CKD to end-stage kidney disease.

Problem Definition & Design Thinking

Empathy Map



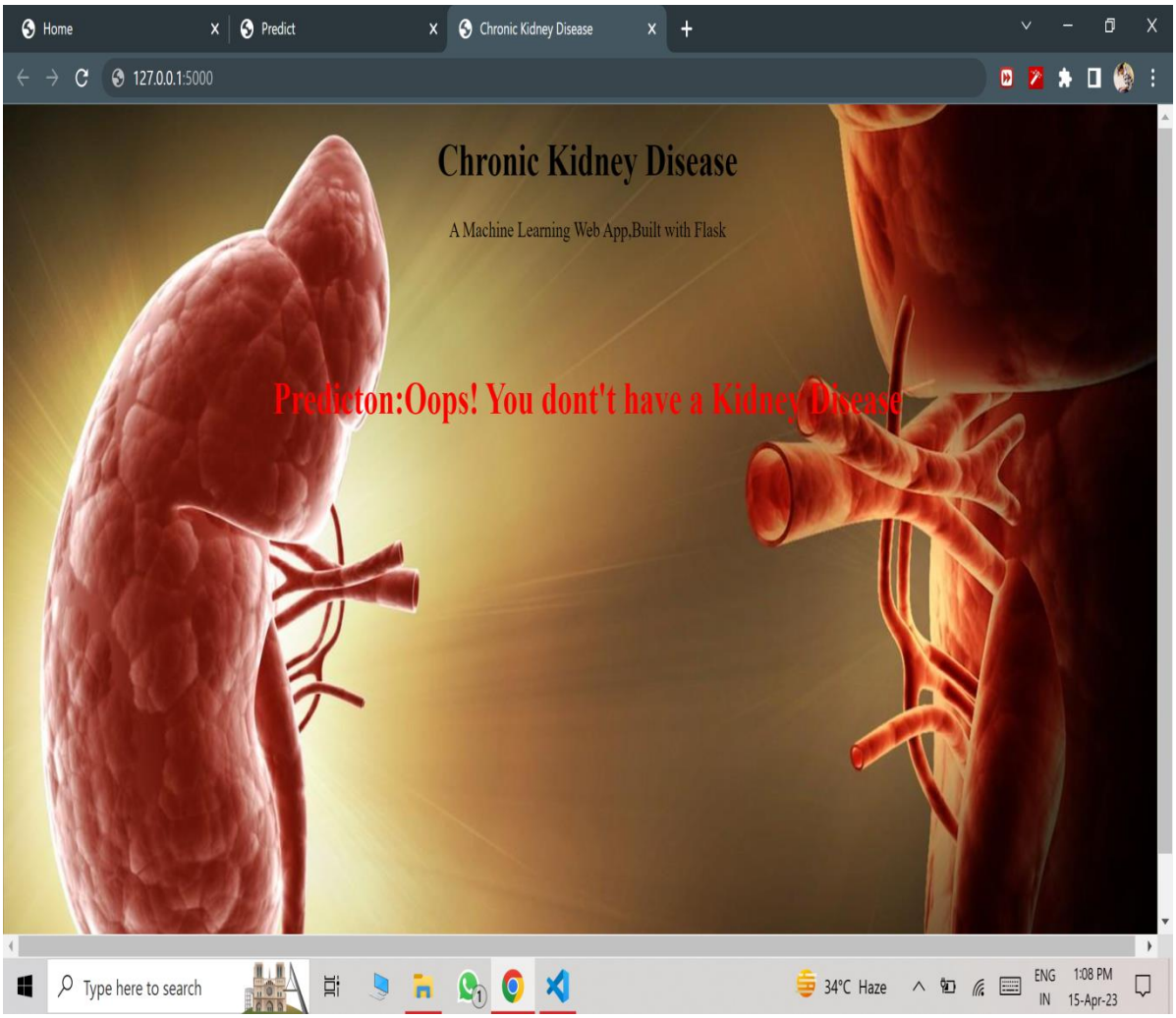
Ideation & Brainstorming Map



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.

Result Image



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.

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.

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 extension to this work is that implementation of deep learning since deep learning provides high-quality performance than machine learning algorithm.

Source Code

Milestone 2:

kidney-disease

April 14, 2023

[1]:
```
#importing Libraries
import pandas as pd #used for data manipulation
import numpy as np #used for numerical analysis
from collections import Counter as c #returns counts of number of classes
import matplotlib.pyplot as plt #used for data visualization
import seaborn as sns #data visualization library
import missingno as msno #finding missing values
from sklearn.metrics import accuracy_score,confusion_matrix #model performance
from sklearn.model_selection import train_test_split #splits data in random␣
↪train and test array
from sklearn.preprocessing import LabelEncoder #encoding the levels of␣
↪categorical features
from sklearn.linear_model import LogisticRegression#classification of algorithms
import pickle #python object hierarchy is converted into a byr stream
```

[3]:
```
#Read the Dataset
data=pd.read_csv("/content/chronic_kidney_disease.csv")
data.head()
```

[3]:

	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 x 26 columns]

[4]:
```
data.drop(["id"],axis=1,inplace=True)
```

[5]:
```
data.columns
```

[5]:
```
Index(['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')
```

[6]:
```
#Data preparation
#Rename The Columns

data.columns=['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
```

[6]:
```
Index(['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')
```

[7]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 25 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   age                     391 non-null    float64
 1   blood_pressure          388 non-null    float64
 2   specific_gravity        353 non-null    float64
 3   albumin                 354 non-null    float64
 4   sugar                   351 non-null    float64
 5   red_blood_cells         248 non-null    object
 6   pus_cell                335 non-null    object
 7   pus_cell_clumps         396 non-null    object
 8   bacteria                396 non-null    object
```

```
 9   blood glucose random     356 non-null    float64
 10  blood_urea               381 non-null    float64
 11  serum_creatinine         383 non-null    float64
 12  sodium                   313 non-null    float64
 13  potassium                312 non-null    float64
 14  hemoglobin               348 non-null    float64
 15  packed_cell_volume       330 non-null    object
 16  white_blood_cell_count   295 non-null    object
 17  red_blood_cell_count     270 non-null    object
 18  hypertension             398 non-null    object
 19  diabetesmellitus         398 non-null    object
 20  coronary_artery_disease  398 non-null    object
 21  appetite                 399 non-null    object
 22  pedal_edema              399 non-null    object
 23  anemia                   399 non-null    object
 24  class                    400 non-null    object
dtypes: float64(11), object(14)
memory usage: 78.2+ KB
```

[8]:
```
data.isnull().sum()
```

[8]:
```
age                          9
blood_pressure              12
specific_gravity            47
albumin                     46
sugar                       49
red_blood_cells            152
pus_cell                    65
pus_cell_clumps              4
bacteria                     4
blood glucose random        44
blood_urea                  19
serum_creatinine            17
sodium                      87
potassium                   88
hemoglobin                  52
packed_cell_volume          70
white_blood_cell_count     105
red_blood_cell_count       130
hypertension                 2
diabetesmellitus             2
coronary_artery_disease      2
appetite                     1
pedal_edema                  1
anemia                       1
class                        0
dtype: int64
```

[9]:
```
data.describe()
```

[9]:
```
              age  blood_pressure  specific_gravity     albumin       sugar  \
count  391.000000      388.000000       353.000000  354.000000  351.000000
mean    51.483376       76.469072         1.017408    1.016949    0.450142
std     17.169714       13.683637         0.005717    1.352679    1.099191
min      2.000000       50.000000         1.005000    0.000000    0.000000
25%     42.000000       70.000000         1.010000    0.000000    0.000000
50%     55.000000       80.000000         1.020000    0.000000    0.000000
75%     64.500000       80.000000         1.020000    2.000000    0.000000
max     90.000000      180.000000         1.025000    5.000000    5.000000

       blood glucose random  blood_urea  serum_creatinine      sodium  \
count            356.000000  381.000000        383.000000  313.000000
mean             148.036517   57.425722          3.072454  137.528754
std               79.281714   50.503006          5.741126   10.408752
min               22.000000    1.500000          0.400000    4.500000
25%               99.000000   27.000000          0.900000  135.000000
50%              121.000000   42.000000          1.300000  138.000000
75%              163.000000   66.000000          2.800000  142.000000
max              490.000000  391.000000         76.000000  163.000000

        potassium  hemoglobin
count  312.000000  348.000000
mean     4.627244   12.526437
std      3.193904    2.912587
min      2.500000    3.100000
25%      3.800000   10.300000
50%      4.400000   12.650000
75%      4.900000   15.000000
max     47.000000   17.800000
```

[10]:
```
data['class'].unique()
```

[10]:
```
array(['ckd', 'ckd\t', 'notckd'], dtype=object)
```

[11]:
```
data['class']=data['class'].replace("ckd\t","ckd")
data['class'].unique()
```

[11]:
```
array(['ckd', 'notckd'], dtype=object)
```

[12]:
```
np.unique(data.dtypes,return_counts=True)
```

[12]:
```
(array([dtype('float64'), dtype('O')], dtype=object), array([11, 14]))
```

[13]:
```
catcols=set(data.dtypes[data.dtypes=='O'].index.values)
print(catcols)
```

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

[14]:
```
for i in catcols:
  print("columns:",i)
  print(c(data[i]))
  print('*'*120+'\n')
```

```
columns: red_blood_cells
Counter({'normal': 201, nan: 152, 'abnormal': 47})
************************************************************************************************************************

columns: packed_cell_volume
Counter({nan: 70, '52': 21, '41': 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: red_blood_cell_count
Counter({nan: 130, '5.2': 18, '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: appetite
Counter({'good': 317, 'poor': 82, nan: 1})
************************************************************************************************************************

columns: pus_cell
Counter({'normal': 259, 'abnormal': 76, nan: 65})
************************************************************************************************************************

columns: anemia
Counter({'no': 339, 'yes': 60, nan: 1})
```

```
**************************************************************************************
**************************************

columns: class
Counter({'ckd': 250, 'notckd': 150})
**************************************************************************************
**************************************

columns: white_blood_cell_count
Counter({nan: 105, '9800': 11, '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: hypertension
Counter({'no': 251, 'yes': 147, nan: 2})
**************************************************************************************
**************************************

columns: coronary_artery_disease
Counter({'no': 362, 'yes': 34, '\tno': 2, nan: 2})
**************************************************************************************
**************************************

columns: diabetesmellitus
Counter({'no': 258, 'yes': 134, '\tno': 3, '\tyes': 2, nan: 2, ' yes': 1})
**************************************************************************************
**************************************

columns: pedal_edema
Counter({'no': 323, 'yes': 76, nan: 1})
**************************************************************************************
**************************************

columns: pus_cell_clumps
Counter({'notpresent': 354, 'present': 42, nan: 4})
```

```
**************************************************************************************
**************************************
```

```
columns: bacteria
Counter({'notpresent': 374, 'present': 22, nan: 4})
```

```
**************************************************************************************
**************************************
```

[15]:
```
catcols.remove('red_blood_cell_count')
catcols.remove('packed_cell_volume')
catcols.remove('white_blood_cell_count')
print(catcols)
```

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

[16]:
```
contcols=set(data.dtypes[data.dtypes!='O'].index.values)
print(contcols)
```

```
{'hemoglobin', 'sugar', 'blood_urea', 'albumin', 'serum_creatinine',
'blood_pressure', 'potassium', 'specific_gravity', 'age', 'blood glucose
random', 'sodium'}
```

[17]:
```
for i in contcols:
  print("Continous Columns:",i)
  print(c(data[i]))
  print('*'*120+'\n')
```

```
Continous Columns: hemoglobin
Counter({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, nan: 1, nan: 1, 12.9: 1, nan: 1, nan: 1, nan: 1, nan: 1, 6.6: 1, nan: 1,
nan: 1, 7.5: 1, nan: 1, nan: 1, 4.8: 1, nan: 1, nan: 1, 7.1: 1, nan: 1, nan: 1,
nan: 1, 9.2: 1, nan: 1, 6.2: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1,
8.2: 1, nan: 1, nan: 1, 6.1: 1, nan: 1, nan: 1, nan: 1, nan: 1, 8.4: 1, nan: 1,
9.0: 1, nan: 1, nan: 1, 10.6: 1, nan: 1, nan: 1, nan: 1, 10.7: 1, nan: 1, 5.5:
1, nan: 1, 5.8: 1, 6.8: 1, 8.5: 1, 7.3: 1, nan: 1, nan: 1, 12.8: 1, nan: 1, nan:
1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, 6.3: 1, nan: 1, 3.1: 1, nan: 1, 17.3:
```

```
1, nan: 1, nan: 1, nan: 1, nan: 1, 17.7: 1, 17.5: 1, nan: 1, 17.6: 1})
********************************************************************************
**************************************

Continous Columns: sugar
Counter({0.0: 290, 2.0: 18, 3.0: 14, 4.0: 13, 1.0: 13, 5.0: 3, nan: 1, nan: 1,
nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1,
nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1,
nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1,
nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1,
nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1})
********************************************************************************
**************************************

Continous Columns: blood_urea
Counter({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, nan: 1,
75.0: 1, 65.0: 1, 103.0: 1, 70.0: 1, 202.0: 1, 114.0: 1, nan: 1, nan: 1, 164.0:
1, 142.0: 1, 391.0: 1, nan: 1, nan: 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, nan: 1, 21.0: 1, 219.0: 1, 166.0:
1, 208.0: 1, 176.0: 1, nan: 1, 145.0: 1, 165.0: 1, 322.0: 1, 235.0: 1, 76.0: 1,
nan: 1, nan: 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, nan: 1, 150.0: 1, nan: 1,
61.0: 1, 57.0: 1, nan: 1, 95.0: 1, 191.0: 1, nan: 1, 93.0: 1, 241.0: 1, 64.0: 1,
79.0: 1, 215.0: 1, 309.0: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1})
********************************************************************************
**************************************

Continous Columns: albumin
Counter({0.0: 199, 1.0: 44, 2.0: 43, 3.0: 43, 4.0: 24, nan: 1, nan: 1, nan: 1,
nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1,
nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, 5.0: 1,
nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1,
nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1,
nan: 1, nan: 1, nan: 1, nan: 1})
********************************************************************************
**************************************

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,
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, nan: 1, 10.8: 1, 5.9: 1, 3.25: 1, nan: 1, 9.7: 1, 6.4: 1, 32.0: 1, nan: 1,
nan: 1, 8.5: 1, 15.0: 1, 3.6: 1, 10.2: 1, 11.5: 1, nan: 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, nan: 1,
nan: 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, nan:
1, 12.8: 1, 11.9: 1, nan: 1, nan: 1, nan: 1, 12.0: 1, nan: 1, 13.4: 1, 15.2: 1,
13.3: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, 0.4: 1})
********************************************************************************
****************************************

Continous Columns: blood_pressure
Counter({80.0: 116, 70.0: 112, 60.0: 71, 90.0: 53, 100.0: 25, 50.0: 5, 110.0: 3,
nan: 1, nan: 1, 140.0: 1, 180.0: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan:
1, nan: 1, 120.0: 1, nan: 1, nan: 1, nan: 1})
********************************************************************************
****************************************

Continous Columns: potassium
Counter({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, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1,
nan: 1, nan: 1, 6.4: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1,
nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, 6.6: 1,
39.0: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, 7.6: 1, nan: 1,
nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1,
nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, 47.0: 1, nan: 1, 5.1: 1, nan: 1,
nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, 2.8: 1,
nan: 1, nan: 1, nan: 1, nan: 1, 2.7: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1,
nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1,
nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1,
nan: 1})
********************************************************************************
****************************************

Continous Columns: specific_gravity
Counter({1.02: 106, 1.01: 84, 1.025: 81, 1.015: 75, 1.005: 7, nan: 1, nan: 1,
nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1,
nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1,
nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1,
nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1,
nan: 1, nan: 1, nan: 1, nan: 1, nan: 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, 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, nan: 1, 82.0: 1, 11.0: 1, 26.0: 1, nan: 1, nan: 1, nan: 1, nan: 1, 81.0:
1, 14.0: 1, 27.0: 1, nan: 1, 83.0: 1, 4.0: 1, 3.0: 1, 6.0: 1, nan: 1, 90.0: 1,
78.0: 1, nan: 1, 2.0: 1, nan: 1, 22.0: 1, 79.0: 1})
********************************************************************************
****************************************

Continous Columns: blood glucose random
Counter({99.0: 10, 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, nan: 1, 423.0: 1, 410.0: 1, 380.0: 1, 157.0: 1, 263.0: 1,
173.0: 1, nan: 1, nan: 1, nan: 1, 156.0: 1, 264.0: 1, nan: 1, 159.0: 1, 270.0:
1, nan: 1, nan: 1, nan: 1, 162.0: 1, nan: 1, 246.0: 1, nan: 1, nan: 1, nan: 1,
nan: 1, 182.0: 1, 146.0: 1, nan: 1, 425.0: 1, 250.0: 1, nan: 1, nan: 1, nan: 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, nan: 1, nan: 1, nan: 1, nan: 1, 323.0: 1, nan: 1,
nan: 1, 90.0: 1, 308.0: 1, 224.0: 1, nan: 1, 268.0: 1, nan: 1, 256.0: 1, nan: 1,
84.0: 1, nan: 1, 288.0: 1, 273.0: 1, 242.0: 1, 148.0: 1, nan: 1, 160.0: 1, nan:
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, nan: 1, nan: 1, 239.0: 1, nan: 1, nan: 1, 184.0: 1, nan: 1, 252.0:
1, 230.0: 1, 341.0: 1, nan: 1, 255.0: 1, nan: 1, 238.0: 1, 248.0: 1, 241.0: 1,
269.0: 1, nan: 1, nan: 1, 201.0: 1, 203.0: 1, 463.0: 1, 176.0: 1, nan: 1, nan:
1, 116.0: 1, nan: 1, nan: 1, nan: 1, 134.0: 1, 87.0: 1, nan: 1})
********************************************************************************
****************************************

Continous Columns: sodium
Counter({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, nan: 1, nan: 1, nan:
1, 111.0: 1, nan: 1, 104.0: 1, nan: 1, nan: 1, nan: 1, nan: 1, 4.5: 1, nan: 1,
129.0: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan:
1, nan: 1, nan: 1, nan: 1, nan: 1, 163.0: 1, nan: 1, nan: 1, nan: 1, nan: 1,
nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1,
```

```
nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1,
nan: 1, 126.0: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan:
1, nan: 1, nan: 1, 115.0: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1,
nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1,
nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1, nan: 1,
nan: 1, nan: 1, nan: 1, nan: 1})
**********************************************************************************
**************************************
```

[18]:
```
contcols.remove('specific_gravity')
contcols.remove('albumin')
contcols.remove('sugar')
print(contcols)
```

```
{'hemoglobin', 'blood_urea', 'serum_creatinine', 'blood_pressure', 'potassium',
'age', 'blood glucose random', 'sodium'}
```

[19]:
```
contcols.add('red_blood_cell_count')
contcols.add('white_blood_cell_count')
contcols.add('packed_cell_volume')
print(contcols)
```

```
{'hemoglobin', 'red_blood_cell_count', 'packed_cell_volume', 'blood_urea',
'serum_creatinine', 'white_blood_cell_count', 'blood_pressure', 'potassium',
'age', 'blood glucose random', 'sodium'}
```

[20]:
```
# adding the categorical columns

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

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

[21]:
```
data['coronary_artery_disease']=data.coronary_artery_disease.
  ↪replace('\tno','no')
c(data['coronary_artery_disease'])
```

[21]:
```
Counter({'no': 364, 'yes': 34, nan: 2})
```

[22]:
```
data['diabetesmellitus']=data.diabetesmellitus.replace(to_replace={'\tno':
  ↪'no','\tyes':'yes','yes':'yes'})
c(data['diabetesmellitus'])
```

[22]:
```
Counter({'yes': 136, 'no': 261, ' yes': 1, nan: 2})
```

[23]:
```
#converting the columns into numeric type

data.packed_cell_volume=pd.to_numeric(data.packed_cell_volume,errors='coerce')
data.white_blood_cell_count=pd.to_numeric(data.
 ↪white_blood_cell_count,errors='coerce')
data.red_blood_cell_count=pd.to_numeric(data.
 ↪red_blood_cell_count,errors='coerce')
```

[24]:
```
data['blood glucose random'].fillna(data['blood glucose random'].
 ↪mean(),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['packed_cell_volume'].fillna(data['packed_cell_volume'].
 ↪mean(),inplace=True)
data['potassium'].fillna(data['potassium'].mean(),inplace=True)
data['red_blood_cell_count'].fillna(data['red_blood_cell_count'].
 ↪mean(),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'].
 ↪mean(),inplace=True)


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

[25]:
```
data.isnull().sum()
```

[25]:
```
age                     0
blood_pressure          0
specific_gravity        0
```

```
albumin                   0
sugar                     0
red_blood_cells           0
pus_cell                  0
pus_cell_clumps           0
bacteria                  0
blood glucose random      0
blood_urea                0
serum_creatinine          0
sodium                    0
potassium                 0
hemoglobin                0
packed_cell_volume        0
white_blood_cell_count    0
red_blood_cell_count      0
hypertension              0
diabetesmellitus          0
coronary_artery_disease   0
appetite                  0
pedal_edema               0
anemia                    0
class                     0
dtype: int64
```

[26]:
```
#Label Encoding of Categorical column
catcols=['hypertension', 'anemia', 'red_blood_cells', 'bacteria', 'pus_cell',␣
↪'diabetesmellitus',
        'appetite', 'coronary_artery_disease', 'pus_cell_clumps',␣
↪'pedal_edema', 'class']
```

[27]:
```
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)
```

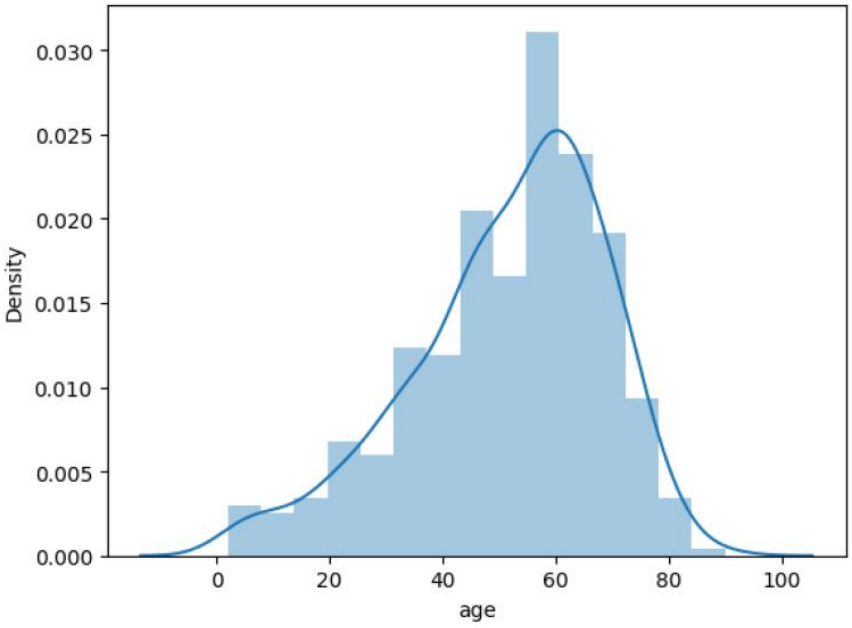
```
LABEL ENCODING OF: hypertension
Counter({'no': 253, 'yes': 147})
Counter({0: 253, 1: 147})
****************************************************************************************
********************
LABEL ENCODING OF: anemia
Counter({'no': 340, 'yes': 60})
Counter({0: 340, 1: 60})
```

```
********************************************************************************
********************
LABEL ENCODING OF: red_blood_cells
Counter({'normal': 353, 'abnormal': 47})
Counter({1: 353, 0: 47})
********************************************************************************
********************
LABEL ENCODING OF: 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})
********************************************************************************
********************
LABEL ENCODING OF: diabetesmellitus
Counter({'no': 263, 'yes': 136, ' yes': 1})
Counter({1: 263, 2: 136, 0: 1})
********************************************************************************
********************
LABEL ENCODING OF: appetite
Counter({'good': 318, 'poor': 82})
Counter({0: 318, 1: 82})
********************************************************************************
********************
LABEL ENCODING OF: coronary_artery_disease
Counter({'no': 366, 'yes': 34})
Counter({0: 366, 1: 34})
********************************************************************************
********************
LABEL ENCODING OF: pus_cell_clumps
Counter({'notpresent': 358, 'present': 42})
Counter({0: 358, 1: 42})
********************************************************************************
********************
LABEL ENCODING OF: pedal_edema
Counter({'no': 324, 'yes': 76})
Counter({0: 324, 1: 76})
********************************************************************************
********************
LABEL ENCODING OF: class
Counter({'ckd': 250, 'notckd': 150})
Counter({0: 250, 1: 150})
********************************************************************************
********************
```

[28]:
```
sns.distplot(data.age)
```

```
<ipython-input-28-868c85374ad7>: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)
```

[28]: `<Axes: xlabel='age', ylabel='Density'>`

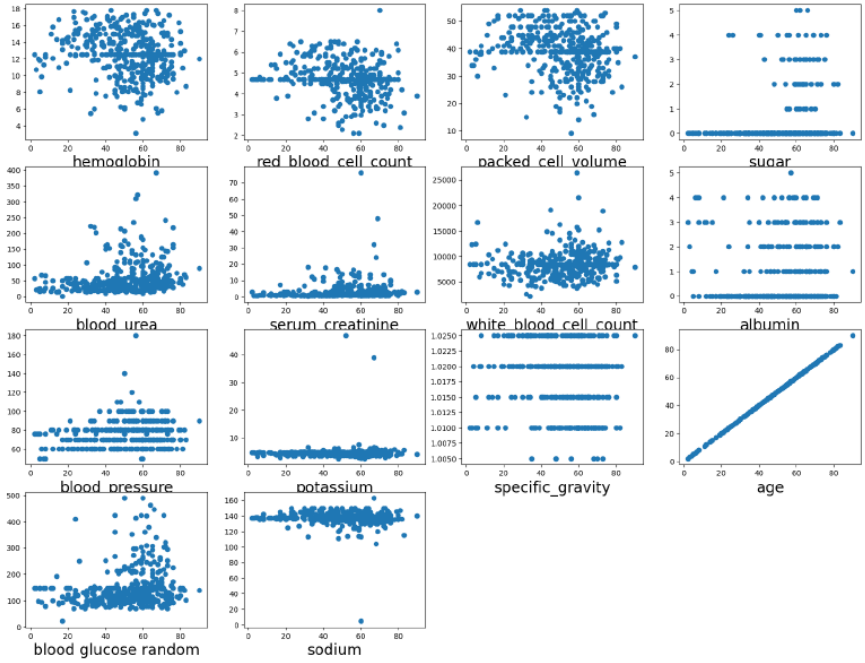


[29]:
```
#Data Visualization
#Age VS Blood pressure

import matplotlib.pyplot as plt
fig=plt.figure(figsize=(5,5))
```

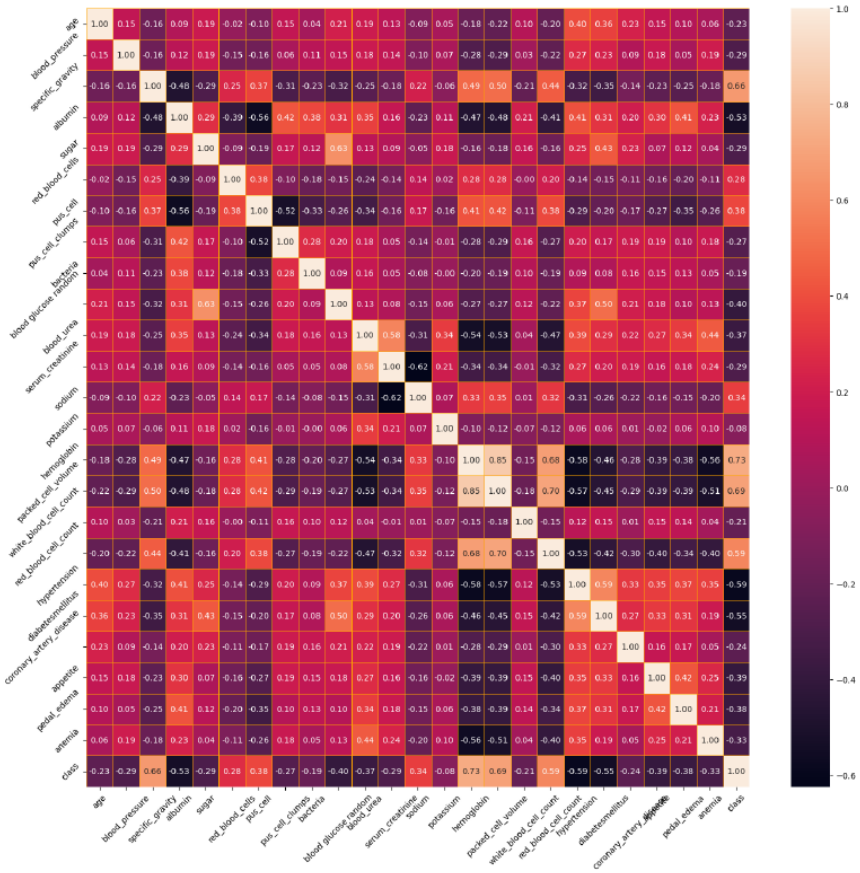
```
plt.scatter(data['age'],data['blood_pressure'],color='blue')
plt.xlabel('age')
plt.ylabel('blood pressure')
plt.title("age vs blood pressure scatter plot")
```

[29]: Text(0.5, 1.0, 'age vs blood pressure scatter plot')

[30]:
```
#Age VS All continuous columns

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

for column in contcols:
  if plotnumber<=14 :
    ax=plt.subplot(4,4,plotnumber)
    plt.scatter(data['age'],data[column])
    plt.xlabel(column,fontsize=20)
  plotnumber+=1
plt.show()
```



[31]:
```
#Finding correlation between independent columns

f,ax=plt.subplots(figsize=(18,17))
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()
```

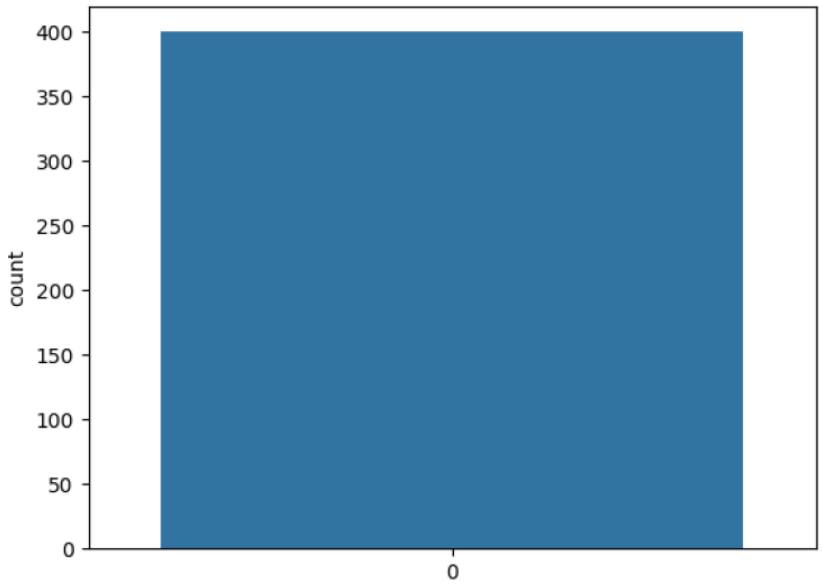


[32]:
```
data['class'].unique()
```

[32]:
```
array([0, 1])
```

[33]:
```
sns.countplot(data['class'])
```

[33]:
```
<Axes: ylabel='count'>
```



[34]:
```
#creating indepentent and dependent

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)
```

[35]:
```
#splitting the data into train and test

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=2)
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(280, 8)
(280, 1)
```

```
(120, 8)
(120, 1)
```

[36]:
```
# Model Building
#TRain the model in multiple algorithms

#ANN Model
import tensorflow
from  tensorflow.keras.models import Sequential
from  tensorflow.keras.layers import Dense
```

[37]:
```
#creating ANN skleton View

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'))
```

[38]:
```
#compiling the ANN model
classification.
 ↪compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

[39]:
```
#training the model
classification.fit(x_train,y_train,batch_size=10,validation_split=0.
 ↪2,epochs=100)
```

```
Epoch 1/100
23/23 [==============================] - 2s 9ms/step - loss: 0.9125 - accuracy:
0.5580 - val_loss: 0.6803 - val_accuracy: 0.6607
Epoch 2/100
23/23 [==============================] - 0s 2ms/step - loss: 0.6131 - accuracy:
0.5759 - val_loss: 0.5487 - val_accuracy: 0.6607
Epoch 3/100
23/23 [==============================] - 0s 2ms/step - loss: 0.5897 - accuracy:
0.6027 - val_loss: 0.5366 - val_accuracy: 0.7143
Epoch 4/100
23/23 [==============================] - 0s 3ms/step - loss: 0.6008 - accuracy:
0.5714 - val_loss: 0.5212 - val_accuracy: 0.6964
Epoch 5/100
23/23 [==============================] - 0s 2ms/step - loss: 0.5531 - accuracy:
0.6473 - val_loss: 0.5250 - val_accuracy: 0.6786
Epoch 6/100
23/23 [==============================] - 0s 3ms/step - loss: 0.6822 - accuracy:
0.5982 - val_loss: 0.5212 - val_accuracy: 0.6607
Epoch 7/100
23/23 [==============================] - 0s 3ms/step - loss: 0.5694 - accuracy:
```

```
0.6295 - val_loss: 0.6512 - val_accuracy: 0.6607
Epoch 8/100
23/23 [==============================] - 0s 2ms/step - loss: 0.6148 - accuracy:
0.6295 - val_loss: 0.5280 - val_accuracy: 0.6607
Epoch 9/100
23/23 [==============================] - 0s 2ms/step - loss: 0.5420 - accuracy:
0.6652 - val_loss: 0.5058 - val_accuracy: 0.6607
Epoch 10/100
23/23 [==============================] - 0s 2ms/step - loss: 0.5525 - accuracy:
0.6116 - val_loss: 0.6184 - val_accuracy: 0.6607
Epoch 11/100
23/23 [==============================] - 0s 2ms/step - loss: 0.5948 - accuracy:
0.6250 - val_loss: 0.5033 - val_accuracy: 0.6429
Epoch 12/100
23/23 [==============================] - 0s 3ms/step - loss: 0.5642 - accuracy:
0.6339 - val_loss: 0.4942 - val_accuracy: 0.6607
Epoch 13/100
23/23 [==============================] - 0s 3ms/step - loss: 0.5019 - accuracy:
0.7009 - val_loss: 0.4952 - val_accuracy: 0.6250
Epoch 14/100
23/23 [==============================] - 0s 2ms/step - loss: 0.5065 - accuracy:
0.6741 - val_loss: 0.4973 - val_accuracy: 0.6429
Epoch 15/100
23/23 [==============================] - 0s 2ms/step - loss: 0.5178 - accuracy:
0.6429 - val_loss: 0.4862 - val_accuracy: 0.6250
Epoch 16/100
23/23 [==============================] - 0s 3ms/step - loss: 0.4878 - accuracy:
0.7143 - val_loss: 0.4842 - val_accuracy: 0.7143
Epoch 17/100
23/23 [==============================] - 0s 3ms/step - loss: 0.4924 - accuracy:
0.6830 - val_loss: 0.4896 - val_accuracy: 0.7679
Epoch 18/100
23/23 [==============================] - 0s 3ms/step - loss: 0.4742 - accuracy:
0.7455 - val_loss: 0.4802 - val_accuracy: 0.6964
Epoch 19/100
23/23 [==============================] - 0s 3ms/step - loss: 0.4653 - accuracy:
0.7634 - val_loss: 0.4636 - val_accuracy: 0.6964
Epoch 20/100
23/23 [==============================] - 0s 2ms/step - loss: 0.4665 - accuracy:
0.7054 - val_loss: 0.4600 - val_accuracy: 0.6786
Epoch 21/100
23/23 [==============================] - 0s 3ms/step - loss: 0.4686 - accuracy:
0.7857 - val_loss: 0.5542 - val_accuracy: 0.6607
Epoch 22/100
23/23 [==============================] - 0s 3ms/step - loss: 0.4735 - accuracy:
0.7009 - val_loss: 0.5090 - val_accuracy: 0.6786
Epoch 23/100
23/23 [==============================] - 0s 2ms/step - loss: 0.4486 - accuracy:
```

```
0.7634 - val_loss: 0.4406 - val_accuracy: 0.7679
Epoch 24/100
23/23 [==============================] - 0s 2ms/step - loss: 0.4383 - accuracy:
0.7545 - val_loss: 0.4478 - val_accuracy: 0.8036
Epoch 25/100
23/23 [==============================] - 0s 3ms/step - loss: 0.4486 - accuracy:
0.7634 - val_loss: 0.6424 - val_accuracy: 0.6250
Epoch 26/100
23/23 [==============================] - 0s 4ms/step - loss: 0.4911 - accuracy:
0.7321 - val_loss: 0.4658 - val_accuracy: 0.6250
Epoch 27/100
23/23 [==============================] - 0s 2ms/step - loss: 0.5191 - accuracy:
0.7143 - val_loss: 0.4875 - val_accuracy: 0.7857
Epoch 28/100
23/23 [==============================] - 0s 3ms/step - loss: 0.4291 - accuracy:
0.7946 - val_loss: 0.5017 - val_accuracy: 0.7143
Epoch 29/100
23/23 [==============================] - 0s 3ms/step - loss: 0.4377 - accuracy:
0.7812 - val_loss: 0.4396 - val_accuracy: 0.6250
Epoch 30/100
23/23 [==============================] - 0s 3ms/step - loss: 0.4103 - accuracy:
0.8125 - val_loss: 0.4090 - val_accuracy: 0.8036
Epoch 31/100
23/23 [==============================] - 0s 2ms/step - loss: 0.4037 - accuracy:
0.8125 - val_loss: 0.4081 - val_accuracy: 0.7857
Epoch 32/100
23/23 [==============================] - 0s 2ms/step - loss: 0.3826 - accuracy:
0.8348 - val_loss: 0.4184 - val_accuracy: 0.6786
Epoch 33/100
23/23 [==============================] - 0s 3ms/step - loss: 0.4174 - accuracy:
0.7589 - val_loss: 0.4050 - val_accuracy: 0.7500
Epoch 34/100
23/23 [==============================] - 0s 2ms/step - loss: 0.3813 - accuracy:
0.8080 - val_loss: 0.4181 - val_accuracy: 0.6607
Epoch 35/100
23/23 [==============================] - 0s 3ms/step - loss: 0.4094 - accuracy:
0.7902 - val_loss: 0.3861 - val_accuracy: 0.8750
Epoch 36/100
23/23 [==============================] - 0s 2ms/step - loss: 0.3895 - accuracy:
0.7857 - val_loss: 0.3916 - val_accuracy: 0.8750
Epoch 37/100
23/23 [==============================] - 0s 2ms/step - loss: 0.3785 - accuracy:
0.8080 - val_loss: 0.5989 - val_accuracy: 0.6607
Epoch 38/100
23/23 [==============================] - 0s 2ms/step - loss: 0.4337 - accuracy:
0.7723 - val_loss: 0.3682 - val_accuracy: 0.8750
Epoch 39/100
23/23 [==============================] - 0s 3ms/step - loss: 0.3829 - accuracy:
```

```
0.7857 - val_loss: 0.3803 - val_accuracy: 0.8929
Epoch 40/100
23/23 [==============================] - 0s 2ms/step - loss: 0.3764 - accuracy:
0.8125 - val_loss: 0.3905 - val_accuracy: 0.6964
Epoch 41/100
23/23 [==============================] - 0s 3ms/step - loss: 0.3434 - accuracy:
0.8571 - val_loss: 0.4170 - val_accuracy: 0.8393
Epoch 42/100
23/23 [==============================] - 0s 3ms/step - loss: 0.3297 - accuracy:
0.8438 - val_loss: 0.4199 - val_accuracy: 0.6786
Epoch 43/100
23/23 [==============================] - 0s 2ms/step - loss: 0.3745 - accuracy:
0.8036 - val_loss: 0.3719 - val_accuracy: 0.8929
Epoch 44/100
23/23 [==============================] - 0s 3ms/step - loss: 0.3168 - accuracy:
0.8839 - val_loss: 0.3821 - val_accuracy: 0.8750
Epoch 45/100
23/23 [==============================] - 0s 3ms/step - loss: 0.3314 - accuracy:
0.8616 - val_loss: 0.3742 - val_accuracy: 0.8750
Epoch 46/100
23/23 [==============================] - 0s 2ms/step - loss: 0.3581 - accuracy:
0.8304 - val_loss: 0.3479 - val_accuracy: 0.8750
Epoch 47/100
23/23 [==============================] - 0s 3ms/step - loss: 0.2925 - accuracy:
0.8884 - val_loss: 0.3607 - val_accuracy: 0.8929
Epoch 48/100
23/23 [==============================] - 0s 3ms/step - loss: 0.3019 - accuracy:
0.8705 - val_loss: 0.3563 - val_accuracy: 0.8750
Epoch 49/100
23/23 [==============================] - 0s 3ms/step - loss: 0.3534 - accuracy:
0.8080 - val_loss: 0.3755 - val_accuracy: 0.8571
Epoch 50/100
23/23 [==============================] - 0s 3ms/step - loss: 0.3295 - accuracy:
0.8393 - val_loss: 0.4020 - val_accuracy: 0.8571
Epoch 51/100
23/23 [==============================] - 0s 3ms/step - loss: 0.3906 - accuracy:
0.7946 - val_loss: 0.3387 - val_accuracy: 0.8750
Epoch 52/100
23/23 [==============================] - 0s 2ms/step - loss: 0.3271 - accuracy:
0.8438 - val_loss: 0.3695 - val_accuracy: 0.8750
Epoch 53/100
23/23 [==============================] - 0s 3ms/step - loss: 0.3503 - accuracy:
0.8527 - val_loss: 0.3480 - val_accuracy: 0.8571
Epoch 54/100
23/23 [==============================] - 0s 3ms/step - loss: 0.3154 - accuracy:
0.8393 - val_loss: 0.4560 - val_accuracy: 0.8393
Epoch 55/100
23/23 [==============================] - 0s 3ms/step - loss: 0.2782 - accuracy:
```

```
0.8795 - val_loss: 0.4491 - val_accuracy: 0.8571
Epoch 56/100
23/23 [==============================] - 0s 3ms/step - loss: 0.2690 - accuracy:
0.8929 - val_loss: 0.5520 - val_accuracy: 0.6429
Epoch 57/100
23/23 [==============================] - 0s 3ms/step - loss: 0.4089 - accuracy:
0.7902 - val_loss: 0.5094 - val_accuracy: 0.7857
Epoch 58/100
23/23 [==============================] - 0s 3ms/step - loss: 0.3579 - accuracy:
0.8527 - val_loss: 0.3604 - val_accuracy: 0.8571
Epoch 59/100
23/23 [==============================] - 0s 3ms/step - loss: 0.2863 - accuracy:
0.8973 - val_loss: 0.3507 - val_accuracy: 0.8929
Epoch 60/100
23/23 [==============================] - 0s 3ms/step - loss: 0.2772 - accuracy:
0.8661 - val_loss: 0.3820 - val_accuracy: 0.8750
Epoch 61/100
23/23 [==============================] - 0s 3ms/step - loss: 0.2890 - accuracy:
0.8705 - val_loss: 0.3809 - val_accuracy: 0.8750
Epoch 62/100
23/23 [==============================] - 0s 3ms/step - loss: 0.3348 - accuracy:
0.8304 - val_loss: 0.4085 - val_accuracy: 0.8571
Epoch 63/100
23/23 [==============================] - 0s 2ms/step - loss: 0.3321 - accuracy:
0.8304 - val_loss: 0.4566 - val_accuracy: 0.6429
Epoch 64/100
23/23 [==============================] - 0s 2ms/step - loss: 0.3128 - accuracy:
0.8527 - val_loss: 0.3375 - val_accuracy: 0.8750
Epoch 65/100
23/23 [==============================] - 0s 3ms/step - loss: 0.2944 - accuracy:
0.8750 - val_loss: 0.3387 - val_accuracy: 0.8929
Epoch 66/100
23/23 [==============================] - 0s 3ms/step - loss: 0.2755 - accuracy:
0.8750 - val_loss: 0.3862 - val_accuracy: 0.8750
Epoch 67/100
23/23 [==============================] - 0s 3ms/step - loss: 0.2574 - accuracy:
0.8839 - val_loss: 0.3703 - val_accuracy: 0.8750
Epoch 68/100
23/23 [==============================] - 0s 3ms/step - loss: 0.2716 - accuracy:
0.8884 - val_loss: 0.3519 - val_accuracy: 0.8929
Epoch 69/100
23/23 [==============================] - 0s 2ms/step - loss: 0.2577 - accuracy:
0.8795 - val_loss: 0.3493 - val_accuracy: 0.8750
Epoch 70/100
23/23 [==============================] - 0s 3ms/step - loss: 0.2809 - accuracy:
0.8795 - val_loss: 0.3471 - val_accuracy: 0.8750
Epoch 71/100
23/23 [==============================] - 0s 3ms/step - loss: 0.2731 - accuracy:
```

```
0.8973 - val_loss: 0.3333 - val_accuracy: 0.9107
Epoch 72/100
23/23 [==============================] - 0s 3ms/step - loss: 0.3065 - accuracy:
0.8705 - val_loss: 0.3317 - val_accuracy: 0.8750
Epoch 73/100
23/23 [==============================] - 0s 2ms/step - loss: 0.2599 - accuracy:
0.9062 - val_loss: 0.3808 - val_accuracy: 0.8571
Epoch 74/100
23/23 [==============================] - 0s 3ms/step - loss: 0.2833 - accuracy:
0.8750 - val_loss: 0.3845 - val_accuracy: 0.7321
Epoch 75/100
23/23 [==============================] - 0s 2ms/step - loss: 0.2557 - accuracy:
0.8973 - val_loss: 0.3417 - val_accuracy: 0.8750
Epoch 76/100
23/23 [==============================] - 0s 3ms/step - loss: 0.2699 - accuracy:
0.8795 - val_loss: 0.4810 - val_accuracy: 0.7857
Epoch 77/100
23/23 [==============================] - 0s 2ms/step - loss: 0.3054 - accuracy:
0.8438 - val_loss: 0.3174 - val_accuracy: 0.8750
Epoch 78/100
23/23 [==============================] - 0s 3ms/step - loss: 0.3476 - accuracy:
0.8393 - val_loss: 0.8189 - val_accuracy: 0.6964
Epoch 79/100
23/23 [==============================] - 0s 2ms/step - loss: 0.3991 - accuracy:
0.7902 - val_loss: 0.3520 - val_accuracy: 0.7679
Epoch 80/100
23/23 [==============================] - 0s 2ms/step - loss: 0.2979 - accuracy:
0.8571 - val_loss: 0.3352 - val_accuracy: 0.8750
Epoch 81/100
23/23 [==============================] - 0s 2ms/step - loss: 0.2619 - accuracy:
0.8839 - val_loss: 0.3340 - val_accuracy: 0.8571
Epoch 82/100
23/23 [==============================] - 0s 2ms/step - loss: 0.2708 - accuracy:
0.8884 - val_loss: 0.3596 - val_accuracy: 0.8571
Epoch 83/100
23/23 [==============================] - 0s 3ms/step - loss: 0.2447 - accuracy:
0.9107 - val_loss: 0.3253 - val_accuracy: 0.9107
Epoch 84/100
23/23 [==============================] - 0s 3ms/step - loss: 0.2507 - accuracy:
0.8884 - val_loss: 0.3442 - val_accuracy: 0.9107
Epoch 85/100
23/23 [==============================] - 0s 2ms/step - loss: 0.2385 - accuracy:
0.9018 - val_loss: 0.3113 - val_accuracy: 0.9107
Epoch 86/100
23/23 [==============================] - 0s 2ms/step - loss: 0.2747 - accuracy:
0.8929 - val_loss: 0.3089 - val_accuracy: 0.8929
Epoch 87/100
23/23 [==============================] - 0s 3ms/step - loss: 0.2769 - accuracy:
```

```
0.8839 - val_loss: 0.3133 - val_accuracy: 0.9107
Epoch 88/100
23/23 [==============================] - 0s 3ms/step - loss: 0.2356 - accuracy:
0.9196 - val_loss: 0.3434 - val_accuracy: 0.8750
Epoch 89/100
23/23 [==============================] - 0s 4ms/step - loss: 0.2418 - accuracy:
0.9018 - val_loss: 0.3575 - val_accuracy: 0.8750
Epoch 90/100
23/23 [==============================] - 0s 3ms/step - loss: 0.2268 - accuracy:
0.9062 - val_loss: 0.3731 - val_accuracy: 0.8750
Epoch 91/100
23/23 [==============================] - 0s 3ms/step - loss: 0.2295 - accuracy:
0.8839 - val_loss: 0.5058 - val_accuracy: 0.6786
Epoch 92/100
23/23 [==============================] - 0s 3ms/step - loss: 0.3236 - accuracy:
0.8527 - val_loss: 0.3618 - val_accuracy: 0.8571
Epoch 93/100
23/23 [==============================] - 0s 3ms/step - loss: 0.2491 - accuracy:
0.9018 - val_loss: 0.3280 - val_accuracy: 0.8929
Epoch 94/100
23/23 [==============================] - 0s 3ms/step - loss: 0.2326 - accuracy:
0.9062 - val_loss: 0.3272 - val_accuracy: 0.8929
Epoch 95/100
23/23 [==============================] - 0s 4ms/step - loss: 0.2786 - accuracy:
0.8839 - val_loss: 0.3242 - val_accuracy: 0.8571
Epoch 96/100
23/23 [==============================] - 0s 3ms/step - loss: 0.2307 - accuracy:
0.9062 - val_loss: 0.3563 - val_accuracy: 0.7857
Epoch 97/100
23/23 [==============================] - 0s 2ms/step - loss: 0.2901 - accuracy:
0.8750 - val_loss: 0.3540 - val_accuracy: 0.8750
Epoch 98/100
23/23 [==============================] - 0s 3ms/step - loss: 0.2407 - accuracy:
0.9062 - val_loss: 0.3668 - val_accuracy: 0.8214
Epoch 99/100
23/23 [==============================] - 0s 3ms/step - loss: 0.3160 - accuracy:
0.8705 - val_loss: 0.3164 - val_accuracy: 0.8929
Epoch 100/100
23/23 [==============================] - 0s 3ms/step - loss: 0.2770 - accuracy:
0.8839 - val_loss: 0.3741 - val_accuracy: 0.7679
```

[39]:
```
<keras.callbacks.History at 0x7ff3ec394c10>
```

[40]:
```
#Random Forest Model

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

[41]:
```
rfc.fit(x_train,y_train)
```

```
<ipython-input-41-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)
```

[41]:
```
RandomForestClassifier(criterion='entropy', n_estimators=10)
```

[42]:
```
y_predict=rfc.predict(x_test)
```

[43]:
```
y_predict_train=rfc.predict(x_train)
```

[44]:
```
#Decision Tree Model

from sklearn.tree import DecisionTreeClassifier
dtc=DecisionTreeClassifier(max_depth=4,splitter='best',criterion='entropy')
```

[45]:
```
dtc.fit(x_train,y_train)
```

[45]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

[46]:
```
y_predict=dtc.predict(x_test)
y_predict
```

[46]:
```
array([0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1,
       0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0,
       0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1,
       1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0,
       1, 0, 1, 0, 1, 0, 0, 1, 1, 0])
```

[47]:
```
y_predict_train=dtc.predict(x_train)
```

[48]:
```
##Logistic Regression
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)
```

[48]:
```
LogisticRegression()
```

[49]:
```
#predicting our output  with model which we build
```

[50]:
```
#logistic Regression
from sklearn.metrics import accuracy_score, classification_report
y_predict= lgr.predict([[129,99,1,0,0,1,0,1]])
```

```
/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(
```

[51]:
```
print(y_predict)
c(y_predict)
```

```
[1]
```

[51]:
```
Counter({1: 1})
```

[52]:
```
y_predict=lgr.predict(x_test)
accuracy_score(y_test,y_predict)
```

[52]:
```
0.9
```

[53]:
```
#testing the model


#Random forest classifier

y_predict=rfc.predict([[1,1,121,000,36.,0,0,1]])

print(y_predict)
y_predict
```

```
[0]
```

```
/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(
```

[53]:
```
array([0])
```

[54]:
```
#Decision tree classifier

y_predict=dtc.predict([[1,1,121,000,36.,0,0,1]])

print(y_predict)
y_predict
```

```
[0]
```

```
/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(
```

[54]:
```
array([0])
```

[55]:
```
y_predict=lgr.predict([[1,1,121.000000,36.0,0,0,1,0]])
print(y_predict)
y_predict
```

```
[1]
```

```
/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(
```

[55]:
```
array([1])
```

[56]:
```
classification.save("ckd.h5")
```

[57]:
```
y_predict = classification.predict(x_test)

y_predict
```

```
4/4 [==============================] - 0s 2ms/step
```

[57]:
```
array([[1.0008919e-03],
       [2.9427707e-02],
       [2.5963090e-10],
       [9.1551123e-03],
       [7.9761815e-01],
       [4.7365866e-06],
       [2.6207253e-08],
       [4.4834405e-07],
       [5.4398024e-01],
       [7.6936358e-06],
       [4.9709044e-33],
       [6.7185098e-04],
       [3.6107621e-01],
       [3.9200738e-01],
       [3.3040407e-27],
       [6.0049032e-14],
       [0.0000000e+00],
       [5.9966922e-01],
       [5.0358713e-01],
```

```
[9.6401171e-09],
[8.3113563e-01],
[4.1347301e-01],
[0.0000000e+00],
[3.6476058e-01],
[3.8444514e-06],
[6.5729296e-01],
[7.7306173e-23],
[5.3621135e-10],
[9.0010256e-01],
[1.0591455e-04],
[6.2898434e-05],
[6.1352485e-01],
[6.2051141e-03],
[3.4240885e-03],
[4.5251403e-09],
[4.1257488e-04],
[8.1279647e-01],
[5.2320212e-04],
[3.2894945e-01],
[8.8876206e-01],
[4.9833343e-05],
[2.7040321e-01],
[2.5878518e-04],
[1.1230054e-03],
[4.9250519e-09],
[7.0755512e-01],
[1.5302639e-05],
[7.4643809e-01],
[5.1389235e-01],
[5.6326032e-02],
[6.9304975e-04],
[5.9272497e-05],
[7.0952816e-24],
[4.6470478e-01],
[2.3024929e-06],
[3.0405514e-03],
[2.6061932e-06],
[7.4744582e-01],
[7.1124905e-01],
[1.1582615e-28],
[7.7674555e-04],
[4.8149478e-01],
[7.6979494e-01],
[8.5867941e-06],
[2.0051445e-03],
[1.2946904e-03],
```

```
[1.4498310e-06],
[4.3328393e-01],
[4.7728297e-04],
[7.2224391e-01],
[8.7237370e-01],
[8.0620005e-08],
[5.0770207e-03],
[3.7546635e-01],
[1.5030718e-01],
[2.3210241e-05],
[3.7823909e-05],
[1.7968147e-01],
[5.0659072e-01],
[1.3580698e-03],
[6.1232126e-01],
[6.9388860e-01],
[1.3950574e-06],
[3.6230627e-01],
[2.0818906e-03],
[7.8938323e-01],
[7.7758580e-01],
[2.8920609e-01],
[6.8539172e-01],
[6.3043368e-01],
[7.2605595e-02],
[4.3460135e-05],
[8.2914633e-01],
[5.5926698e-01],
[1.3737418e-05],
[8.1218064e-01],
[1.0963844e-06],
[2.3290324e-12],
[4.8020523e-04],
[9.4144695e-02],
[9.4144695e-02],
[8.4940380e-01],
[4.5657263e-04],
[3.5682655e-04],
[2.8377550e-10],
[1.5662390e-01],
[7.7813224e-17],
[2.1343010e-03],
[1.3664159e-01],
[2.2485382e-03],
[7.4178582e-01],
[9.6627728e-06],
[5.5331647e-01],
```

```
       [4.0624700e-06],
       [6.3366348e-01],
       [1.8987850e-04],
       [7.3095458e-07],
       [8.4843451e-01],
       [8.3055663e-01],
       [1.1739837e-10]], dtype=float32)
```

[58]:
```
y_predict = (y_predict > 0.5)
y_predict
```

[58]:
```
array([[False],
       [False],
       [False],
       [False],
       [ True],
       [False],
       [False],
       [False],
       [ True],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [ True],
       [ True],
       [False],
       [ True],
       [False],
       [False],
       [False],
       [False],
       [ True],
       [False],
       [False],
       [ True],
       [False],
       [False],
       [ True],
       [False],
       [False],
       [False],
       [False],
```

```
[ True],
[False],
[False],
[ True],
[False],
[False],
[False],
[False],
[False],
[ True],
[False],
[ True],
[ True],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[ True],
[ True],
[False],
[False],
[False],
[ True],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[ True],
[ True],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[ True],
[False],
[ True],
[ True],
[False],
```

```
       [False],
       [False],
       [ True],
       [ True],
       [False],
       [ True],
       [ True],
       [False],
       [False],
       [ True],
       [ True],
       [False],
       [ True],
       [False],
       [False],
       [False],
       [False],
       [False],
       [ True],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [ True],
       [False],
       [ True],
       [False],
       [ True],
       [False],
       [False],
       [ True],
       [ True],
       [False]])
```

[59]:
```
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 59ms/step
prediction: Low chance of CKD.
```

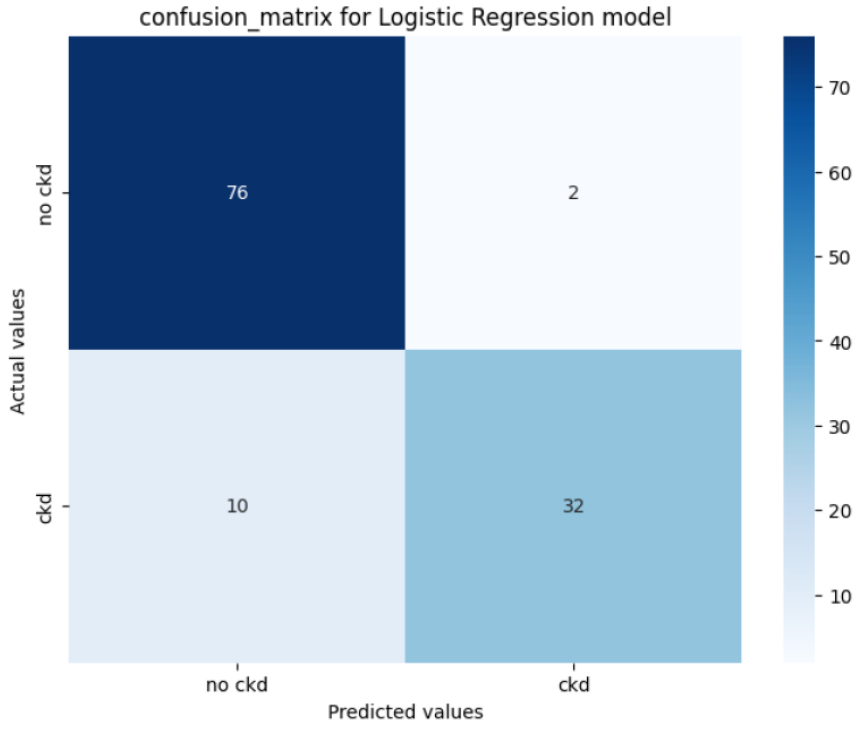
[60]:
```
from sklearn.neural_network import MLPClassifier
```

[61]:
```
#Milestone 5

from sklearn import model_selection


def classification_report(y_test, y_predict):
  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_predict,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)
  return final
  print(classification_report(y_test, y_predict))
```

[62]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_predict)
cm
```

[62]:
```
array([[76,  2],
       [10, 32]])
```

[63]:
```
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()
```



[ ]:
```
classification_report(y_test, y_predict)
```

[66]:
```
#Dumping our model in pickle form

pickle.dump(lgr,open('CKD.pkl','wb'))
```

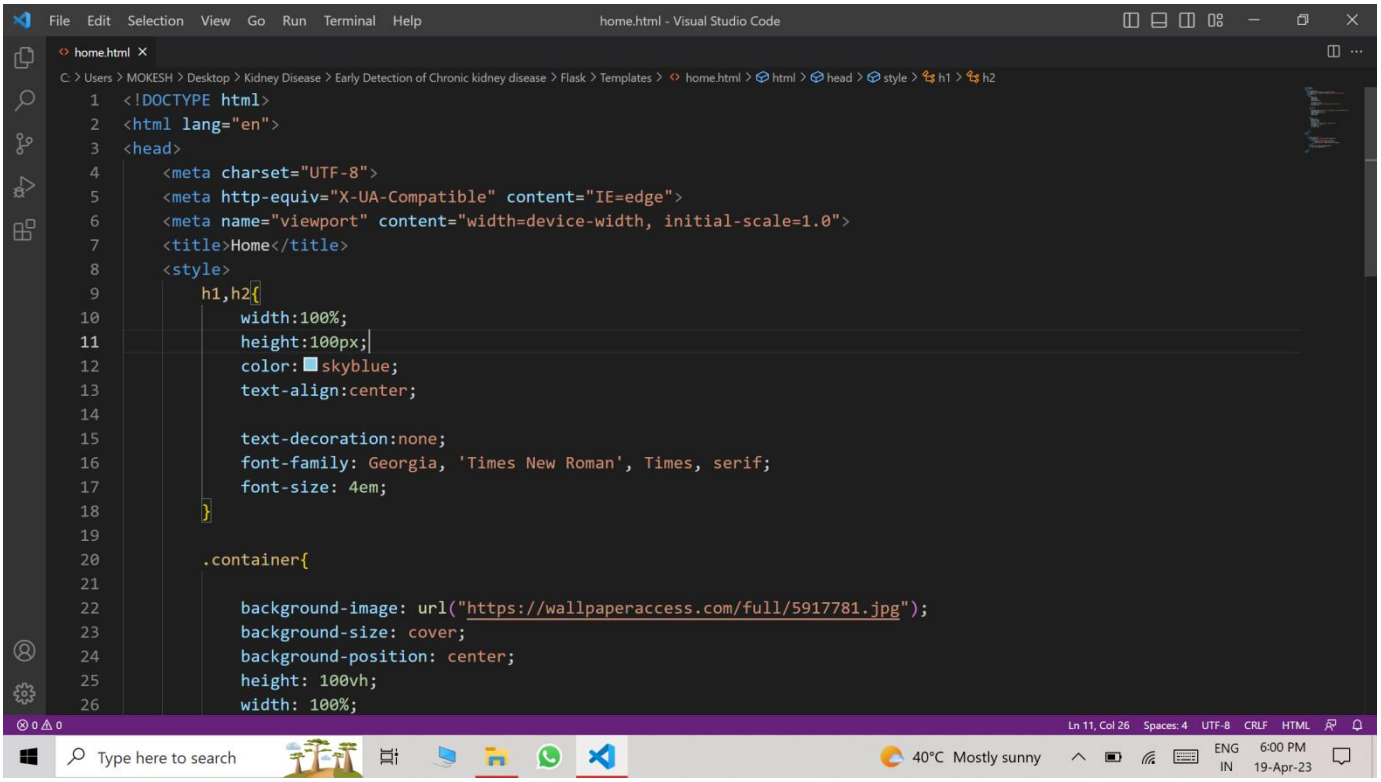
Milestone:6

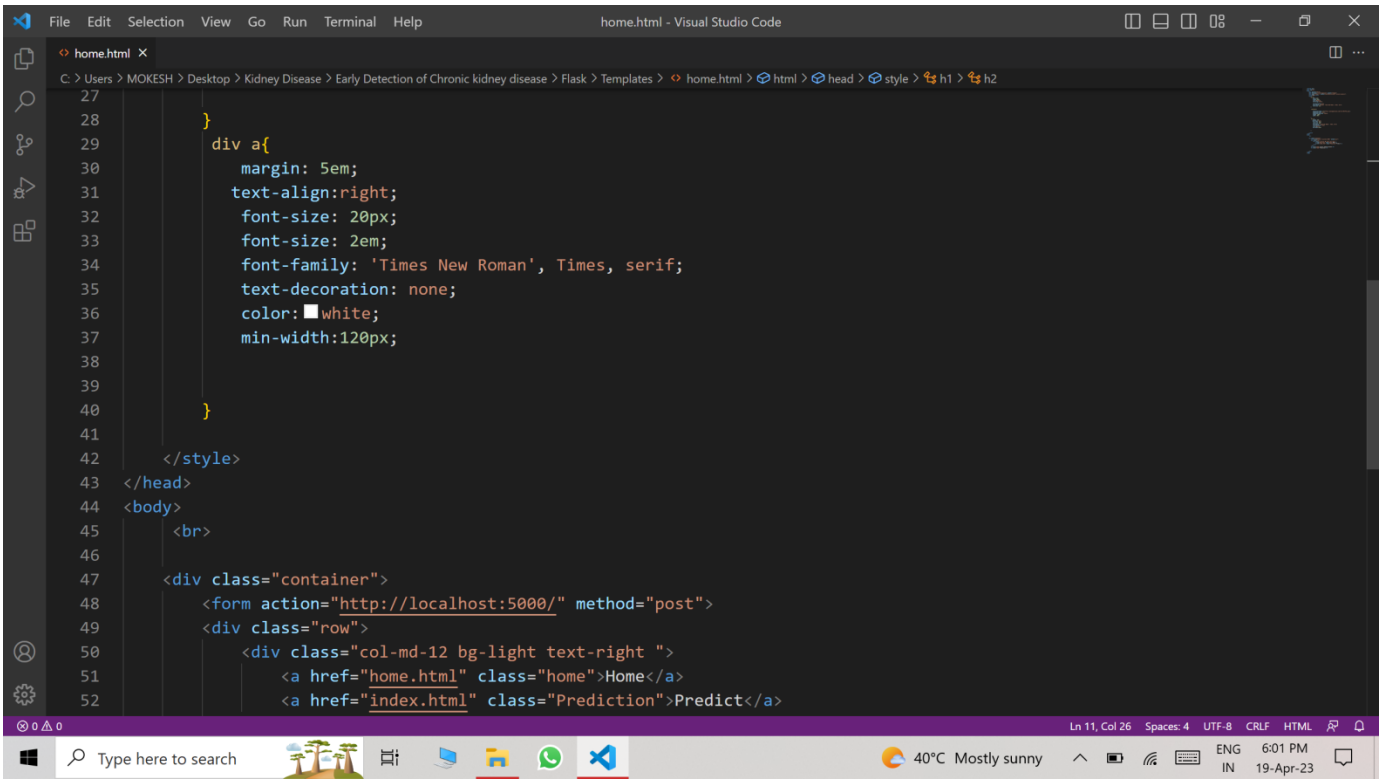
```
#importing the necessary dependencies

import numpy as np
import pandas as pd
from flask import Flask,request,render_template
import pickle
import os

app=Flask(__name__,template_folder='templates')
#iniatialize a Flask app

model=
pickle.load(open(os.path.join('C:/Users/MOKESH/Desktop/K
idney Disease/Early Detection of Chronic kidney
disease/Flask',
        'pkl_objects','CKD.pkl'),'rb'))

@app.route('/')#route to display home page
def home():
    return render_template('home.html')

@app.route('/Prediction',methods=['POST','GET'])
def prediction():
    return render_template("index.html")

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

```
    return render_template('home.html')

@app.route('/Predict',methods=['POST']) #route to show
prediction in web UI
def pred():

    #reading the inputs given by the user
    input_features =[float(x) for x in
request.form.values()]
    features_value =[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=features_name
)

    #prediction using the loaded model file
    output=model.predict(df)

    #showing the prediction results in UI
    return
render_template('result.html',prediction_text=output)

if __name__=='__main__' :
    #running the app
    app.run(debug=True)
```

Template

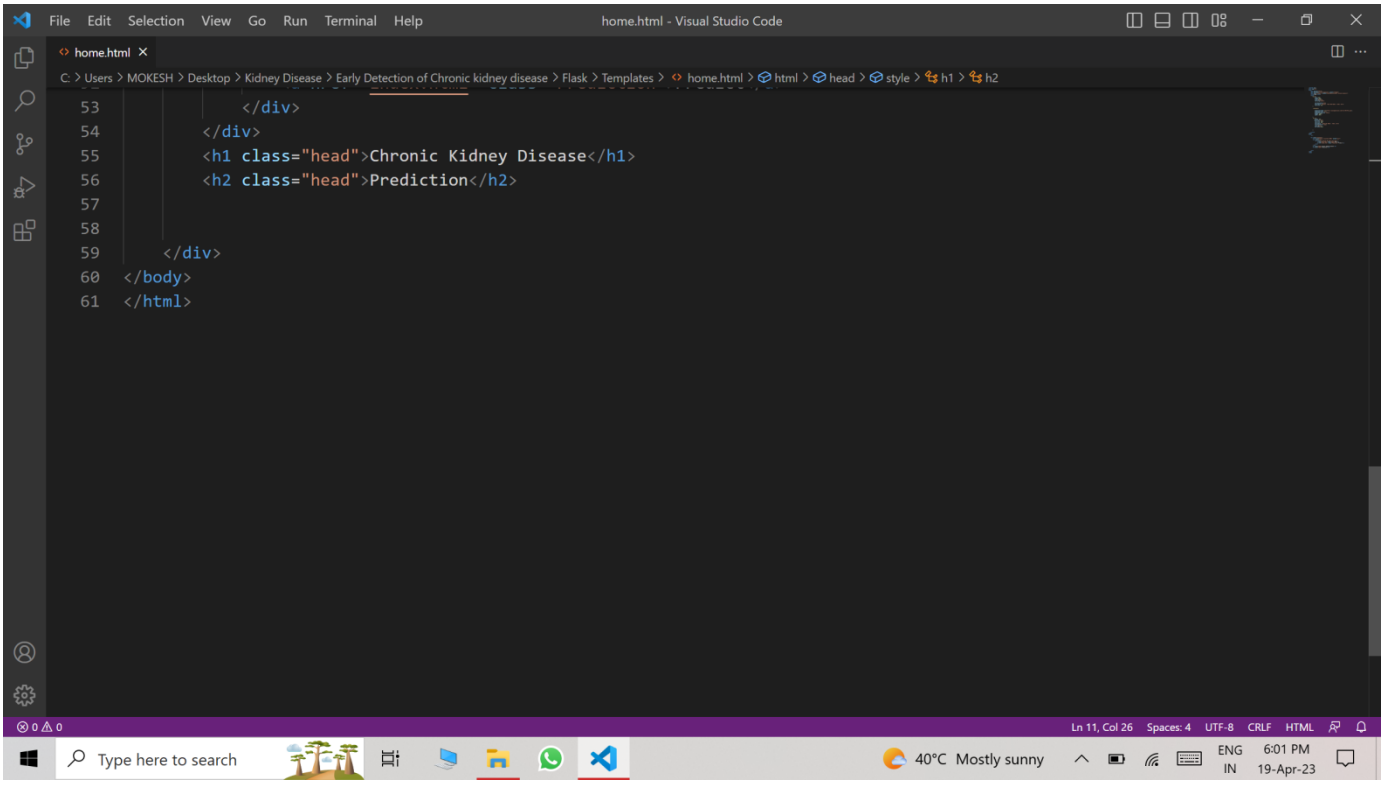
Home.html




File Edit Selection View Go Run Terminal Help
home.html - Visual Studio Code
home.html
C: > Users > MOKESH > Desktop > Kidney Disease > Early Detection of Chronic kidney disease > Flask > Templates > home.html > html > head > style > h1 > h2
}
div a{
margin: 5em;
text-align:right;
font-size: 20px;
font-size: 2em;
font-family: 'Times New Roman', Times, serif;
text-decoration: none;
color: white;
min-width:120px;
}
</style>
</head>
<body>

<form action="http://localhost:5000/" method="post">

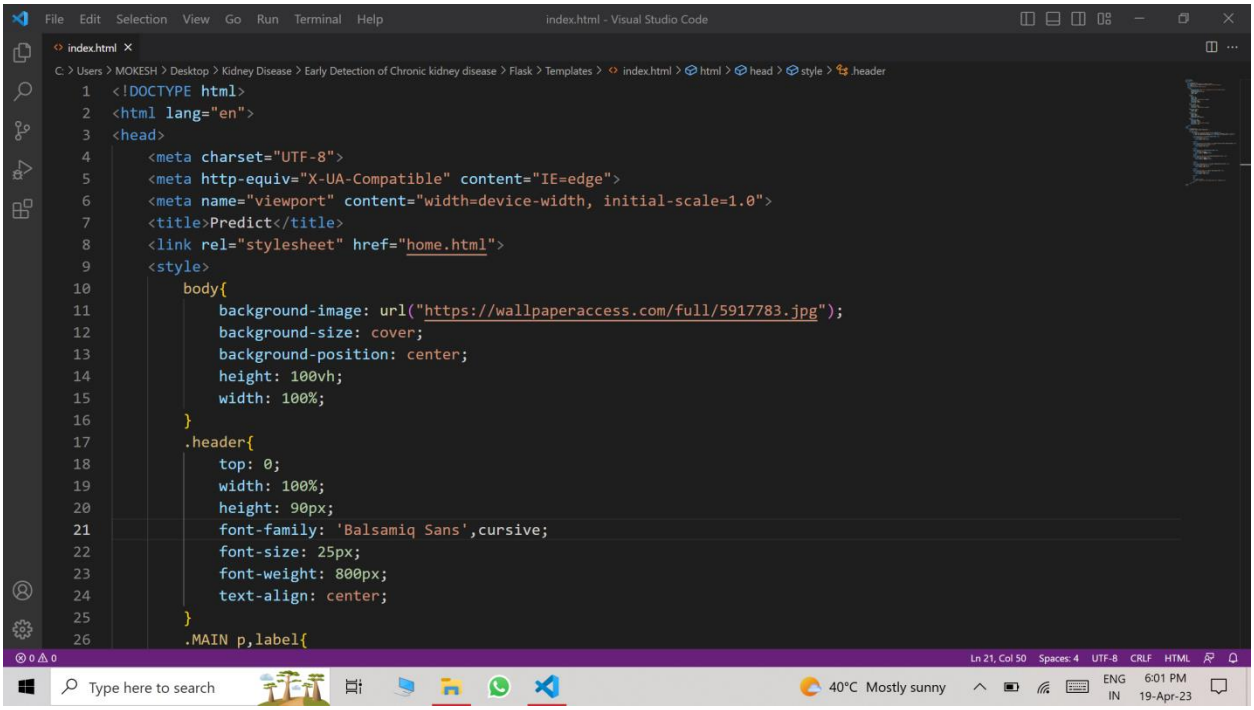
Home
Predict
Ln 11, Col 26 Spaces: 4 UTF-8 CRLF HTML
Type here to search
40°C Mostly sunny
ENG IN
6:01 PM 19-Apr-23


File Edit Selection View Go Run Terminal Help
home.html - Visual Studio Code
home.html
C: > Users > MOKESH > Desktop > Kidney Disease > Early Detection of Chronic kidney disease > Flask > Templates > home.html > html > head > style > h1 > h2

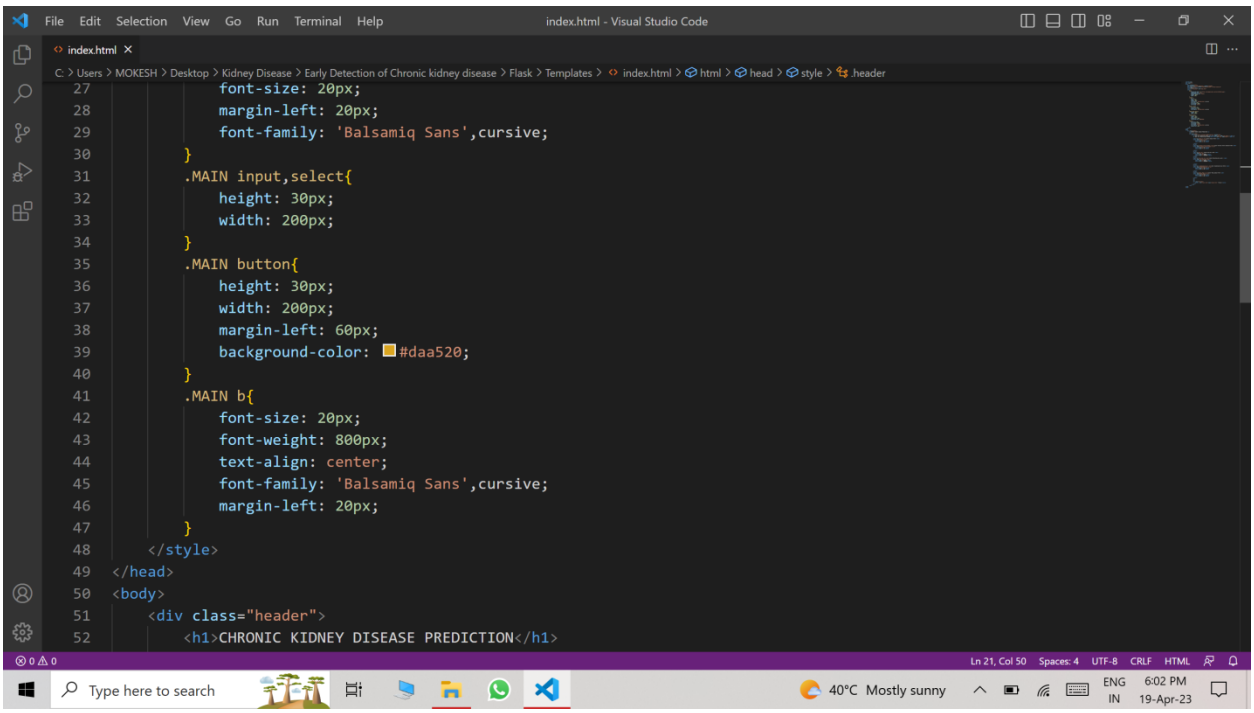
<h1 class="head">Chronic Kidney Disease</h1>
<h2 class="head">Prediction</h2>

</body>
</html>
Ln 11, Col 26 Spaces: 4 UTF-8 CRLF HTML
Type here to search
40°C Mostly sunny
ENG IN
6:01 PM 19-Apr-23

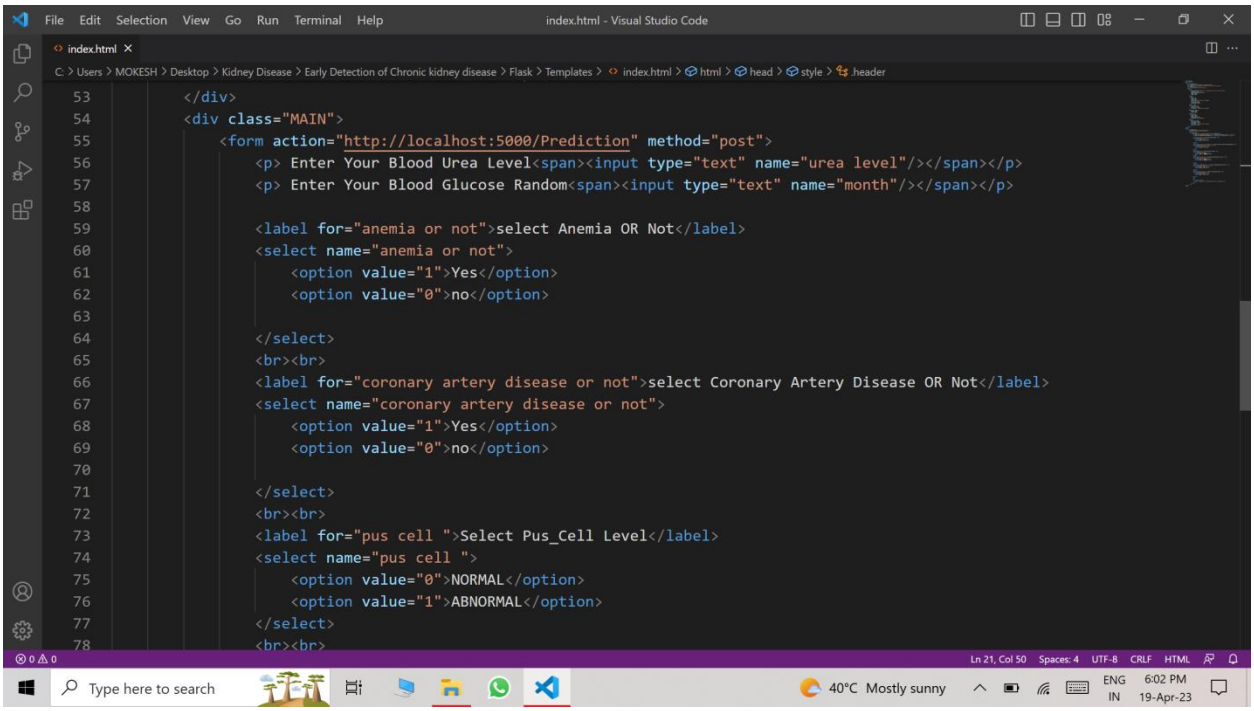
Index.html



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Predict</title>
    <link rel="stylesheet" href="home.html">
    <style>
        body{
            background-image: url("https://wallpaperaccess.com/full/5917783.jpg");
            background-size: cover;
            background-position: center;
            height: 100vh;
            width: 100%;
        }
        .header{
            top: 0;
            width: 100%;
            height: 90px;
            font-family: 'Balsamiq Sans',cursive;
            font-size: 25px;
            font-weight: 800px;
            text-align: center;
        }
        .MAIN p,label{
```



```
            font-size: 20px;
            margin-left: 20px;
            font-family: 'Balsamiq Sans',cursive;
        }
        .MAIN input,select{
            height: 30px;
            width: 200px;
        }
        .MAIN button{
            height: 30px;
            width: 200px;
            margin-left: 60px;
            background-color: #daa520;
        }
        .MAIN b{
            font-size: 20px;
            font-weight: 800px;
            text-align: center;
            font-family: 'Balsamiq Sans',cursive;
            margin-left: 20px;
        }
    </style>
</head>
<body>
    <div class="header">
        <h1>CHRONIC KIDNEY DISEASE PREDICTION</h1>
```

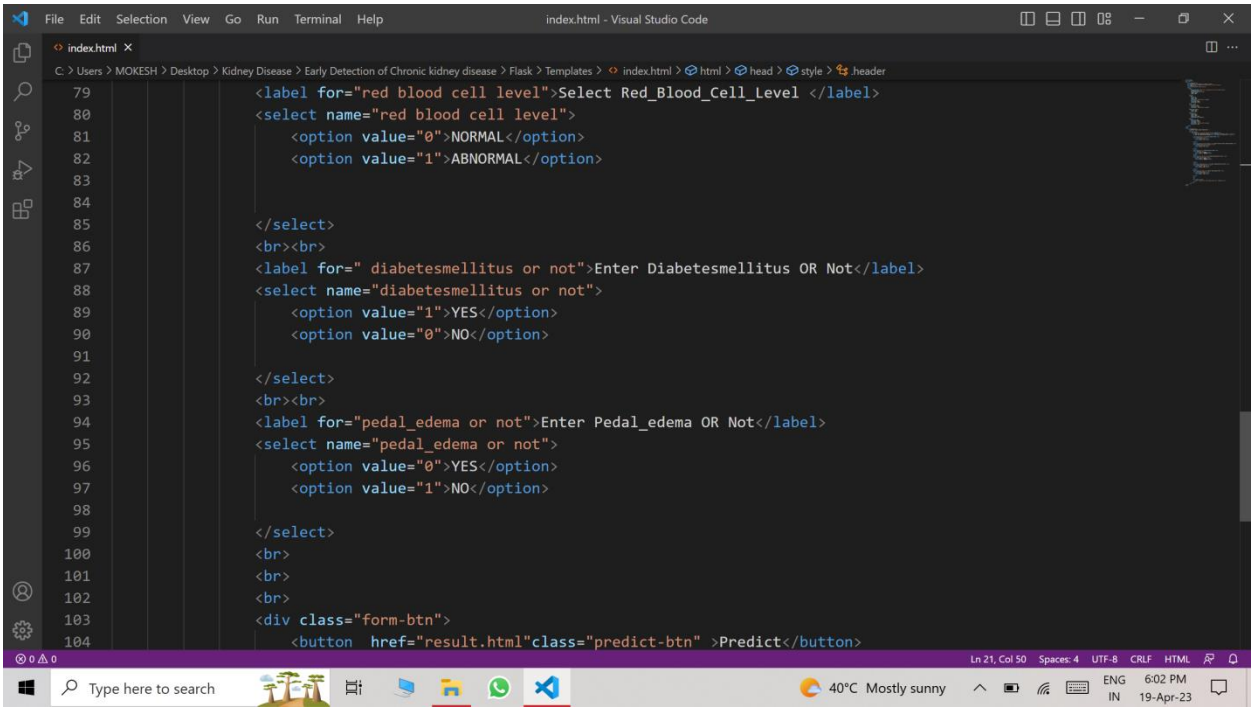

File Edit Selection View Go Run Terminal Help
index.html - Visual Studio Code
index.html
C: > Users > MOKESH > Desktop > Kidney Disease > Early Detection of Chronic kidney disease > Flask > Templates > index.html > html > head > style > .header

<form action="http://localhost:5000/Prediction" method="post">
 Enter Your Blood Urea Level<input type="text" name="urea level"/>
 Enter Your Blood Glucose Random<input type="text" name="month"/>
<label for="anemia or not">select Anemia OR Not</label>
<select name="anemia or not">
<option value="1">Yes</option>
<option value="0">no</option>
</select>

<label for="coronary artery disease or not">select Coronary Artery Disease OR Not</label>
<select name="coronary artery disease or not">
<option value="1">Yes</option>
<option value="0">no</option>
</select>

<label for="pus cell ">Select Pus_Cell Level</label>
<select name="pus cell ">
<option value="0">NORMAL</option>
<option value="1">ABNORMAL</option>
</select>

Ln 21, Col 50 Spaces: 4 UTF-8 CRLF HTML
Type here to search
40°C Mostly sunny
ENG IN
6:02 PM 19-Apr-23


File Edit Selection View Go Run Terminal Help
index.html - Visual Studio Code
index.html
C: > Users > MOKESH > Desktop > Kidney Disease > Early Detection of Chronic kidney disease > Flask > Templates > index.html > html > head > style > .header
<label for="red blood cell level">Select Red_Blood_Cell_Level </label>
<select name="red blood cell level">
<option value="0">NORMAL</option>
<option value="1">ABNORMAL</option>
</select>

<label for=" diabetesmellitus or not">Enter Diabetesmellitus OR Not</label>
<select name="diabetesmellitus or not">
<option value="1">YES</option>
<option value="0">NO</option>
</select>

<label for="pedal_edema or not">Enter Pedal_edema OR Not</label>
<select name="pedal_edema or not">
<option value="0">YES</option>
<option value="1">NO</option>
</select>

<button href="result.html"class="predict-btn" >Predict</button>
Ln 21, Col 50 Spaces: 4 UTF-8 CRLF HTML
Type here to search
40°C Mostly sunny
ENG IN
6:02 PM 19-Apr-23