Chronic kidney disease

Introduction:

CKD is a condition in which the kidneys are damaged and cannot filter blood as well as they should. Because of this, excess fluid and waste from blood remain in the body and may cause other health problems, such as heart disease and stroke.

Some other health consequences of CKD include:

* Anemia or low number of red blood cells
* Increased occurrence of infections
* Low calcium levels, high potassium levels, and high phosphorus levels in the blood
* Loss of appetite or eating less
* Depression or lower quality of life

CKD has varying levels of seriousness. It usually gets worse over time though treatment has been shown to slow progression. If left untreated, CKD can progress to kidney failure and early cardiovascular disease. When the kidneys stop working, [dialysis](https://www.cdc.gov/dialysis/) or kidney transplant is needed for survival. Kidney failure treated with dialysis or kidney transplant is called end-stage renal disease (ESRD).

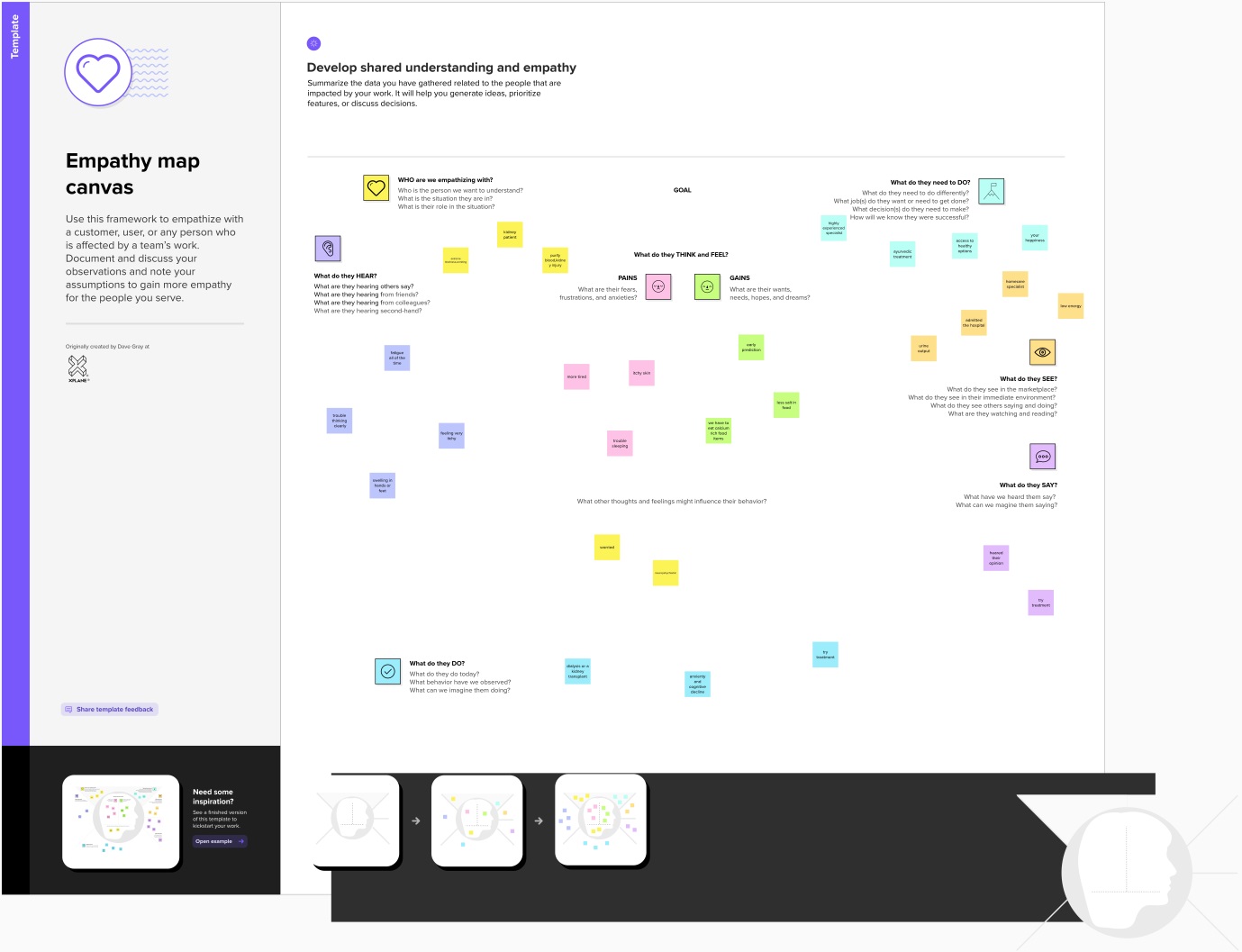
Chronic kidney disease, also called chronic kidney failure, involves a gradual loss of kidney function. Your kidneys filter wastes and excess fluids from your blood, which are then removed in your urine. Advanced chronic kidney disease can cause dangerous levels of fluid, electrolytes and wastes to build up in your body.

In the early stages of chronic kidney disease, you might have few signs or symptoms. You might not realize that you have kidney disease until the condition is advanced.

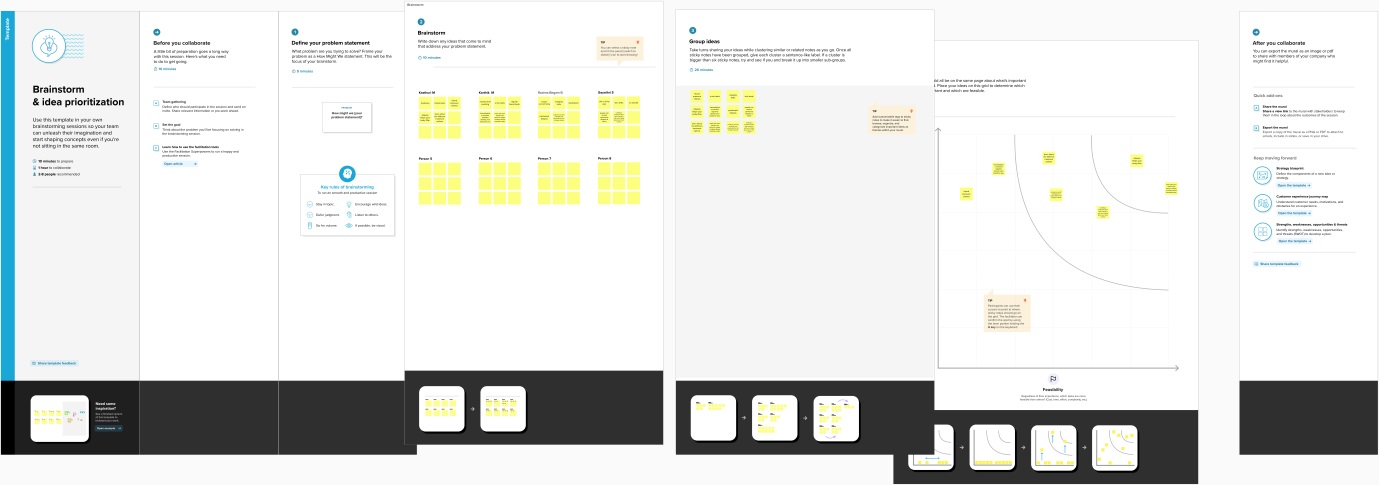
Treatment for chronic kidney disease focuses on slowing the progression of kidney damage, usually by controlling the cause. But, even controlling the cause might not keep kidney damage from progressing. Chronic kidney disease can progress to end-stage kidney failure, which is fatal without artificial filtering (dialysis) or a kidney transplant

Problem definition and design thinking

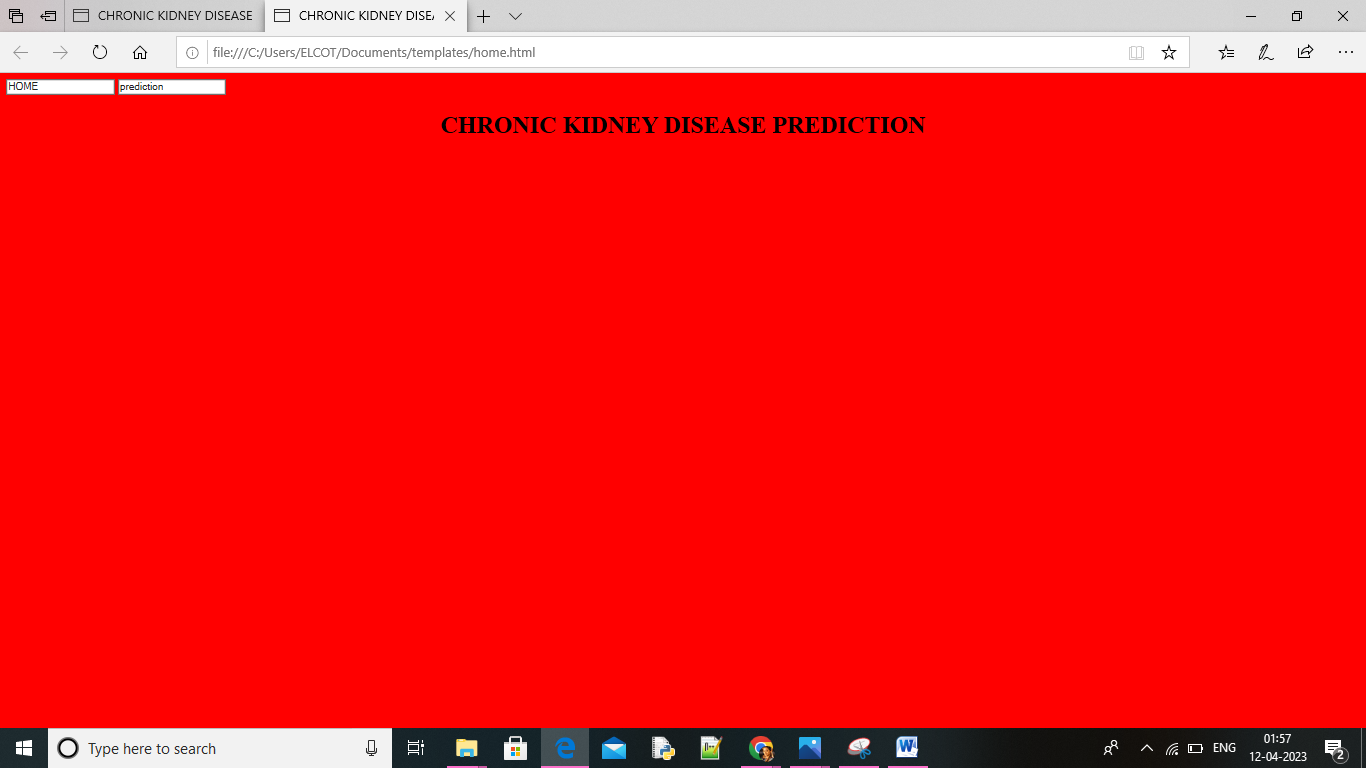
Empathy map:

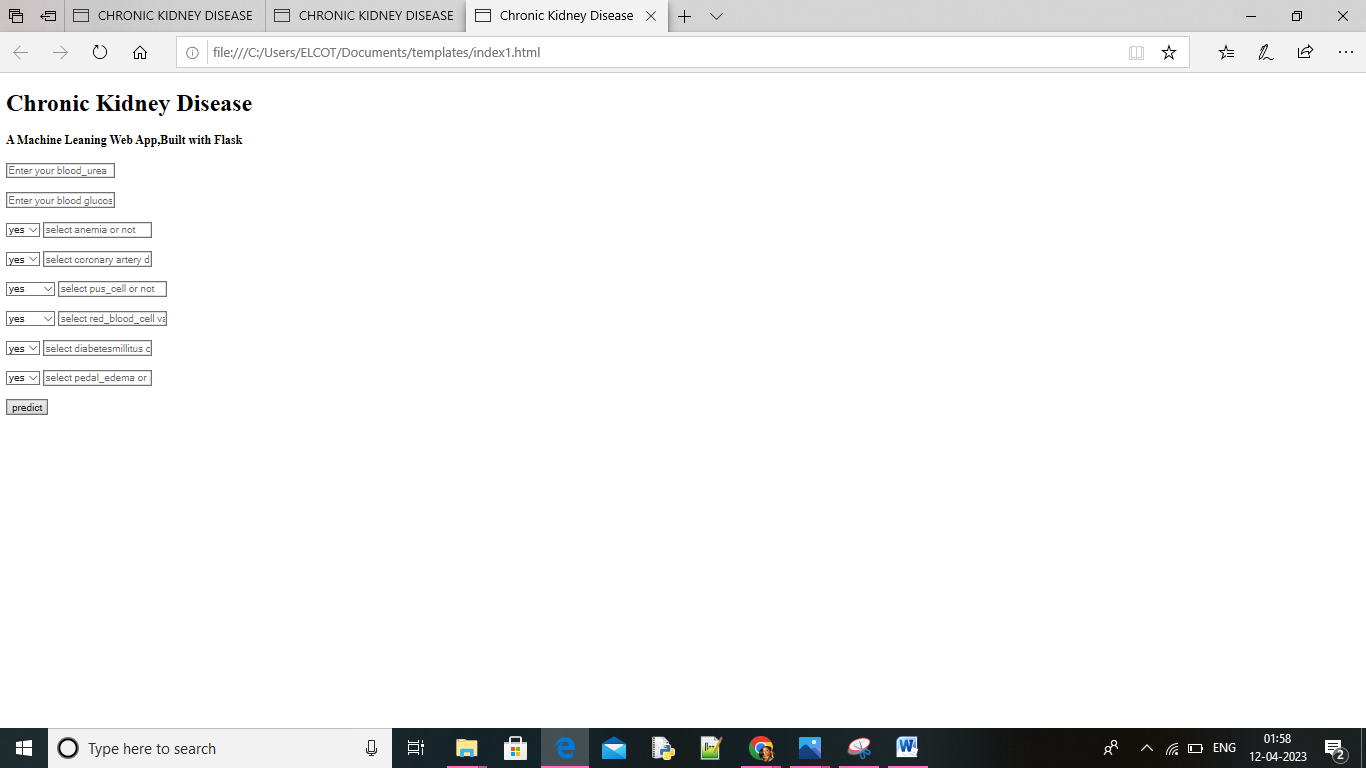


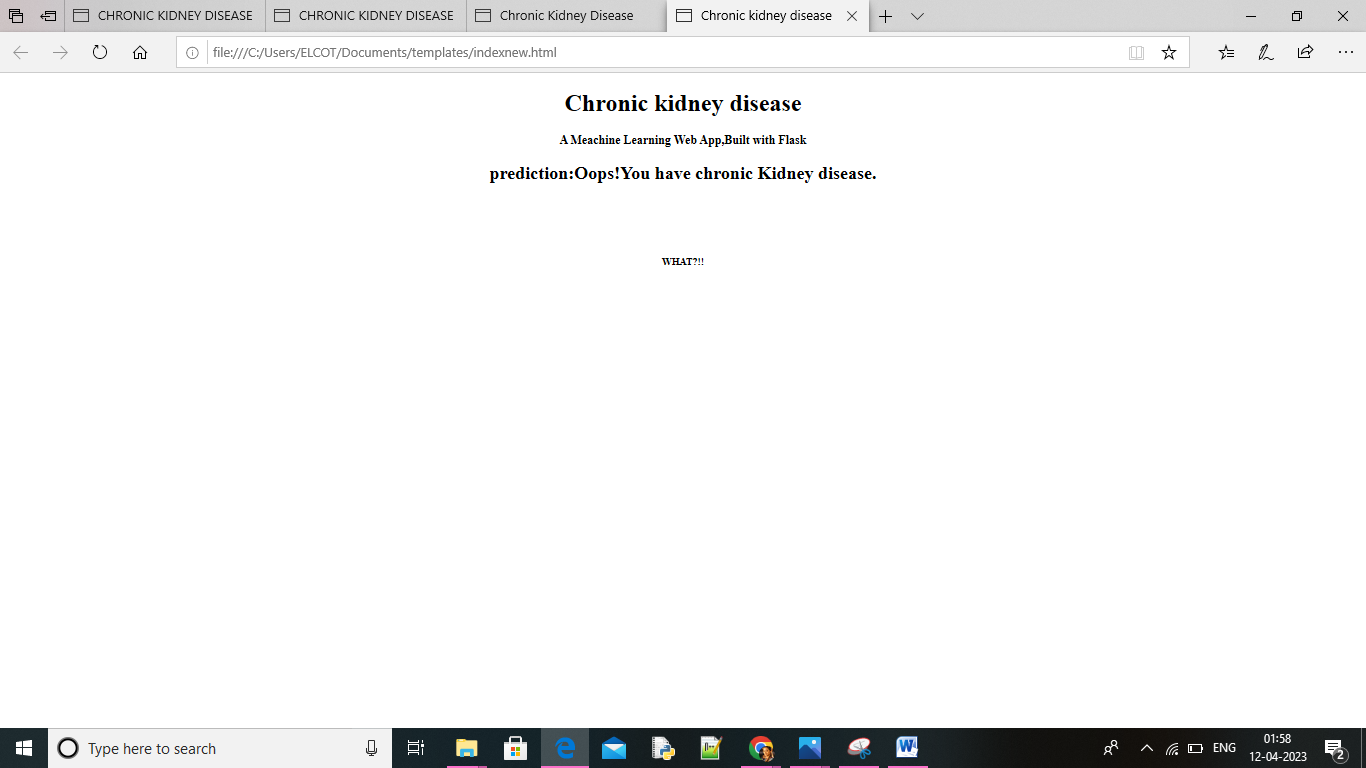
Ideation and brainstorming map:

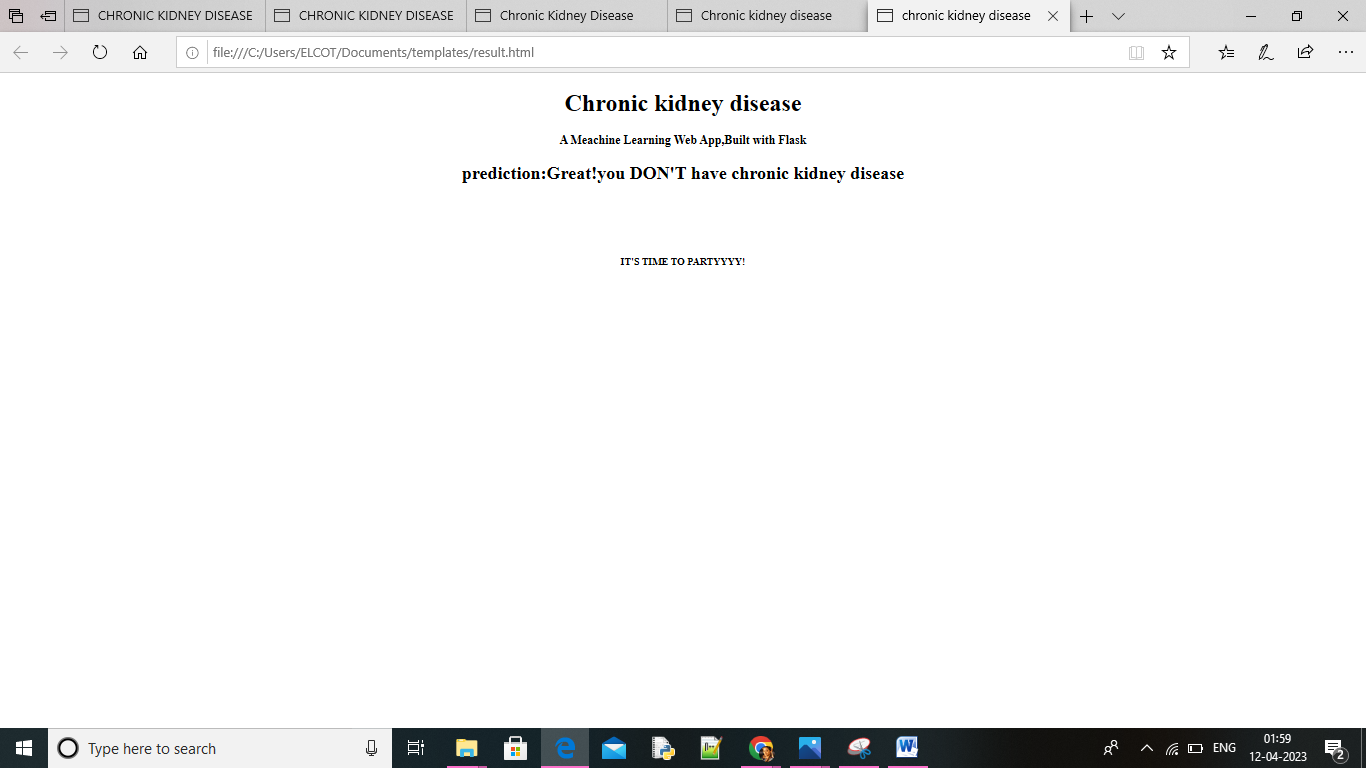


Result:









Advantages and disadvantages

Advantages:

Slows down the progression of the disease

Reduce the risk of complications such as cardiovascular disease

Improves overall health and quality of life

May delay or eliminate the need for dialysis or kidney transplant

Can be tailored to individual needs and preference

May be covered by insurance or government programs

Disadvantages:

Can have side effects such as nausea and vomiting

May require regular monitoring and adjustments

Can be expensive and especially if not covered by temporarily

Can require lifestyle changes such as dietary restrictions or increased physical activity

May not be able to reverse damage that has already occurred in the kidneys

Applications

Chronic disease education programs are increasingly adopting mobile health applications to support self-management practices, reinforcing both the knowledge and understanding components of Kolb’s educational cycle. Many of these programs rely on wireless communication among peripheral objects (i.e., scales, blood pressure cuffs, glucometers) and smartphones, allowing patients to view their home-recorded data (i.e., blood pressure, weight, eating habits) and potentially submit them to a health care provider for clinical care. In the TASMIN-SR trial,, patients with hypertension and early CKD randomly assigned to receive blood pressure self-monitoring and self-titration of their blood pressure medications experienced an 8.4 mmHg (95%CI 1.1-15.8 mmHg) decrease in systolic blood pressure compared to those who received usual care.[31](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5324778/#R31) Little is known whether patients with more advanced CKD will experience additional kidney health benefits if mobile health applications are incorporated into usual CKD care. In one recent pilot study, ,researchers examined the integration of a smart-phone self-management support program that supported 4 behavioral elements (monitoring blood pressure, managing medications, assessing symptoms, and tracking selected laboratory test results) into usual CKD care among 47 study participants. User adherence and satisfaction with the program were high, with >80% of users performing routine assessments with the application, with notable engagement around medication reconciliation. In addition to experiencing a mean reduction in home systolic blood pressure of −3.4mmHg (−5.0 to −1.8 mmHg), patients indicated that they felt more in-control of their CKD after using the application.

conclusion

 Acute renal failure is a serious medical condition that could complicate the course of many of your patients. The mortality rate from acute tubular necrosis is around 50% and hasn't changed much over the past 3 decades, despite significant advances in supportive care.

Future scope

Kidney transplantation was first performed in 1954 by Dr Joseph Murray and is the current gold standard for treatment. However, there are still fewer donors than the relentlessly increasing waiting list

Kidney transplantation was first performed in 1954 by Dr Joseph Murray and is the current gold standard for treatment. However, there are still fewer donors than the relentlessly increasing waiting list [[5](javascript:;)]. Chronic hemodialysis was introduced in 1960 by Dr Belding Scribner, and despite being the major form of renal replacement therapy (RRT) it is associated with numerous short- and long-term complications. Importantly, the life expectancy of patients in dialysis in their twenties is 40 years shorter than for the general population

 Peritoneal dialysis is less frequently utilized, and beside specific complications (peritonitis, cellulitis, metabolic disturbances) it is limited by almost inexorable membrane failure

APPENDIX

A.source code

Chronic kidney disease

import pandas as pd

import numpy as np

from collections import Counter as catcols

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

# Commented out IPython magic to ensure Python compatibility.

from google.colab import drive

drive.mount('/content/drive')

# %cp '/content/drive/MyDrive/Colab Notebooks/kidney\_disease\_prediction/training/kidney\_disease.csv' '/content/'

data = pd.read\_csv('kidney\_disease.csv')

data.head()

data.columns

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

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

'blood glucose random','blood\_urea','serum\_creatinine','sodium','potassium',

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

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

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

data.columns

data.head()

data.info()

data.isnull().any()

#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)

#catcols=set(data.dtypes[data.dtypes=='0'].index.value)

#print(catcols)

#for i in catcols:

#print("Columns:",i)

#print(c(data[i]))

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

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

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

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

#print(catcols)

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

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

'hypertension','pus\_cell','pus\_cell\_clumps','red\_blood\_cells']

from sklearn.preprocessing import LabelEncoder

for i in catcols:

print("LABEL ENCODING OF:",i)

LEi=LabelEncoder()

print(c(data[i]))

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

print(c(data[i]))

print("\*"\*100)

contcols=set(data.dtypes[data.dtypes!='0'].index.values)

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

print(contcols)

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)

#catcols.add('specific\_gravity')

#catcols.add('albumin')

#catcols.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','yes':'yes'})

#c(data['diabetesmellitus'])

#data.describe()

import seaborn as sns

#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

contcols=['potassium','hemoglobin','age','sodium',

'packed\_cell\_volume','serum\_creatinine','red\_blood\_cell\_count','blood\_pressure',

'blood\_urea','white\_blood\_cell\_count','blood glucose random']

for column in contcols:

if plotnumber<=11:

ax=plt.subplot(3,4,plotnumber)#3,4 is refer to 3x4 matrix

#plt.scatter(data['age'],data[column],color=blue)

plt.xlabel(column,fontsize=20)

#plotnumber+=1

plt.show()

#HEAT MAP #correlation of parameters

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

#sns.(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'])

#performing feature Scaling operation using standard scaller on X part of the dataset because

#there different type of values in the columns

from sklearn.preprocessing import StandardScaler

sc=StandardScaler()

#x\_bal=sc.fit\_transform(x)

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

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

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

#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)

import tensorflow

#from tensorflow.keras.model import Sequential

#from tensorflow.keras.model import Dence

#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'))

#Compiling the ANN model

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

#Training the model

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

from sklearn.ensemble import RandomForestClassifier

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

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

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

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

from sklearn.tree import DecisionTreeClassifier

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

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

#y\_predict=dtc.predict(x\_test)

#y\_predict

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

from sklearn.linear\_model import LogisticRegression

lgr=LogisticRegression()

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

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

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

#logistic Regression

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

#print(y\_pred)

#(y\_pred)

#DecisionTree Classsifier

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

#print(y\_pred)

#(y\_pred)

#Random Forest Classifier

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

#print(y\_pred)

#(y\_pred)

#classification.save("ckd.h5")

#Testing the model

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

#y\_pred

#y\_pred=(y\_pred>0.5)

#y\_pred

def predict\_exit(sample\_value):

# Convert list to numpy array

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

#Reshape because sample\_value contains only 1 record

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

#Feature Scaling

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('predicttion:High change of CKD!')

#else:

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

from sklearn import model\_selection

dfs=[]

models=[

('LogReg', LogisticRegression()),

('RF', RandomForestClassifier()),

('DecisionTree',DecisionTreeClassifier())

]

results=[]

names=[]

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

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

for name,model in models:

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

#cv\_results=model\_selection.cross\_validation(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

#Making the Confusion Matix

from sklearn.metrics import confusion\_matrix

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

#cm

#Plotting confusion matrix

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

#sns.heatmap(cm,cmap='Blues',annot=True,xticklabels=['no ckd','ckd'],yticklabels=['no ckd','ckd'])

plt.xlabel('Predictted values')

plt.ylabel('Actual values')

plt.title('Confusion Matrix for Logistic Regression model')

plt.show()

#Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix

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

#cm

#plotting confusion matrix

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

#sns.heatmap(cm,cmap='Blues',annot=True,xticklabels=['no ckd','ckd'],yticklabels=['no ckd','ckd'])

plt.xlabel('Predicted values')

plt.ylabel('Actual values')

plt.title('confusion Matrix for RandomForestClassifier')

plt.show()

#Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix

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

#cm

#plotting confusion matrix

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 DecisionTreeClassifier')

plt.show()

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

#Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix

#cm=confusion\_matrix(y\_test,y\_pred)

#cm

#Plotting confusion matrix

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 ANN model')

plt.show()

bootstraps=[]

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

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

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

#bootstrap.append(bootstrap)

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

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

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

##PERFORMANCE METRICS

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

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

##TIME METRICS

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

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

import matplotlib.pyplot as plt

import seaborn as sns

plt.figure(figsize=(20,12))

sns.set(font\_scale=2.5)

#g=sns.boxplot(x="model",y="metrics",data=results\_long\_nofit,palette="set3")

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

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

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

#pickle.dump(lgr,open('CkD.pk1','wb'))

App.py

from flask import Flask,render\_template,request

import numpy as np

import pickle

# Commented out IPython magic to ensure Python compatibility.

from google.colab import drive

drive.mount('/content/drive')

# %cp -r '/content/drive/MyDrive/kidney\_disease\_prediction/Flask/templates/' '/content/'

# install pyngrok

!pip install pyngrok

#import ngrok

from pyngrok import ngrok

!killall ngrok

app=Flask(\_\_name\_\_)

ngrok.set\_auth\_token("2OGvm12ANc3b2QacMPhowpL66CH\_72vmdKqhkewAvjb8KhEH5")

public\_url=ngrok.connect(5000)

print("url:",public\_url)

#model=pickle.load(open('CKD.pk1','rb'))

@app.route('/')

def home():

return render\_template('home.html')

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

def prediction():

return render\_template('indexnew.html')

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

def my\_home():

return render\_template('home.html')

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

def predict():

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

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

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

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

'diabetesmellitus','pedal\_edema']

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

output=model.predict(df)

return render\_template('result.html',prediction\_text=output)

app.run(debug=True)