Code

```
from tkinter import *

import sqlite3

import re

from tkinter import messagebox

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

#import seaborn as sns

import warnings

warnings.filterwarnings('ignore')



class patient1:

# root = Tk()

 def __init__(self,root):

  self.root= root

  root.title("Input Form")

  root.geometry('500x500')

  #Button(root, text='Reset',width=20,bg='brown',fg='white',command=validation).place(x=180,y=430)

  # Centering Root Window on Screen


  w = 800 # width for the Tk root
```

```
h = 600 # height for the Tk root

# get screen width and height
ws = root.winfo_screenwidth() # width of the screen
hs = root.winfo_screenheight() # height of the screen

# calculate x and y coordinates for the Tk root window
x = (ws/2) - (w/2)
y = (hs/2) - (h/2)


root["bg"] = '#98fb98'
# set the dimensions of the screen
# and where it is placed
root.geometry('%dx%d+%d+%d' % (w, h, x, y))


self.v = IntVar()
self.hb2=StringVar()
self.rbc2=StringVar()
self.sg2 =StringVar()
self.al2=StringVar()
self.sc2=StringVar()
self.ht2=StringVar()
self.sod2=StringVar()
```

```
  self.bp2 =StringVar()
  self.wbc2=StringVar()
  self.age2=StringVar()



# labels for the window
  self.heading = Label(self.root, text="Early Detection of Chronic Kidney Disease using ML", font=('Centaur 20 bold'), bg='#98fb98',fg='red')
  self.heading.place(x=60, y=10)



  self.hb1 = Label(self.root, text="Hemoglobin", font=('arial 12 bold'),bg='#98fb98')
  self.hb1.place(x=50, y=70)

  self.hb_ent = Entry(self.root, width=30, textvar=self.hb2)
  self.hb_ent.place(x=220, y=70)

  self.rbc1 = Label(self.root, text="Red Blood Cells", font=('arial 12 bold'),bg='#98fb98')
  self.rbc1.place(x=50, y=120)

  self.rbc_ent = Entry(self.root, width=30, textvar=self.rbc2)
  self.rbc_ent.place(x=220, y=120)

  self.sg1 = Label(self.root, text="Specific Gravity", font=('arial 12 bold'),bg='#98fb98')
  self.sg1.place(x=50, y=170)
self.sg_ent = Entry(self.root, width=30, textvar=self.sg2)

self.sg_ent.place(x=220, y=170)


self.al1 = Label(self.root, text="Albumin", font=('arial 12 bold'),bg='#98fb98')

self.al1.place(x=50, y=220)


self.al_ent = Entry(self.root, width=30, textvar=self.al2)

self.al_ent.place(x=220, y=220)


self.sc1 = Label(self.root, text="Searum Creatinite", font=('arial 12 bold'),bg='#98fb98')

self.sc1.place(x=50, y=270)


self.sc_ent = Entry(self.root, width=30, textvar=self.sc2)

self.sc_ent.place(x=220, y=270)


self.ht1 = Label(self.root, text="Hypertension", font=('arial 12 bold'),bg='#98fb98')

self.ht1.place(x=50, y=320)


self.ht_ent = Entry(self.root, width=30, textvar=self.ht2)

self.ht_ent.place(x=220, y=320)




self.sod1 = Label(self.root, text="Sodium", font=('arial 12 bold'),bg='#98fb98')

self.sod1.place(x=50, y=370)
```

```
  self.sod_ent = Entry(self.root, width=30, textvar=self.sod2)
  self.sod_ent.place(x=220, y=370)

  self.bp1 = Label(self.root, text="Blood Pressure", font=('arial 12 bold'),bg='#98fb98')
  self.bp1.place(x=50, y=420)

  self.bp_ent = Entry(self.root, width=30, textvar=self.bp2)
  self.bp_ent.place(x=220, y=420)



  self.wbc1 = Label(self.root, text="White Blood Cells", font=('arial 12 bold'),bg='#98fb98')
  self.wbc1.place(x=50, y=470)

  self.wbc_ent = Entry(self.root, width=30, textvar=self.wbc2)
  self.wbc_ent.place(x=220, y=470)



  self.age1 = Label(self.root, text="Age", font=('arial 12 bold'),bg='#98fb98')
  self.age1.place(x=50, y=520)

  self.age_ent = Entry(self.root, width=30, textvar=self.age2)
  self.age_ent.place(x=220, y=520)

# button to perform a command
```

```
  self.submit = Button(self.root, text="Initialize DS", font="aried 12 bold",width=10, height=2, bg='lightgreen',command=self.initds)
  self.submit.place(x=450, y=100)


  self.submit1 = Button(self.root, text="Cleaning DS", font="aried 12 bold",width=10, height=2, bg='lightgreen',command=self.logform)
  self.submit1.place(x=450, y=200)


  self.submit2 = Button(self.root, text="Test Train Split", font="aried 12 bold",width=10, height=2, bg='lightgreen',command=self.splitds)
  self.submit2.place(x=450, y=300)


  self.submit3 = Button(self.root, text="Create Model ", font="aried 12 bold",width=10, height=2, bg='lightgreen',command=self.classifyy)
  self.submit3.place(x=450, y=400)


  self.submit4 = Button(self.root, text="Prediction ", font="aried 12 bold",width=10, height=2, bg='lightgreen',command=self.predictt)
  self.submit4.place(x=450, y=500)


 def initds(self):
  self.df=pd.read_csv("kidney_disease.csv")
  print("The dataset shape is {}".format(self.df.shape))
  # remove "id" feature
  self.df.drop('id',axis=1,inplace=True)
  print("DS Init Success")
 def logform(self):

  #in our dataset some features ['pcv','wc','rc','dm','cad','classification'] contains some special
character.so replace them with appropriate values.


  # cleaning 'PCV'

  self.df['pcv']=self.df['pcv'].apply(lambda x:x if type(x)==type(3.5) else
x.replace('\t43','43').replace('\t?','Nan'))


  # cleaning "WC"

  self.df['wc']=self.df['wc'].apply(lambda x:x if type(x)==type(3.5) else
x.replace('\t?','Nan').replace('\t6200','6200').replace('\t8400','8400'))


  # cleaning "RC"

  self.df['rc']=self.df['rc'].apply(lambda x:x if type(x)==type(3.5) else x.replace('\t?','Nan'))


  # cleaning "dm"

  self.df['dm']=self.df['dm'].apply(lambda x:x if type(x)==type(3.5) else
x.replace('\tno','no').replace('\tyes','yes').replace(' yes','yes'))


  # cleaning "CAD"

  self.df['cad']=self.df['cad'].apply(lambda x:x if type(x)==type(3.5) else x.replace('\tno','no'))


  # cleaning "Classification"

  self.df['classification']=self.df['classification'].apply(lambda x:x if type(x)==type(3.5) else
x.replace('ckd\t','ckd'))
```

```
#Note: Some features are mistyped as "object".so convert them into "float" type


mistyped=[['pcv','rc','wc']]

for i in mistyped:

 self.df[i]=self.df[i].astype('float')

#  define categoricsl features

cat_cols=list(self.df.select_dtypes('object'))

cat_cols


# define numeric features

self.num_cols=list(self.df.select_dtypes(['int64','float64']))

self.num_cols

# Checking missing/Nan values

self.df.isnull().sum().sort_values(ascending=False)


# Let's impute Nan Values with median in numeric features

for col in self.num_cols:

 self.df[col]=self.df[col].fillna(self.df[col].median())

# let's impute categorical features with most frequent value

self.df['rbc'].fillna('normal',inplace=True)

self.df['pc'].fillna('normal',inplace=True)

self.df['pcc'].fillna('notpresent',inplace=True)

self.df['ba'].fillna('notpresent',inplace=True)

self.df['htn'].fillna('no',inplace=True)

self.df['dm'].fillna('no',inplace=True)
```

```
 self.df['cad'].fillna('no',inplace=True)

 self.df['appet'].fillna('good',inplace=True)

 self.df['pe'].fillna('no',inplace=True)

 self.df['ane'].fillna('no',inplace=True)

 self.df.isna().sum().sort_values(ascending=False)

 print("DS Cleaning Finished")


def splitds(self):


 self.df['rbc']=self.df['rbc'].map({'normal':0,'abnormal':1})

 self.df['pc']=self.df['pc'].map({'normal':0,'abnormal':1})

 self.df['pcc']=self.df['pcc'].map({'notpresent':0,'present':1})

 self.df['ba']=self.df['ba'].map({'notpresent':0,'present':1})

 self.df['htn']=self.df['htn'].map({'no':0,'yes':1})

 self.df['dm']=self.df['dm'].map({'no':0,'yes':1})

 self.df['cad']=self.df['cad'].map({'no':0,'yes':1})

 self.df['pe']=self.df['pe'].map({'no':0,'yes':1})

 self.df['ane']=self.df['ane'].map({'no':0,'yes':1})

 self.df['appet']=self.df['appet'].map({'good':0,'poor':1})


 # scaling with MinMaxScaler

 from sklearn.preprocessing import StandardScaler,MinMaxScaler

 mm_scaler=MinMaxScaler()

 self.df[self.num_cols]=mm_scaler.fit_transform(self.df[self.num_cols])
 from sklearn.model_selection import train_test_split
 x=self.df.drop('classification',axis=1)
 y=self.df['classification']

 self.X_train,self.X_test,self.y_train,self.y_test=train_test_split(x,y,test_size=0.2,random_state=0)
 print("X_train size {} , X_test size {}".format(self.X_train.shape,self.X_test.shape))

def classifyy(self):
 from sklearn.ensemble import RandomForestClassifier
 from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
 # Creating Random Forest model
 self.rf=RandomForestClassifier(max_depth=5,n_estimators=5)
 self.rf.fit(self.X_train,self.y_train)
 self.y_pred=self.rf.predict(self.X_test)

 # Accuracy score
 score=round(accuracy_score(self.y_test,self.y_pred),3)
 print("Accuracy on the Test set: {}".format(score))

 # Classification report
 print(classification_report(self.y_test,self.y_pred))

 # Creating a confusion matrix for training set
 self.y_train_pred=self.rf.predict(self.X_train)
 # Accuracy score
 score=round(accuracy_score(self.y_train,self.y_train_pred),3)
 print("Accuracy on training set: {}".format(score))

 print(classification_report(self.y_train,self.y_train_pred))

def predict1(self,hemo,rc,sg,al,sc,htn,sod,bp,wc,age):
 print(hemo)
 hemo=float(hemo)
 rc=float(rc)
 sg=float(sg)
 sc=float(sc)
 htn=int(htn)
 sod=float(sod)
 bp=float(bp)
 wc=float(wc)
 age=int(age)

 x=[[hemo,rc,sg,al,sc,htn,sod,bp,wc,age]]
 return self.rf.predict(x)



def predictt(self):
 self.X_train=self.X_train[['hemo','rc','sg','al','sc','htn','sod','bp','wc','age']]
 self.X_test=self.X_test[['hemo','rc','sg','al','sc','htn','sod','bp','wc','age']]
self.rf.fit(self.X_train,self.y_train)
hb3=self.hb2.get()
rbc3=self.rbc2.get()
sg3=self.sg2.get()
al3=self.al2.get()
sc3=self.sc2.get()
ht3=self.ht2.get()
sod3=self.sod2.get()
bp3=self.bp2.get()
wbc3=self.wbc2.get()
age3=self.age2.get()

prediction = self.predict1(hb3,rbc3,sg3,al3,sc3,ht3,sod3,bp3,wbc3,age3)[0]
#prediction = self.predict1(67.4,7.2,0.99,4,17.0,1,160.6,87,22089,36)[0]
print(prediction)
if prediction:
 print('Oops! You have Chronic Kidney Disease.')
else:
 print("Great! You don't have Chronic Kidney Disease.")
```

```
if __name__ == '__main__':

 root = Tk()

 application=patient1(root)
 #root.geometry('500x500')
 root.mainloop()
```