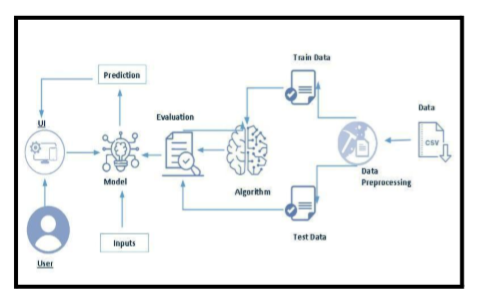
Project Title

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

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.

Technical Architecture:



Project Flow: ● User interacts with the UI to enter the input. ● Entered input is analysed by the model which is integrated. ● Once model analyses the input the prediction is showcased on the UI

To accomplish this, we have to complete all the activities listed below,

● Define Problem / Problem Understanding ○ Specify the business problem ○ Business requirements ○ Literature Survey ○ Social or Business Impact. ● Data Collection & Preparation ○ Collect the dataset ○ Data Preparation ● Exploratory Data Analysis ○ Descriptive statistical ○ Visual Analysis ● Model Building ○ Training the model in multiple algorithms ○ Testing the model ● Performance Testing & Evaluate the results ○ Testing model with multiple evaluation metrics ○ Evaluate the results ● Model Deployment ○ Save the best model ○ Integrate with Web Framework ● Project Demonstration & Documentation ○ Record explanation Video for project end to end solution ○ Project Documentation-Step by step project development procedure

Project Structure:

Create the Project folder which contains files as shown below

We are building a flask application which needs HTML pages stored in the templates folder

and a python script app.py for scripting. ● CKD.pkl is our saved model. Further we will use this model for flask integration. ● Training folder contains a model training file.

Milestone 1: Define Problem / Problem Understanding Activity 1: Specify the business problem

Refer Project Description Activity 2: Business requirements

The business requirements for a machine learning model to predict chronic kidney disease include the ability to accurately predict the ckd based on given information, Minimise the number of false positives (predicting diseased) and false negatives (not diseased). Provide an explanation for the model's decision, to comply with regulations and improve transparency. Activity 3: Literature Survey (Student Will Write)

Chronic kidney disease (CKD) is a significant public health issue, affecting an estimated 14% of the global population. The disease is characterized by a gradual loss of kidney function over time, leading to a range of serious health complications, including end-stage renal disease (ESRD) requiring dialysis or kidney transplant. Early detection and management of CKD is crucial to prevent progression to ESRD and improve patient outcomes.

There have been numerous studies in recent years aimed at developing accurate and efficient methods for predicting CKD progression. These studies have employed a variety of techniques, including machine learning, deep learning, and artificial neural networks. Activity 4: Social or Business Impact.

On a social level, early detection and prediction of CKD can lead to improved patient outcomes and quality of life. By identifying individuals at risk for CKD, healthcare providers can intervene early and slow the progression of the disease through lifestyle changes, medication management, and other treatments. This can help prevent the need for dialysis or kidney transplantation, which can be costly and life-altering for patients. Additionally, early prediction can also help reduce the overall burden of CKD on the healthcare system by reducing the number of hospitalizations and emergency room visits.

Milestone 2: Data Collection & Preparation

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So this section allows you to download the required dataset.

Activity 1: Collect the dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link: <https://www.kaggle.com/datasets/mansoordaku/ckdisease>

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

Note: There are several techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Activity 1.1: Importing the libraries

Import the necessary libraries as shown in the image. (optional) Here we have used visualisation style as fivethirtyeight.

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 LableEncoder

from sklearn.linear\_model import logisticRegression

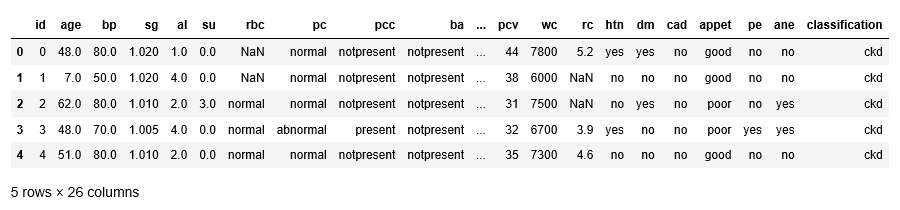
import pickle

Activity 1.2: Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas. In pandas we have a function called read\_csv() to read the dataset. As a parameter we have to give the directory of the csv file.

data=pd.read\_csv("D:\\NMGTR\kidney\_disease.csv")

data.head()



Activity 2: Data Preparation

As we have understood how the data is, let's pre-process the collected data. The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

● Rename the columns

● Handling missing values

● Handling categorical data

● Handling Numerical data

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

Activity 2.1: Rename the columns

data.columns

data.columns=['age','blood\_pressure','specfic\_gravity','albumin',

'suger','red\_blood\_cells','pus\_cell','pus\_cell\_clumps','bacteria',

'blood glucous random','blood\_urea','serum\_ creatinine','sodium','potassium',

'hemoglobin','packed\_cell\_valume','white\_blood\_cell\_count','red\_blood\_cell\_count',

'hypertension','diabetesmellitus','coronary\_artery\_disease','appetite',

'pedal\_edema','anemia','class']

data.columns

Activity 2.2: Handling missing values

● Let’s find the shape of our dataset first. To find the shape of our data, the df.shape method is used. To find the data type, df.info() function is used.

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 bp 388 non-null float64

3 sg 353 non-null float64

4 al 354 non-null float64

5 su 351 non-null float64

6 rbc 248 non-null object

7 pc 335 non-null object

8 pcc 396 non-null object

9 ba 396 non-null object

10 bgr 356 non-null float64

11 bu 381 non-null float64

12 sc 383 non-null float64

13 sod 313 non-null float64

14 pot 312 non-null float64

15 hemo 348 non-null float64

16 pcv 330 non-null object

17 wc 295 non-null object

18 rc 270 non-null object

19 htn 398 non-null object

20 dm 398 non-null object

21 cad 398 non-null object

22 appet 399 non-null object

23 pe 399 non-null object

24 ane 399 non-null object

25 classification 400 non-null object

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

memory usage: 81.4+ KB

data.isnull().any()

id False

age True

bp True

sg True

al True

su True

rbc True

pc True

pcc True

ba True

bgr True

bu True

sc True

sod True

pot True

hemo True

pcv True

wc True

rc True

htn True

dm True

cad True

appet True

pe True

ane True

classification False

dtype: bool

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['hypertention'].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['diabatesmellitus'].fillna(data['diabatesmellitus'].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)

Activity 2.3: Handling Categorical columns

The below code is used for fetching all the object or categorical type of columns from our data and we are storing it as set in variable catcols.

catcols=set(data.dtypes[data.dtypes=='0'].index.values)# only fetch the object type column

print(catcols)

set()

for i in catcols:

print("Columns:",i)

print(c(data[i])) # using counter for number of classes in the column

print('\*'\*120+'\n')

In the above we are looping with each categorical column and printing the classes of each categorical columns using counter function so that we can detect which columns are categorical and which are not.

If you observe some columns have a few classes and some have many, those columns are having many classes can be considered as numerical column and we have to remove it and add it to the continuous columns.

catcols.remove('red\_blood\_cells\_count') # remove is used for removing a particular column

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

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

print(catcols)

#'specific\_gravity','sugar'(as these columns are numerical it is removed)

catcols=['anemia','pedal\_edema','appetite','class','coronary\_artery\_disease','diabetesmellit'

'hypertension','pus\_cell','pus\_cell\_clumps','red\_blood\_cells']#only considered the text class columns

As we store our columns as set, we can make use of remove function which is used to remove the element in our case we can take it as columns.

Activity 2.3.1: Label Encoding for categorical columns

from sklearn.preprocessing import LabelEncoder

# importing the LableEncoding fropm sklearn

for i in catcols:

#looping through all the categorical columns

print("LABEL ENCODING OF:",i)

LEi=LabelEncoder()

#creting on object of LabelEncoder

print(c(data[i])

#getting the classes values before transformation

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

#transforming our text classes to numerical values

print(c(data[i]))

#getting the classes values after transformation

print("\*"\*100)

contcols=set(data.dtypes[data.dtypes!='0'].index.values)# only fetch the float and int type columns

#contcols=pd.DataFrame(data,column=contcols)

print(contcols)

{'pot', 'bu', 'pc', 'htn', 'appet', 'bp', 'classification', 'ane', 'rc', 'ba', 'pe', 'pcc', 'cad', 'sg', 'rbc', 'hemo', 'wc', 'su', 'pcv', 'sc', 'id', 'al', 'sod', 'age', 'dm', 'bgr'}

for i in contcols:

print("Continous Columns:",i)

print(c(data[i]))

print('\*'\*120+'\n')

Continous Columns: pot 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: bu 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: pc Counter({'normal': 259, 'abnormal': 76, nan: 65}) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Continous Columns: htn Counter({'no': 251, 'yes': 147, nan: 2}) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Continous Columns: appet Counter({'good': 317, 'poor': 82, nan: 1}) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Continous Columns: bp 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: classification Counter({'ckd': 248, 'notckd': 150, 'ckd\t': 2}) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Continous Columns: ane Counter({'no': 339, 'yes': 60, nan: 1}) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Continous Columns: rc 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}) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Continous Columns: ba Counter({'notpresent': 374, 'present': 22, nan: 4}) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Continous Columns: pe Counter({'no': 323, 'yes': 76, nan: 1}) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Continous Columns: pcc Counter({'notpresent': 354, 'present': 42, nan: 4}) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Continous Columns: cad Counter({'no': 362, 'yes': 34, '\tno': 2, nan: 2}) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Continous Columns: sg 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: rbc Counter({'normal': 201, nan: 152, 'abnormal': 47}) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Continous Columns: hemo 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: wc 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}) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

contcols.remove('specific\_gravity')

contcols.remove('albumin')

contcols.remove('sugar')

print(contcols)

catcols.add('specific\_gravity')

catcols.add('albumin')

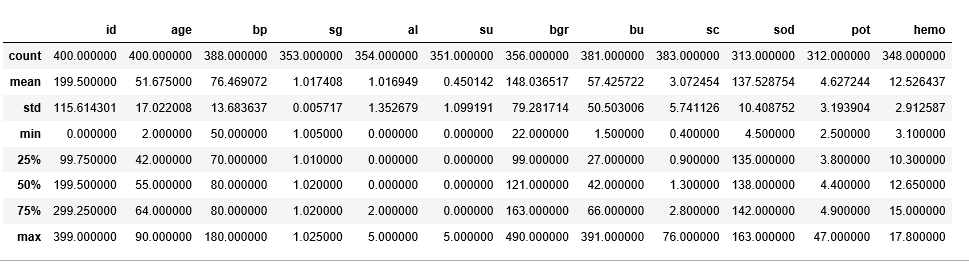
catcols.add('sugar')

print(catcols)

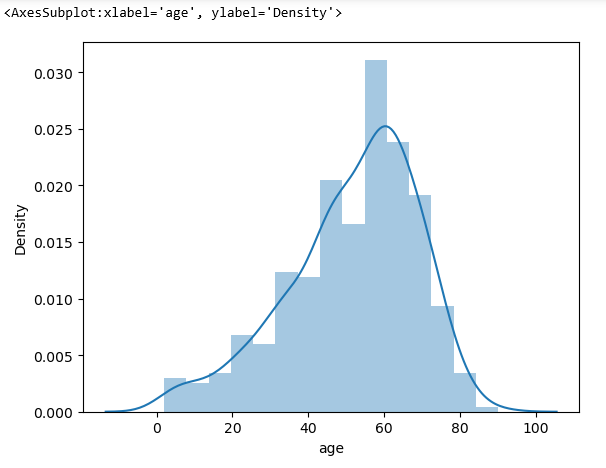
data['coronary\_artery\_disease']=data.coronary\_artery\_disease.replace('\tno','no')# replacing \tno wi

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

data.describe()



sns.distplot(data.age)



import matplotlib.pyplot as plt

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

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

plt.xlabel('age')

plt.ylabel('blood\_pressure')

polt.title("age VS blood Scatter Plot")

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

plotnumber=1

for column in contcols:

if plotnumber<=11:

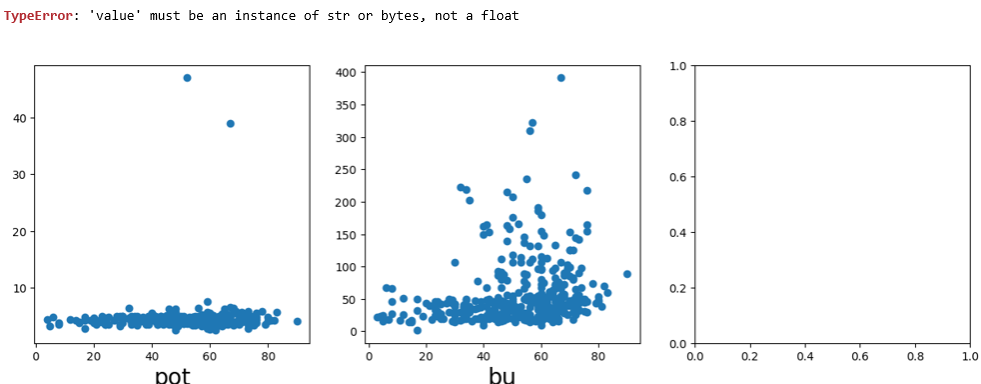
ax=plt.subplot(3,4,plotnumber)

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

plt.xlabel(column,fontsize=20)

plotnumber+=1

plt.show()



f,sx=plt.sublots(figsize=(18,10))

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

plt.xticks(rotation=45)

plt.yticks(rotation=45)

plt.show()

sns.countplot(data['class'])

from sklearn.preprocessing import Standerdscaler

sc=StanederdScaler(

x\_bal=sc.fit\_transformx())

selcols=['red\_blood\_cells','pus\_cells','blood glucose random','blood\_urea',

'pedal\_edema','anemia','diabetesmellitus','coranaory\_artery\_disease']

x=pd.DataFrame(data,coloumns=selcous)

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

print(x.shape)

print(y.shape

import tensorflow

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import dense

classification=Sequential()

classification.add(Denes(30.activation='relu'))

classification.add(Denes(128.activation='relu'))

classification.add(Denes(64.activation='relu'))

classification.add(Denes(32.activation='relu'))

classification.add(Denes(1.activation='relu'))

calsification.combile(optimizer='admin',loss='binary\_crossentqry',metrics=['accuracy'])

classification.fit(x\_train,y\_train.batch\_size=10,validation\_split=0.2,epochs=100)

from sklearn.ensemble import RandamForestClassifier

rfc=RandamForest(n\_estimators=10,criterion='entroy')

rfc.fit(x\_train)

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

from sklearn.linear\_model import LogicticsRegestration

lgr=LogisticRegestration()

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

LogiesticRegistration()

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

y\_predict=lgr.predict(x\_test)

classification.save("{ckd.hs}")

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

y\_pred=(y\_pred>0.5)

y\_pred

def predict\_exit(sample\_value):

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

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

sample\_value=sc.tranceform(sample\_values)

test=classification.preduct([[1,1,121,000000,36.0,0,0,1,0,]])

if test==1;

print('prediction:high change of CKD!')

else;

print('prediction:low change of SKD.')

from sklearn import modle\_selection

from sklearn.matrics import confusion\_matrix

cm

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

sns.heatmap(cm.cmp='blues',annot=true,xticklabels=['no ckd,'ckd],yticklabls=['no ckd','ckd'])

plt.xlabl;e('predicted values')