# INTRODUCTION

# OVERVIEW:

# ABSTRACT

Chronic kidney disease (CKD) is a major burden on the healthcare system because of its increasing prevalence, high risk of progression to end-stage renal disease, and poor morbidity and mortality prognosis. It is rapidly becoming a global health crisis. Unhealthy dietary habits and insufficient water consumption are significant contributors to this disease. Without kidneys, a person can only live for 18 days on average, requiring kidney transplantation and dialysis. It is critical to have reliable techniques at predicting CKD in its early stages. Machine learning (ML) techniques are excellent in predicting CKD. The current study provides a method for predicting the presence of CKD using clinical data that includes information to identify CKD.

**PURPOSE:**

**Specify the Business problem:**

* The CKD affects 5 to 10 percent of the population worldwide.
* Most cases of Chronic Kidney Disease go undiagnosed or are later diagnosed in underdeveloped and developing nations.
* \* This is one of the primary reasons why a higher percentage of such Case come from developing and underdeveloped nations as opposed to developed nations where most people go through regular check-ups and diagnose.
* \* So we need Machine –Based Learning Systems used to diagnose Chronic Kidney Disease .

**Business Requirements:**

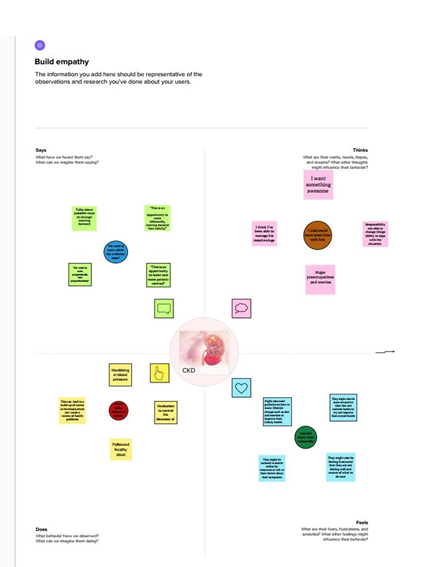
**Machine Based –Learning:**

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.

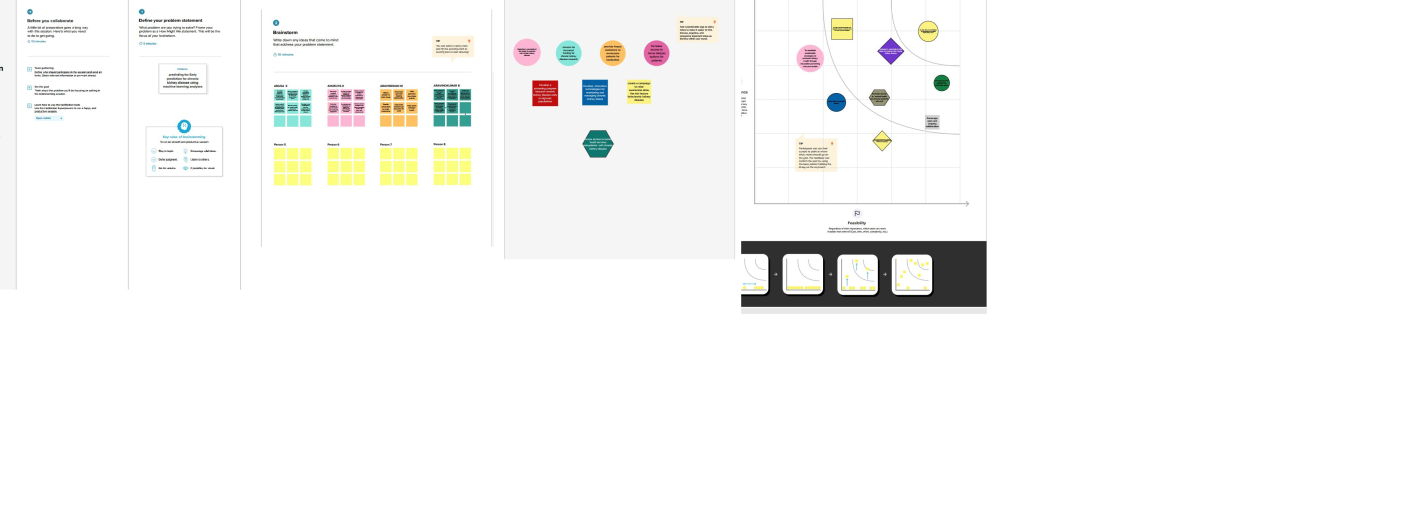
The only way to find out for sure if you have CKD is through specific blood and urine tests. These tests include measurement of both the creating level in the blood and protein in the urine. Kidney diseases are a leading cause of death in the United States. Early CKD has no signs or symptoms

**PROBLEM DEFINITION & DESIGN THINKING:**

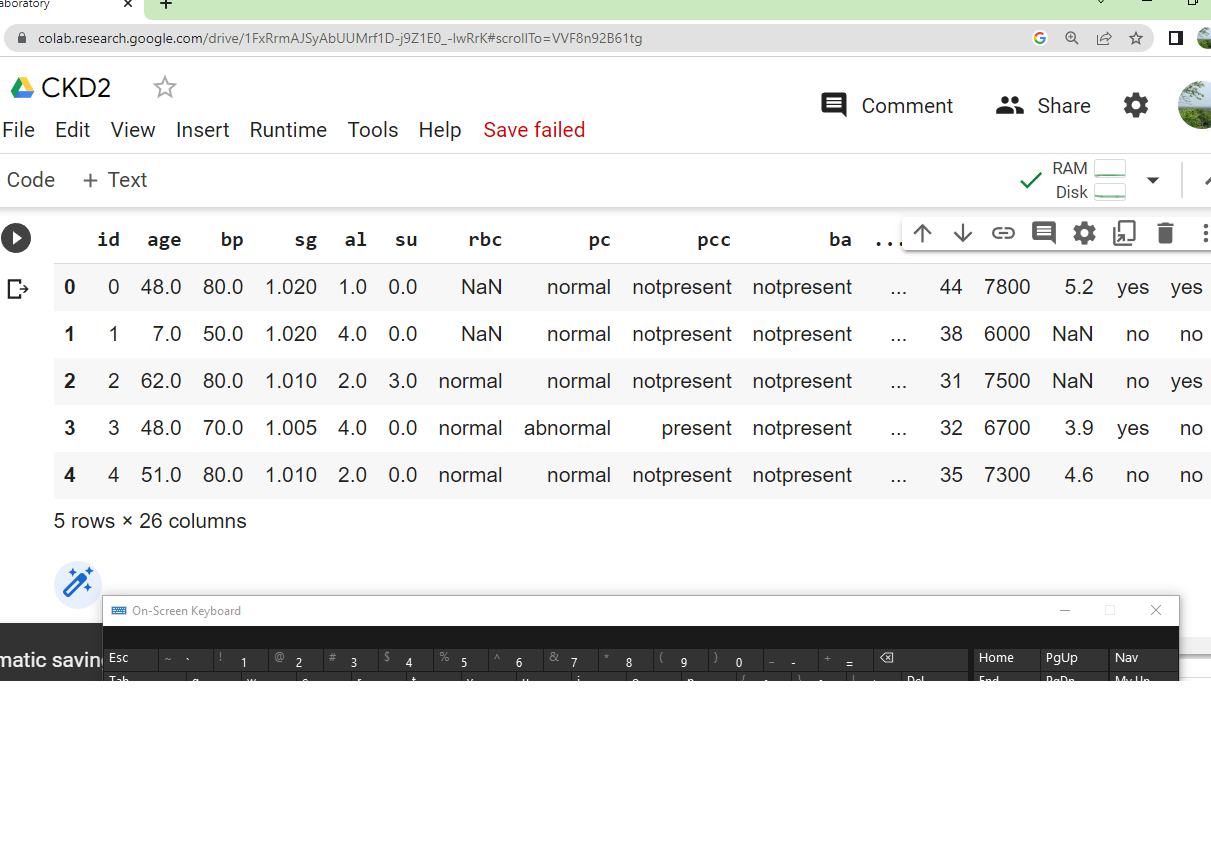
**EMPATHY MAP:**

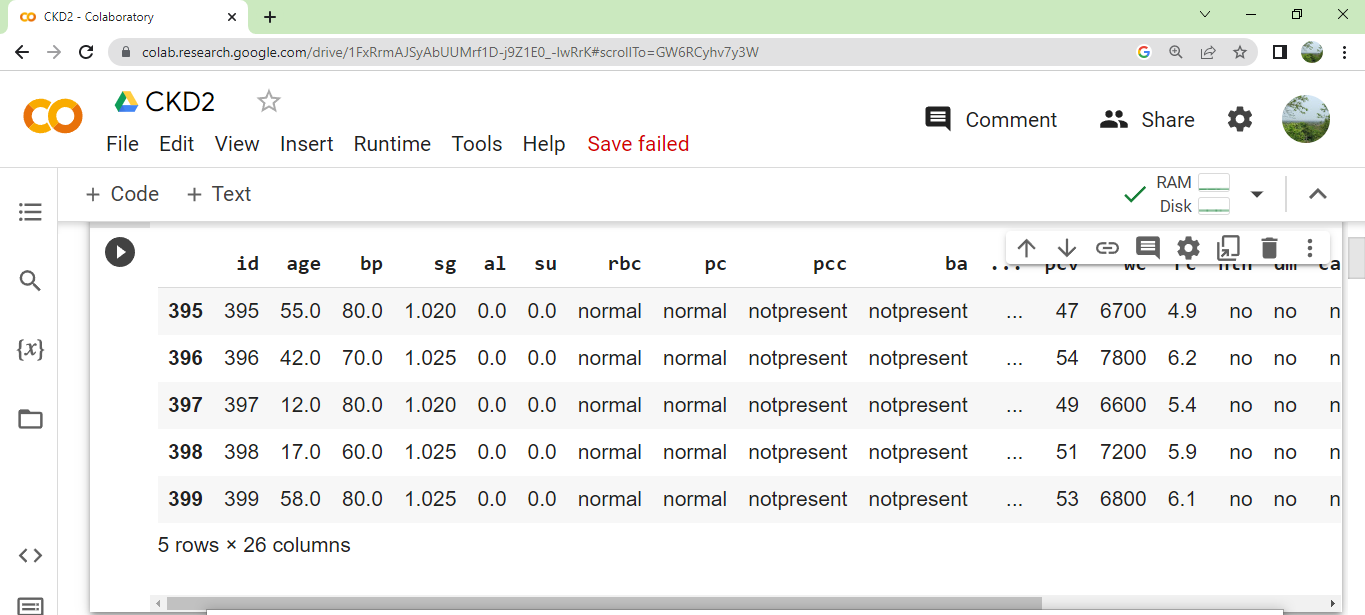
****

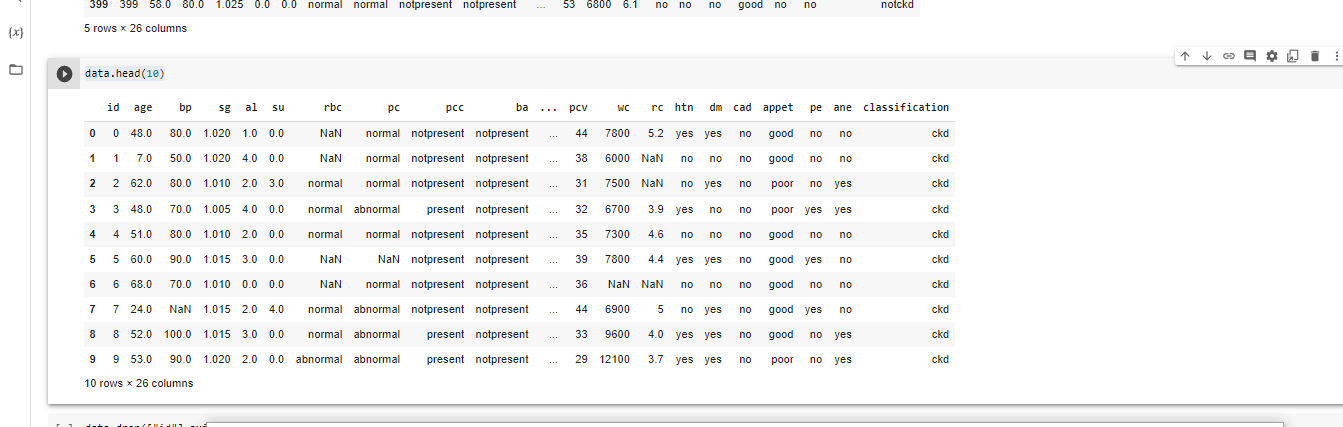
**BRAINSTORM:**

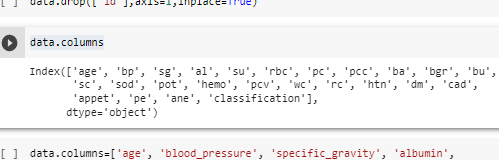
****

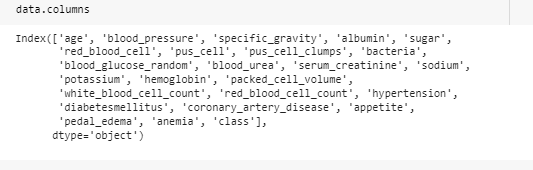
**RESULTS:**

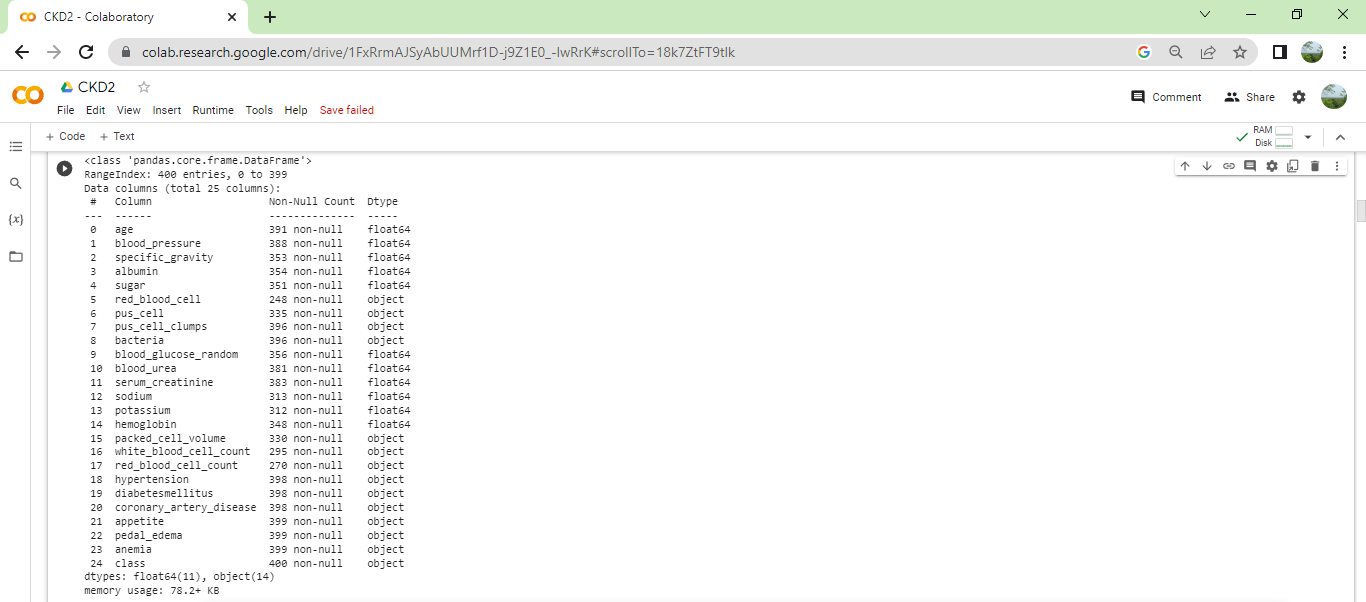




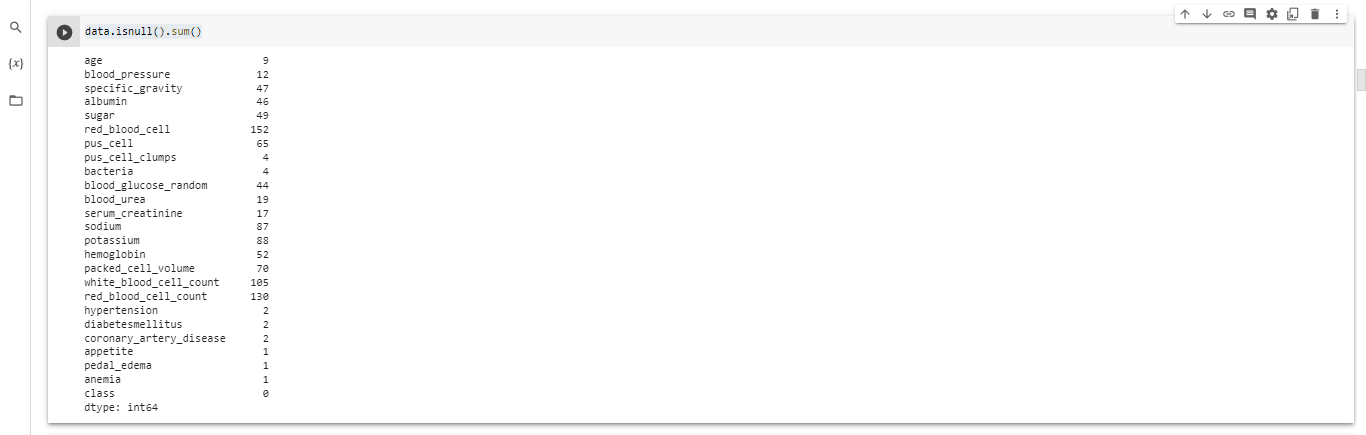


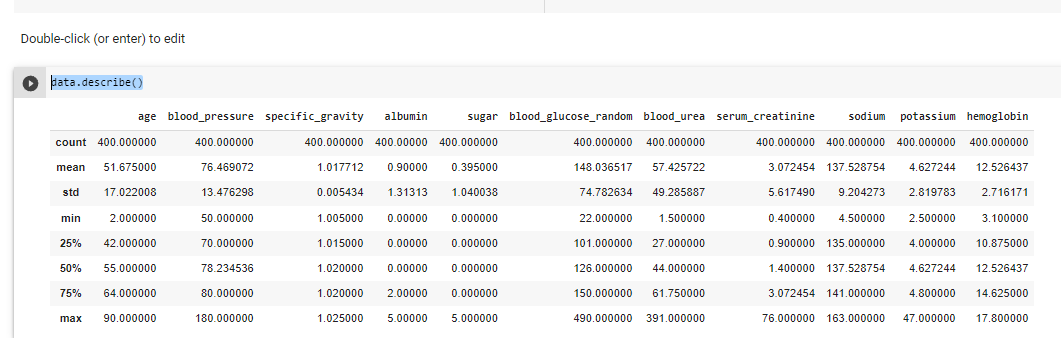


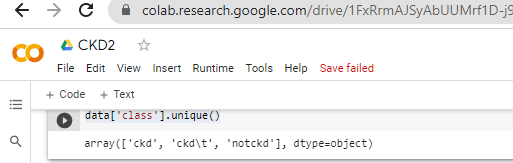


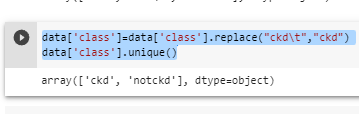


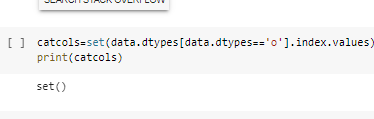


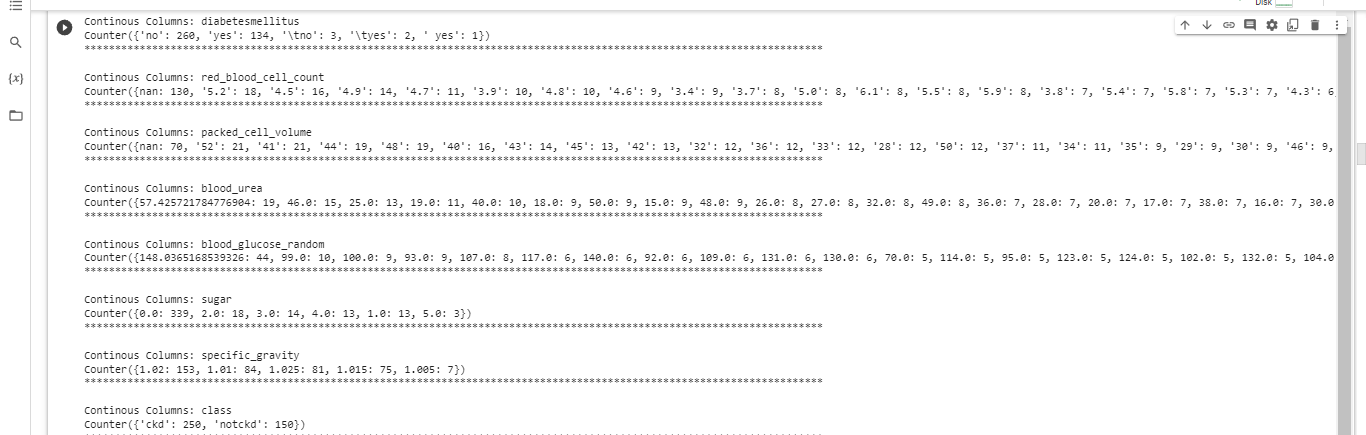


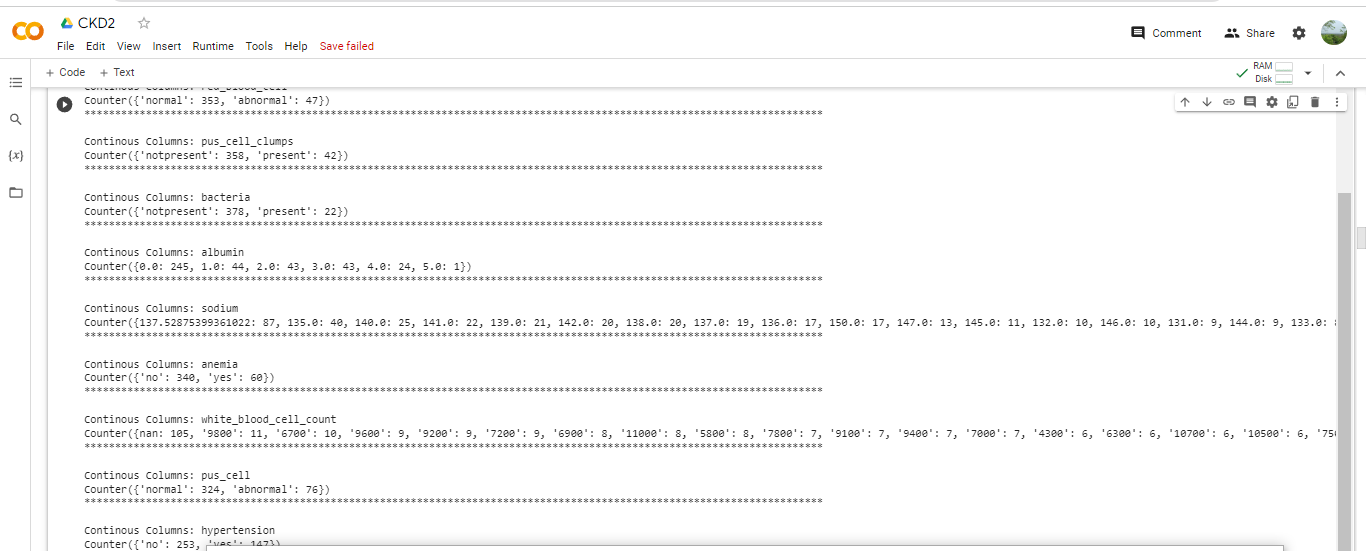


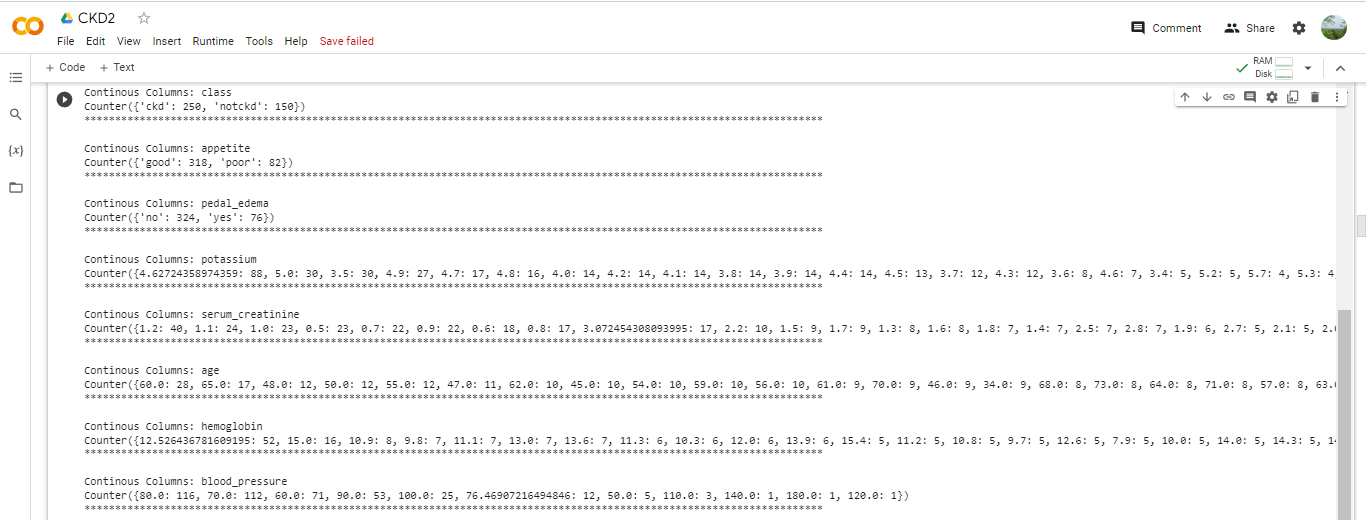




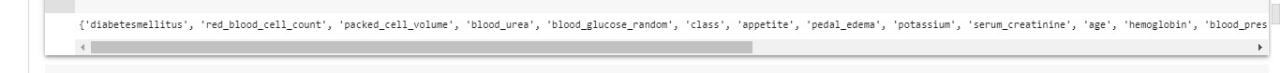






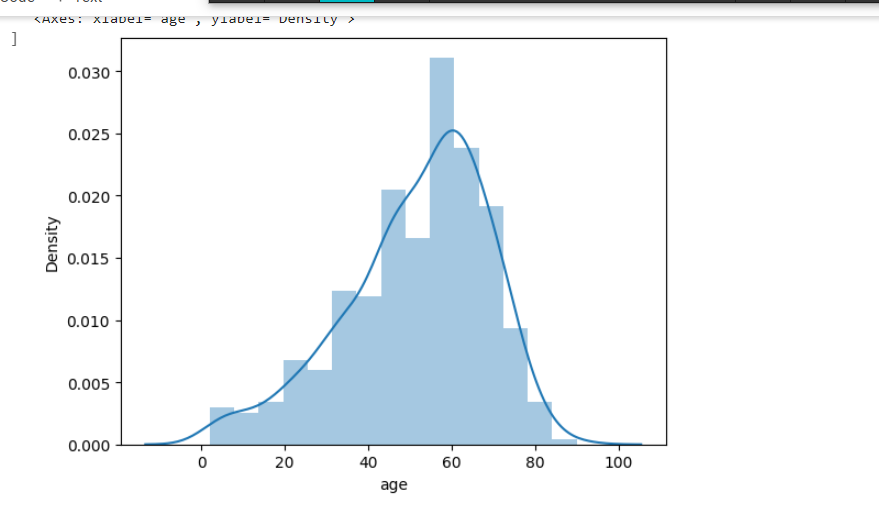


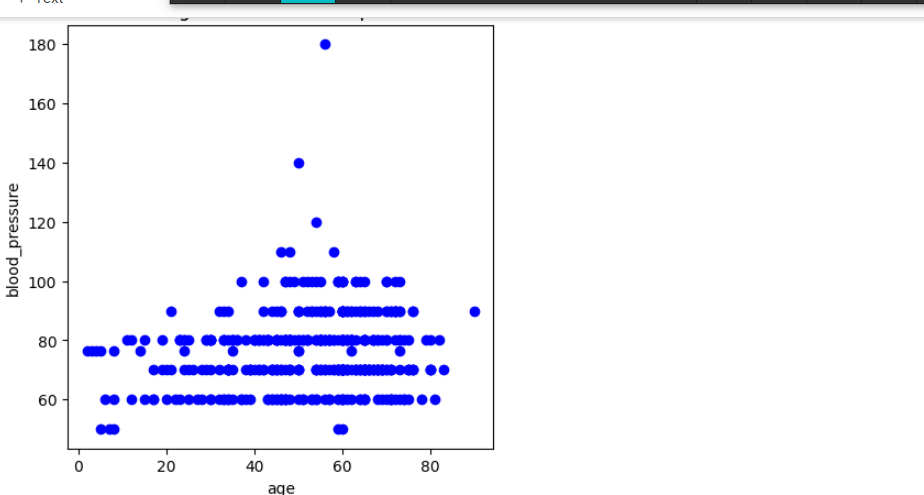


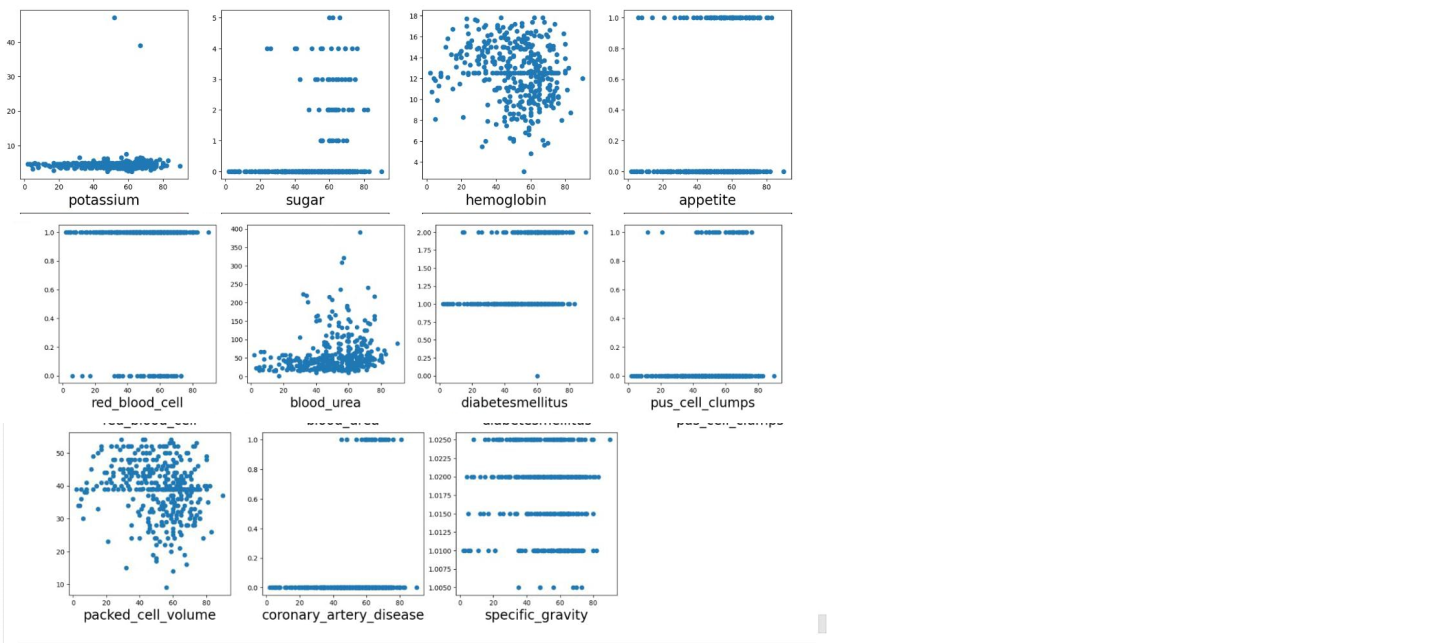


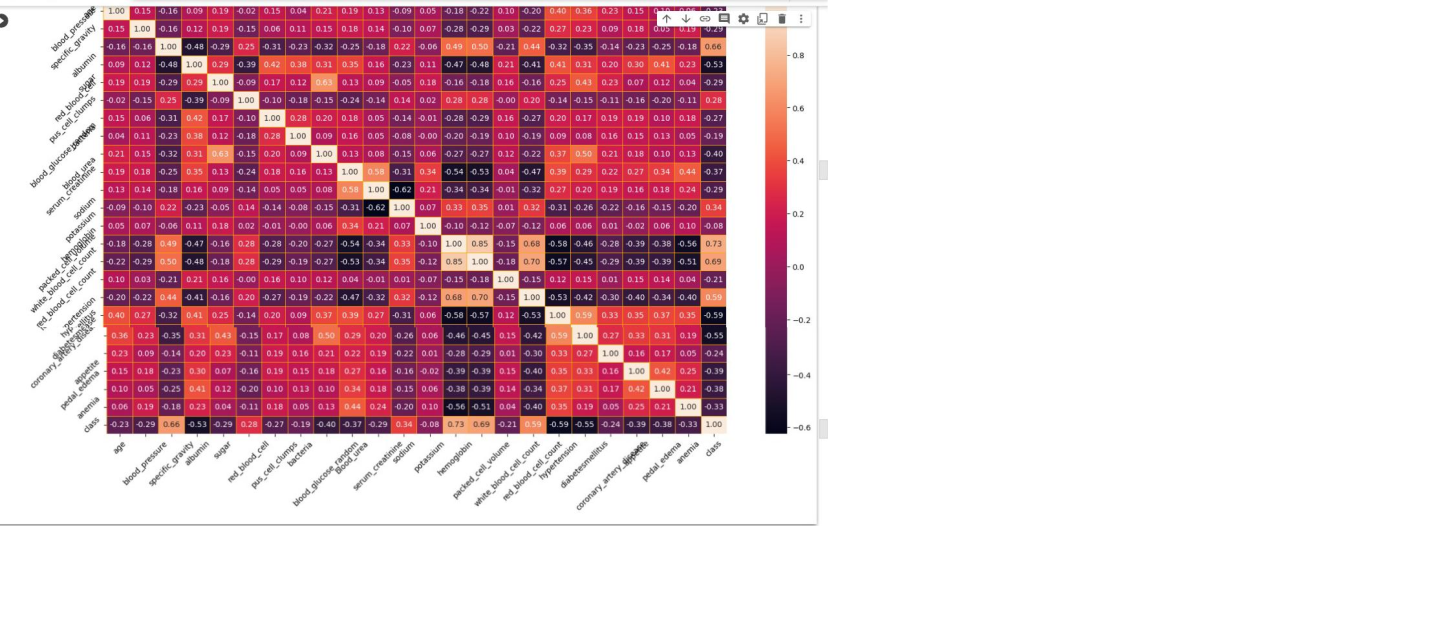


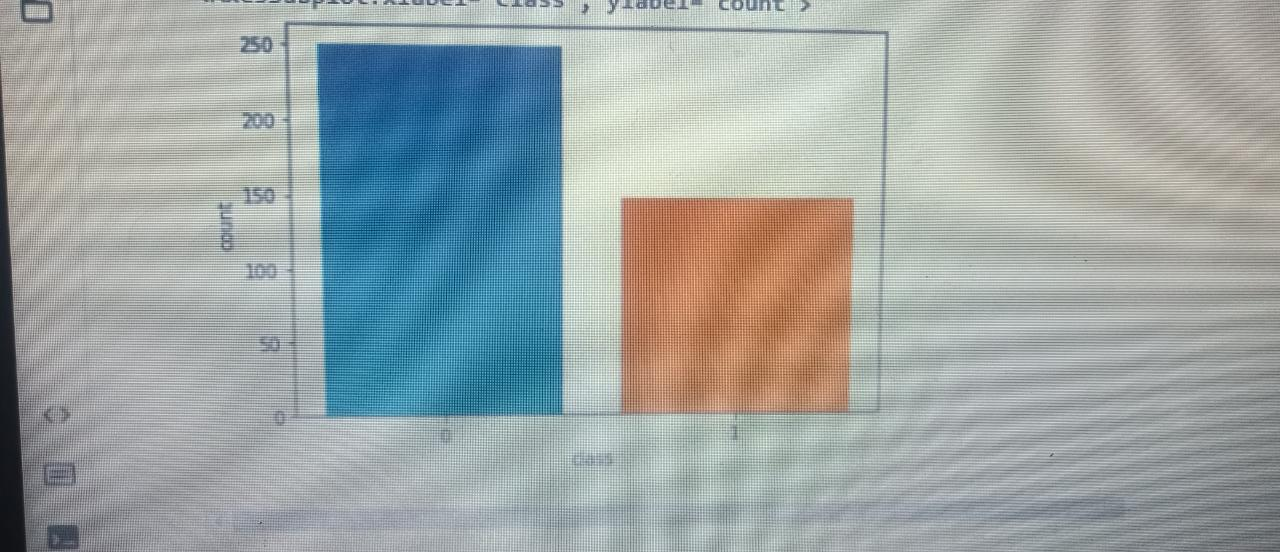




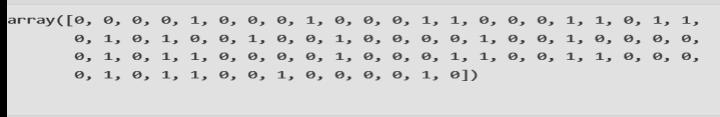


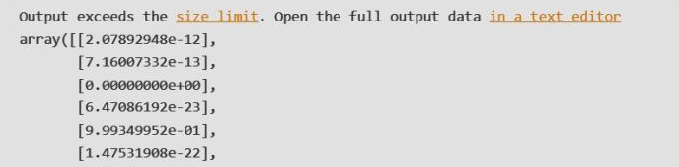


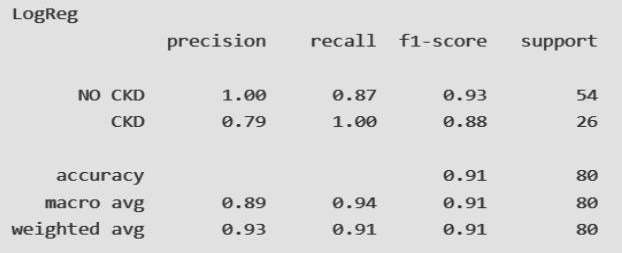


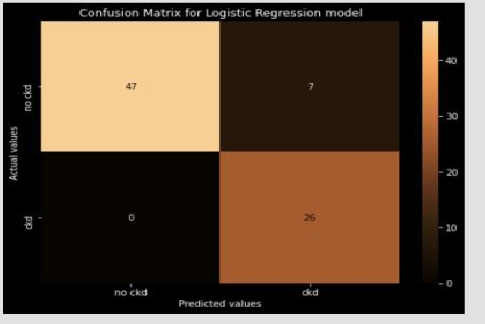


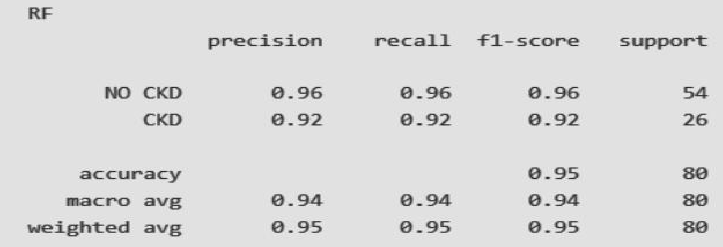


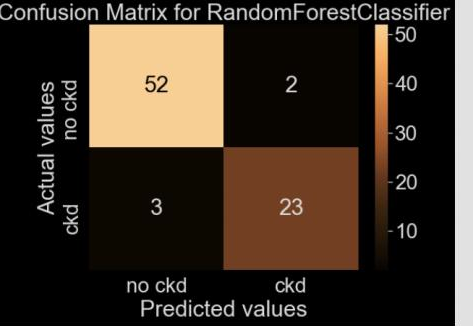


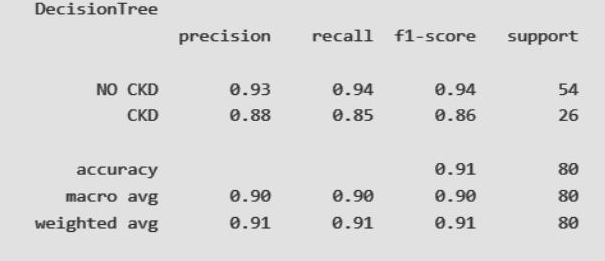


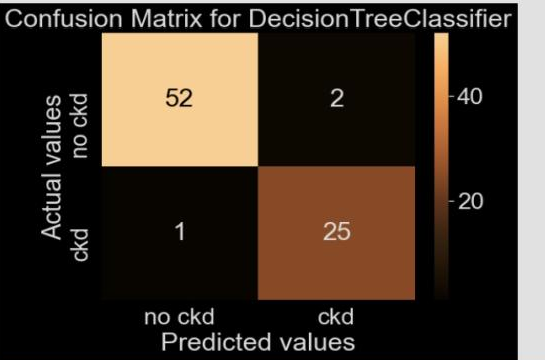


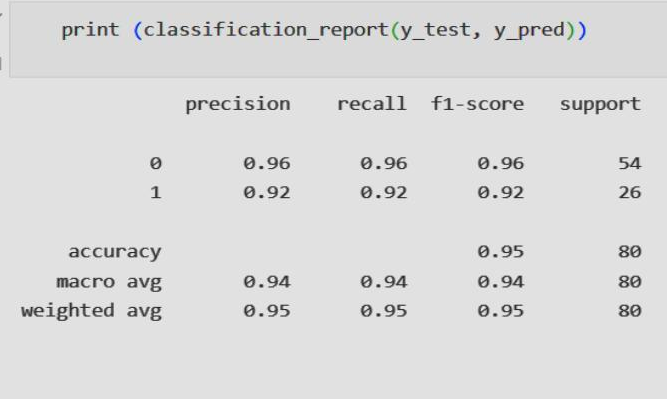


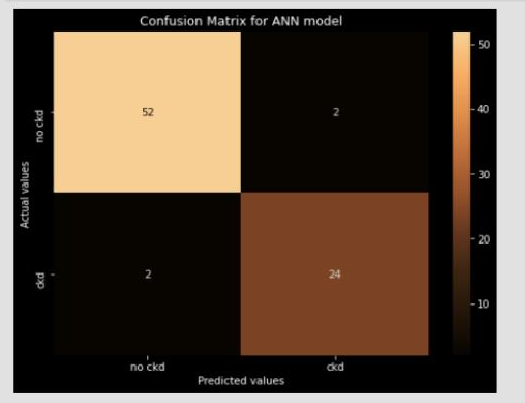


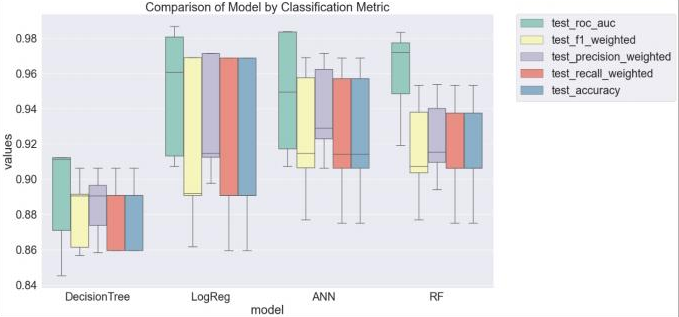


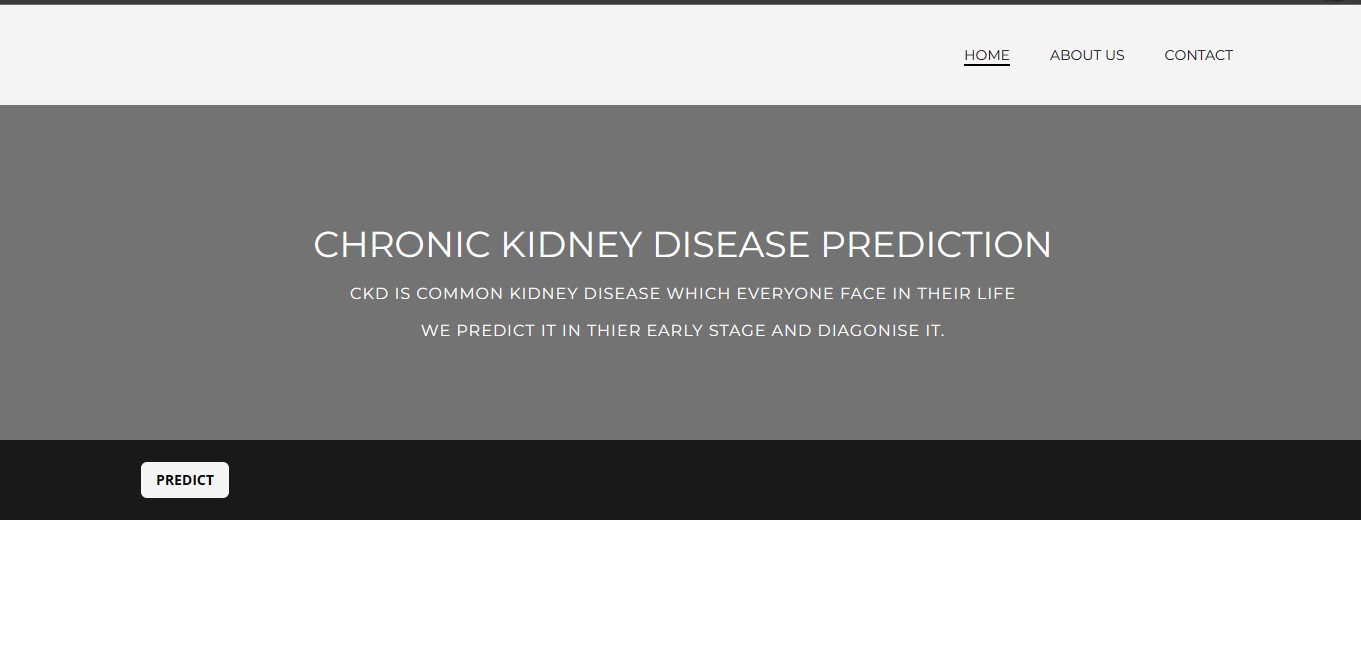


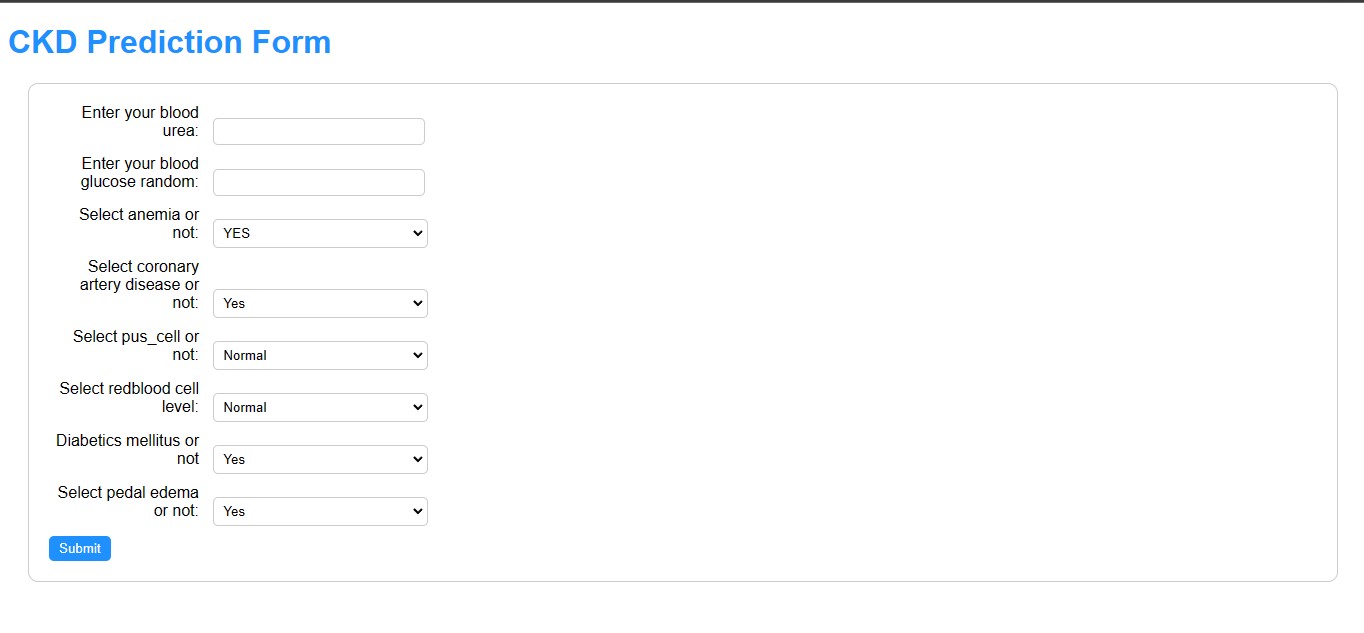








****

****

**ADVANTAGES:**

* Early intervention: Early treatment and prevention can reduce the disease's course and avert complications if CKD is discovered and treated at an early stage. In addition to lowering healthcare expenses, this can help patients live healthier lives.
* Improved outcomes: A reduction in the risk of renal failure, cardiovascular disease, and death can all be achieved with early detection and treatment of CKD.
* Cost-effectiveness: Early CKD diagnosis and treatment can save money by preventing the need for more expensive procedures like dialysis and kidney transplantation.
* Patient education and lifestyle modification: Early diagnosis of CKD can also offer a chance for patient education and lifestyle changes, such as dietary adjustments and increased physical activity, which can aid in slowing the disease's progression.
* Screening and monitoring: Increased screening and monitoring of high-risk populations, such as those with diabetes and hypertension, can result from early detection of CKD and aid in the early diagnosis and treatment of CKD.

**DISADVANTAGES:**

* False Positive Results: Early prediction models could give falsely positive results that claim someone has CKD even while they do not. Unnecessary medical procedures and treatments may result from this, which can be expensive and potentially dangerous.
* False Negative Results: The same is true for early prediction models, which might give falsely negative findings that suggest a person does not have CKD when in fact they do. This may cause a delay in diagnosis and treatment, which could be harmful to the health of the patient.
* Limited Generalizability: Early prediction models are often developed and validated on specific populations, and may not be generalizable to other populations. This can lead to inaccurate predictions for certain groups of people.
* Ethical Concerns: Early-warning systems can cause ethical problems, notably with regard to discrimination and privacy. The models might be used to discriminate against certain people or groups if they are based on private information, like genetic data. Furthermore, the models might be biassed against particular groups of individuals if they are used to decide which medical treatments should be administered.

Early forecasting techniques might create moral issues, notably in the areas of privacy and prejudice. Models that are based on private information, such as genetic data, could be used to discriminate against specific people or groups. Furthermore, the models may be biassed against particular groups of individuals if they are used to decide which patients will receive medical treatment.

**APPLICATIONS:**

Early detection and prevention: Before any symptoms appear, CKD prediction models can assist in identifying those who are more susceptible to developing chronic kidney disease (CKD). As a result, disease progression can be slowed down or prevented through early intervention and preventive measures.

CKD prediction models can be used as a screening tool to find those who have a high chance of developing CKD. This could potentially prevent wasteful testing in low-risk patients and assist healthcare practitioners in prioritising which patients should be evaluated for CKD.

Treatment strategies can be made specifically for each patient based on their risk level with the aid of CKD prediction models. individuals with a high risk of developing CKD, for instance, may benefit from more regular monitoring and vigorous blood pressure control, whereas individuals with a low risk may just need less strenuous measures.

## Resource management: CKD prediction models can also assist healthcare professionals in more effective resource management. Healthcare practitioners can focus their resources on those patients who are most likely to benefit from therapies by identifying individuals who have a high risk of developing CKD.

## Overall, by identifying those who are at high risk of developing CKD and adjusting interventions accordingly, CKD prediction can help patients achieve better outcomes and make the best use of healthcare resources.

**CONCLUSION:**

In conclusion, early diagnosis and prognostication tools for chronic kidney disease (CKD) have benefits and drawbacks. The accuracy, data availability, generalizability, and ethical issues they have limit their capacity to help in the early detection and management of CKD. To guarantee accurate diagnosis and suitable treatment, it is crucial to take these characteristics into account when utilising early prediction models and to combine them with other clinical assessments and diagnostic procedures. These constraints should continue to be addressed in order to increase the effectiveness of early CKD prediction models in clinical settings.

**FUTURE SCOPE:**

Machine learning (ML) has shown promising results in predicting kidney disease in recent years. There is a vast scope for future research in this field to further improve the accuracy of predictions and make them more useful for clinical decision-making. Here are some potential areas for future exploration:

Incorporating more data sources: Currently, most ML models for kidney disease prediction rely on laboratory values and clinical data. However, there is a wealth of other data sources that could be incorporated, such as electronic health records, genomics, and imaging data.

Improving the accuracy of predictions: While ML models have shown good performance in predicting kidney disease, there is still room for improvement. Future research could focus on developing more advanced algorithms, feature engineering, and ensemble methods to further improve the accuracy of predictions.

Developing personalized prediction models: Currently, most ML models provide a general prediction for the likelihood of developing kidney disease. However, personalized prediction models could be developed that take into account individual patient characteristics, such as age, sex, and comorbidities, to provide tailored predictions.

Incorporating longitudinal data: Most ML models for kidney disease prediction use cross-sectional data. However, longitudinal data could be incorporated to provide a more comprehensive understanding of disease progression and to predict future risk.

**APPENDIX:**

import pandas as pd

import numpy as np

from collections import Counter as c

import matplotlib.pyplot as plt

import seaborn as sns

import missingno as msno

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

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

from sklearn.preprocessing import LabelEncoder

from sklearn.linear\_model import LogisticRegression

import pickle

from google.colab import files

uploaded=files.upload()

from google.colab import files

uploaded=files.upload()

data.head()

data.tail() #return you the last 5 rows values

data.head(10)

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

data.columns

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

'sugar', 'red\_blood\_cell', 'pus\_cell', 'pus\_cell\_clumps', 'bacteria',

'blood\_glucose\_random','blood\_urea', 'serum\_creatinine', 'sodium', 'potassium',

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

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

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

data.columns

data.info()

data.isnull().any()

data.isnull().sum()

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

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

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

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

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

data['sodium'].fillna(data['sodium'].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\_cell'].fillna(data['red\_blood\_cell'].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)

data.describe()

data['class'].unique()

data['class']=data['class'].replace("ckd\t","ckd")

data['class'].unique()

np.unique(data.dtypes,return\_counts=True)

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

print(catcols)

for i in catcols:

print("Columns:",i)

print(c(data[i]))

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

print(catcols)

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

print(catcols)

for i in contcols:

print("Continous Columns:",i)

print(c(data[i]))

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

contcols.remove('specific\_gravity')

contcols.remove('albumin')

contcols.remove('sugar')

print(contcols)

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

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

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

print(contcols)

contcols.add('specific\_gravity')

contcols.add('albumin')

contcols.add('sugar')

print(catcols)

contcols.add('specific\_gravity')

contcols.add('albumin')

contcols.add('sugar')

print(catcols)

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

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

data['diabetesmellitus']=data.diabetesmellitus.replace(to\_replace={'\tno':'no','\tyes':'yes'})

c(data['diabetesmellitus'])

data.isnull().any()

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')

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

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

data['sodium'].fillna(data['sodium'].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\_cell'].fillna(data['red\_blood\_cell'].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)

data.isnull().sum()

catcots=['anemia','pedal\_edema','appetite','bacteria','class','coronary\_artery\_disease','diabetesmellitus','hypertension','pus\_cell\_clumps','red\_blood\_cell']

from sklearn.preprocessing import LabelEncoder

for i in catcots:

print("LABEL ENCODING OF:",i)

LEi=LabelEncoder()

print(c(data[i]))

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

print(c(data[i]))

print("\*"\*100)

sns.distplot(data.age)

import matplotlib.pyplot as plt

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

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

plt.xlabel('age')

plt.ylabel('blood\_pressure')

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

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

plotnumber=1

for column in contcols:

if plotnumber<=11:

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

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

plt.xlabel(column,fontsize=20)

plotnumber+=1

plt.show()

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

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

plt.xticks(rotation=45)

plt.yticks(rotation=45)

plt.show()

data['class'].unique()

sns.countplot(data['class'])

setcols=['red\_blood\_cells','pus\_cell','blood\_glucose\_random','blood\_urea',

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

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

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

print(x.shape)

print(y.shape)

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

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

print(x\_train.shape)

print(y\_train.shape)

print(x\_test.shape)

print(y\_test.shape)

import tensorflow

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

classification=Sequential()

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

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

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

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

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

classification.compile(optimizer='adam',loss='binary\_crossentropy',metrics=['accuracy'])

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

from sklearn.ensemble import RandomForestClassifier

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

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

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

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

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

from sklearn.ensemble import RandomForestClassifier

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

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

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

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

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

from sklearn.linear\_model import LogisticRegression

lgr=LogisticRegression()

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

LogisticRegression()

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

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

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

print(y\_pred)

(y\_pred)

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

print(y\_pred)

(y\_pred)

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

print(y\_pred)

(y\_pred) classification.save("ckd.h5")

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

y\_pred

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\_vaue=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 change of CKD!')

else

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

from sklearn import model\_selection

dfs=[]

models=[

('LogReg',LogisticRegression()),

('RF',RandomForestClassifier()),

('DecisionTree',DecisionTreeClassifier()),

]

results=[]

names=[]

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

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

for name,model in models:

kfold=model\_selectioon.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(name)

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

results.append(cv\_results)

names.append(name)

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

this\_df['model']=name

dfs.append(this\_df)

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

return final