DETECTION OF CHRONIC KIDNEY DISEASE USING PYTHON

A MAJOR PROJECT REPORT

SUBMITTED BY

CH.EN.U4AIE21104	A.HARSHAVARDHAN
CH.EN.U4AIE21108	B.CHAITANYA
CH.EN.U4AIE21145	S.PRANEETH
CH.EN.U4AIE21159	M.VEDASAMPREETHA

In partial fulfilment for the award of the degree

Of

BACHELOR OF TECHNOLOGY

IN

ARTIFICIAL INTELLIGENCE

AMRITA SCHOOL OF ENGINEERING , CHENNAI

January 2022

AMRITA VISHWA VIDYAPEETHAM

AMRITA SCHOOL OF ENGINEERING, CHENNAI

BONAFIDE CERTIFICATE

This is to certify that the major project report entitled **" DETECTION OF CHRONIC KIDNEY DISEASE USING PYTHON"** submitted by

CH.EN.U4AIE21104	A.HARSHAVARDHAN
CH.EN.U4AIE21108	B.CHAITANYA
CH.EN.U4AIE21145	S.PRANEETH
CH.EN.U4AIE21159	M.VEDA SAMPREETHA

In partial fulfillment of the requirements for the award of the **Degree of Bachelor of Technology** in **ARTIFICIAL INTELLIGENCE** is a bonafide record of the work carried out under my guidance and supervision at Amrita School of Engineering, Chennai.

Signature

Dr. I R Oviya

Asst. Professor(Sr. Gr), Science Dept.

This project report was evaluated by us on ……

INTERNAL EXAMINER EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We would like to offer our sincere pranams at the lotus feet of Universal guru, **MATA AMRITANANDAMAYI DEVI** who blessed us with her grace to make this a successful major project.

We express our deep sense of gratitude to **Dr. Prasanna kumar**, Chairperson, for his constant help, suggestions, and inspiring guidance. We are grateful to our guide **Dr. I R Oviya**, Assistant Professor (Sr. Gr), Department of Science, ASE, Chennai for his invaluable support and guidance during the major project work.

We would also like to extend our gratitude to our director **Shri. Manikandan**, Principal Dr. Shankar who has always encouraged us. We are also thankful to all our classmates who have always been a source of strength, for always being there and extending their valuable bits of help to the successful completion of this work.

ABSTRACT

Treatment for chronic kidney disease focuses on slowing the progression of the kidney damage, usually by controlling the underlying cause. Chronic kidney disease can progress to end-stage kidney failure, which is fatal without artificial filtering (dialysis) or a kidney transplant.

In the early stages of chronic kidney disease, you may have few signs or symptoms. Chronic kidney disease may not become apparent until your kidney function is significantly impaired.

In this project we will describe python program to predict and classify patience as having chronic kidney disease (ckd) or not using artificial neural networks.

This project proposes a Multi- Layered Perceptron Classifier that uses deep neural network in order to predict whether a patient has CKD or not. The model is trained on a dataset of about four hundred patients and considers diverse signs and symptoms which includes blood pressure, age, sugar level, red blood cell count, etc. The experimental results display that the proposed model can perform classification with the testing accuracy of 99 %. The aim is to help introduce Deep Learning methods in learning from the dataset attribute reports and detect CKD correctly to a large extent.

CHAPTER1

LITERATURE RIEVEW:

Lambodar Jena et al has suggested the use various algorithms on chronic kidney disease datasets and compare the results based on different parameters but only on single interface of WEKA and it has been calculated that multilayer perceptron algorithm performs the best among used algorithms.

Morteza Khavanin Zadeh et al suggested the prediction of early chronic kidney disease and data of 193 patients who underwent homodialysis in Hasheminejad Kidney Center were explored. Eight common attributes of the patients including age, sex, hypertension level, Diabetes Mellitus state, hemoglobin level, smoking behavior, location of Arteriovenous fistula, and thrombosis state were used in the machine learning process and only two algorithms are used for prediction process.

L.Jerlin Rubini et al used only three different algorithms such as radial basis function network, multilayer perceptron, and logistic regression. Also the interface used in this study only one.

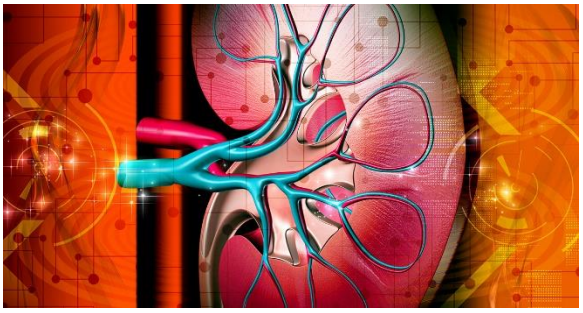
Shital Shah et al suggested the data mining approach which helps in detecting the chronic kidney effectively and realating it to the survival of the patient but again only with the help of limited algorithms and single interface.

Dr. S. Vijayarani et al simply concludes in this paper that comparison two different algorithms such as SVM and Naïve Bayes. It is concluded that SVM performs better than Naïve Bayes algorithm in case of chronic kidney disease with the help of single platform.

K.R.Lakshmi et al has compared the results of three data mining techniques that is Artificial Neural Networks, Decision tree and Logical Regression. It is used to elicit knowledge about the interaction between these variables and patient survival. Finally, ANN is suggested for Kidney dialysis to get better results with accuracy and performance.

Dr. S. Vijayarani et al The objective of this research work is to predict kidney diseases by using Support Vector Machine (SVM) and Artificial Neural Network (ANN). The aim of this work is to compare the performance of these two algorithms on the basis of its accuracy and execution time. From the experimental results it is observed that the performance of the ANN is better than the other algorithm.

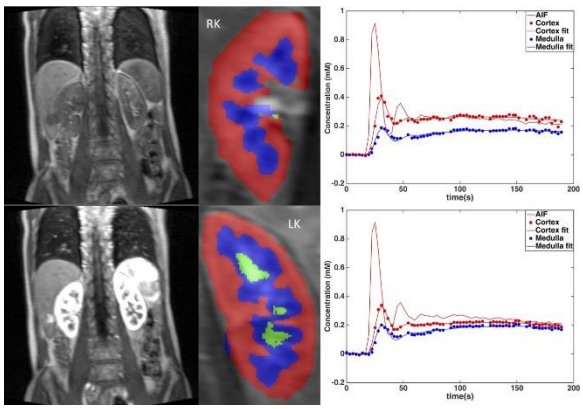
INTRODUCTION:



Chronic kidney disease, also called chronic kidney failure, describes the gradual loss of kidney function. Your kidneys filter wastes and excess fluids from your blood, which are then excreted in your urine. When chronic kidney disease reaches an advanced stage, dangerous levels of fluid, electrolytes and wastes can build up in your body

Chronic Kidney Disease or CKD is one of the most prevalent disease which influence humans on a larger scale and proves to be fatal as it remains dormant unless irreversible damages have been made to the kidney of an individual. Progression of CKD is related to variety of great complications, including increased incidence of various disorders, anemia, hyperlipidemia, nerve damage, pregnancy complications and even complete kidney failure. Millions of people die from this disease every year. Diagnosing CKD is a cumbersome task as there are no major symptom that can be used as a benchmark to detect the disease. In cases when diagnosis persists, some results may be interpreted incorrectly.

Technological development, including machine learning, has a huge impact on health through an effective analysis of various chronic diseases for more accurate diagnosis and successful treatment. In the field of biomedical and healthcare communities the accurate prediction plays the major role to find out the risk of the disease in the patient. The only way to overcome with the mortality due to chronic diseases is to predict it earlier so that the disease prevention can be done. Such model is a Patient's need in which Machine Learning is highly recommendable



It has various physical symptoms such as tiredness, poor appetite, cramping, swollen feets and ankles, etc. but we cannot detect the disease effectively based on only these symptoms. To effectively detect and classify this disease we have to take certain minute and particular symptoms which could help us in determining the disease with high accuracy and these are as follows such as 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, diabetes mellitus, coronary artery disease, appetite, pedal edema, anemia and last one is class.

These twenty four are the particular symptoms that I will be taking in order to detect the chronic kidney disease accurately and the last value that is a class is not a symptom, it is just used a variable which help us to verify either a patient is suffering from chronic kidney disease or not. These symptoms are can also be viewed in tabular format with their short forms which can be used easily at the time of making data sets of the readings of various patients.

Objective

The objective of our study is to develop a screening tool based on disease history using various ML models to predict upcoming RRT at the time of CKD diagnosis, and to validate these models. A further goal is to identify what comorbidities (such as diabetes) play major role in the models that forecast the onset of RRT for different time periods. This tool may serve both medical doctors and CKD patients in better healthcare planning and management of resources.

CHAPTER 2

METHODOLOGY:

Here we are using google collab to make our project. In this software we can code any python code using the required libraries. As mentioned in the code we have imported some packages such as numpy, panda etc to make this project.

We have collected some data of the patients to predict that they have this kideny disease or not. Now we created a dataset to import our data into google collab using csv command.In that data set we have several voice measurement frequencies of the patients to recognize the diseased one.

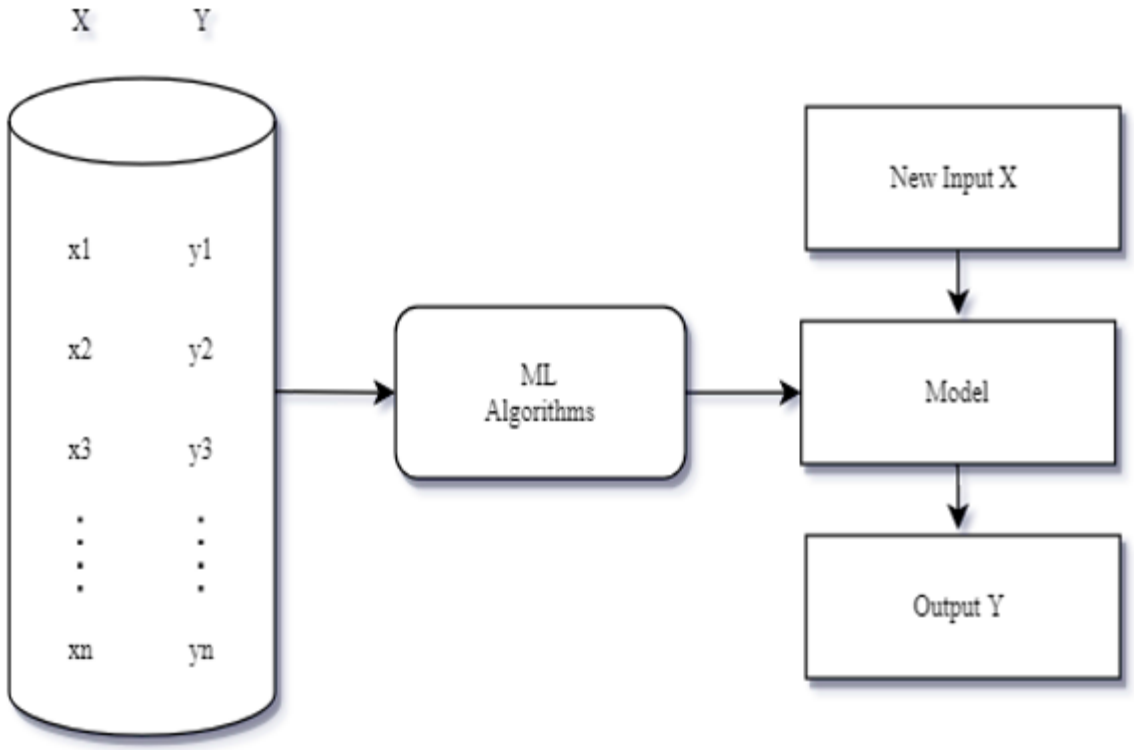
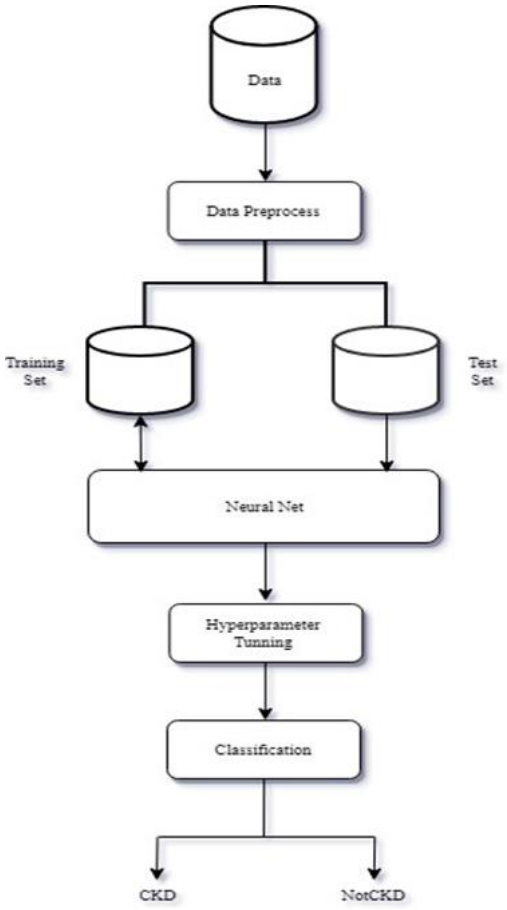


Fig 3. Supervised Learning



The below flow chart indicates how we are detecting this Chronic kideny disease using Python and other machine learning concepts

The model is trained using the principles of Supervised Machine Learning [9] where for given value of X i.e., input values (x1, x2, x3, …, xn) there is a corresponding Y value i.e., target or output value. As shown in the figure, The data is fed into the model where X comprises various features such as blood features such as blood pressure, age, sugar, etc. and Y is the target class that consists of binaries values i.e., affected by CKD or not. The model uses batch learning or offline learning mechanism for training, in which the training data is provided to the ANN model in batches, and then validated and tested. The dataset used in the proposed model is collected from Apollo Hospitals in Managiri and Karaikudi, Tamil Nadu, India. It contains some 25 attributes and a site, which is a branch of Google LLC, and this site is an Online machine learning community experts and data scientists, allowing users to publish and find data sets of various problems called Kaggle.

```
Data Set Column Information:age  -      age
bp     -      blood pressure
sg     -      specific gravity
al     -       albumin
su     -      sugar
rbc    -      red blood cells
pc     -      pus cell
pcc    -      pus cell clumps
ba     -      bacteria
bgr    -      blood glucose random
bu     -      blood urea
sc     -      serum creatinine
sod    -      sodium
pot    -      potassium
hemo   -      hemoglobin
pcv    -      packed cell volume
wc     -      white blood cell count
rc     -      red blood cell count
htn    -      hypertension
dm     -      diabetes mellitus
cad    -      coronary artery disease
appet  -      appetite
pe     -      pedal edema
ane    -      anemia
class  -      classification
```

Import the libraries

```
#Import Libraries
import glob
from keras.models import Sequential, load_model
import numpy as np
import pandas as pd
import keras as k
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
import matplotlib.pyplot as plt
  #load the data
  from google.colab import files #Only use for Google Colab
  uploaded = files.upload()     #Only use for Google Colab
  df = pd.read_csv("kidney_disease.csv")
```

```
#Print the first 5 rows
df.head()
```

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	sc	sod	pot	hemo	pcv	wc
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	121.0	36.0	1.2	NaN	NaN	15.4	44	7800
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	NaN	18.0	0.8	NaN	NaN	11.3	38	6000
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	423.0	53.0	1.8	NaN	NaN	9.6	31	7500
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	117.0	56.0	3.8	111.0	2.5	11.2	32	6700
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	106.0	26.0	1.4	NaN	NaN	11.6	35	7300

A sample of the data set

Get the number of rows and columns in the data set. Remember each row represents a patient and each column is a data point on that patient.

```
#Get the shape of the data (the number of rows & columns)
df.shape
```

Data Manipulation: Clean The Data

Now we will transform the data. By getting rid of missing data and removing some columns. First we will create a list of column names that we want to keep or retain.

Next we drop or remove all columns except for the columns that we want to retain.

Finally we drop or remove the rows that have missing values from the data set.

```
#Create a list of columns to retain
columns_to_retain = ["sg", "al", "sc", "hemo",
                     "pcv", "wbcc", "rbcc", "htn", "classification"]

#columns_to_retain = df.columns, Drop the columns that are not in columns_to_retain
df = df.drop([col for col in df.columns if not col in columns_to_retain], axis=1)

# Drop the rows with na or missing values
df = df.dropna(axis=0)
```

Let's loop through all of the columns and find the columns that do not contain number values. For those columns we will transform the values into numeric data.

```
#Transform non-numeric columns into numerical columns
for column in df.columns:
    if df[column].dtype == np.number:
        continue
    df[column] = LabelEncoder().fit_transform(df[column])
```

We will print the first 5 rows of the new data set.

```
df.head()
```

	sg	al	sc	hemo	pcv	htn	classification
0	1.020	1.0	1.2	15.4	28	1	0
1	1.020	4.0	0.8	11.3	22	0	0
2	1.010	2.0	1.8	9.6	15	0	0
3	1.005	4.0	3.8	11.2	16	1	0
4	1.010	2.0	1.4	11.6	19	0	0

Sample of the first 5 rows of new data set

Data Manipulation: Split & Scale The Data

Let's split the data set into a independent data set that we will call X which is the feature data set and a dependent data set that we will call y which is the target data set.

```
#Split the data
X = df.drop(["classification"], axis=1)
y = df["classification"]
```

Next we will scale the feature data set to be values between 0 and 1 inclusively.

```
#Feature Scaling
x_scaler = MinMaxScaler()
x_scaler.fit(X)
```

```
column_names = X.columns
X[column_names] = x_scaler.transform(X)
```

Once we are done with all of that, we will split the data sets into 80% training (X_train and y_train) and 20% testing (X_test and y_test) data sets, and shuffle the data before training.

```
#Split the data into 80% training and 20% testing
X_train,  X_test, y_train, y_test = train_test_split(
      X, y, test_size= 0.2, shuffle=True)
```

Build The Model (Artificial Neural Network):

We are ready to build the model also known as the Artificial Neural Network ! First we must create the models architecture, then we will add 2 layers, the first layer with 256 neurons and the '**ReLu**' activation function with a normal distribution initializer for the weights. Since that layer is the first layer we must also specify the number of features/columns in the data set len(X.columns).

The second layer which happens to be the last layer as well, will have 1 neuron and use the '**hard_sigmoid**' activation function.

```
#Build The model

model = Sequential()model.add(Dense(256, input_dim=len(X.columns),
kernel_initializer=k.initializers.random_normal(seed=13),
activation="relu"))model.add(Dense(1, activation="hard_sigmoid"))
```

Compile the model, and give it the loss function called '**binary_crossentropy**' which is a loss function used for binary classification, it measures how well the model did on training and then tries to improve on it using the optimizer.

The optimizer that we will give it is called the '**adam**' optimizer. We also want to see how well the model does, so we will get some metrics on the models accuracy.

```
#Compile the model
model.compile(loss='binary_crossentropy',
              optimizer='adam', metrics=['accuracy'])
```

Train the model using the training data sets (X_train and y_train). Give it 2000 epcochs and a batch size equal to the number of patients/rows in the data set.

Batch: Total number of training examples present in a single batch

Epoch:The number of iterations when an ENTIRE dataset is passed forward and backward through the neural network only ONCE.

Fit: *Another word for train*

```
#Train the model
history = model.fit(X_train, y_train,
                epochs=2000,
                batch_size=X_train.shape[0])
```

```
Epoch 1988/2000
229/229 [==============================] - 0s 20us/step - loss: 0.0087 - acc: 0.9956
Epoch 1989/2000
229/229 [==============================] - 0s 24us/step - loss: 0.0087 - acc: 0.9956
Epoch 1990/2000
229/229 [==============================] - 0s 22us/step - loss: 0.0088 - acc: 0.9956
Epoch 1991/2000
229/229 [==============================] - 0s 23us/step - loss: 0.0087 - acc: 0.9956
Epoch 1992/2000
229/229 [==============================] - 0s 20us/step - loss: 0.0087 - acc: 0.9956
Epoch 1993/2000
229/229 [==============================] - 0s 20us/step - loss: 0.0087 - acc: 0.9956
Epoch 1994/2000
229/229 [==============================] - 0s 18us/step - loss: 0.0087 - acc: 0.9956
Epoch 1995/2000
229/229 [==============================] - 0s 22us/step - loss: 0.0087 - acc: 0.9956
Epoch 1996/2000
229/229 [==============================] - 0s 29us/step - loss: 0.0087 - acc: 0.9956
Epoch 1997/2000
229/229 [==============================] - 0s 23us/step - loss: 0.0087 - acc: 0.9956
Epoch 1998/2000
229/229 [==============================] - 0s 20us/step - loss: 0.0087 - acc: 0.9956
Epoch 1999/2000
229/229 [==============================] - 0s 17us/step - loss: 0.0087 - acc: 0.9956
Epoch 2000/2000
229/229 [==============================] - 0s 17us/step - loss: 0.0087 - acc: 0.9956
```

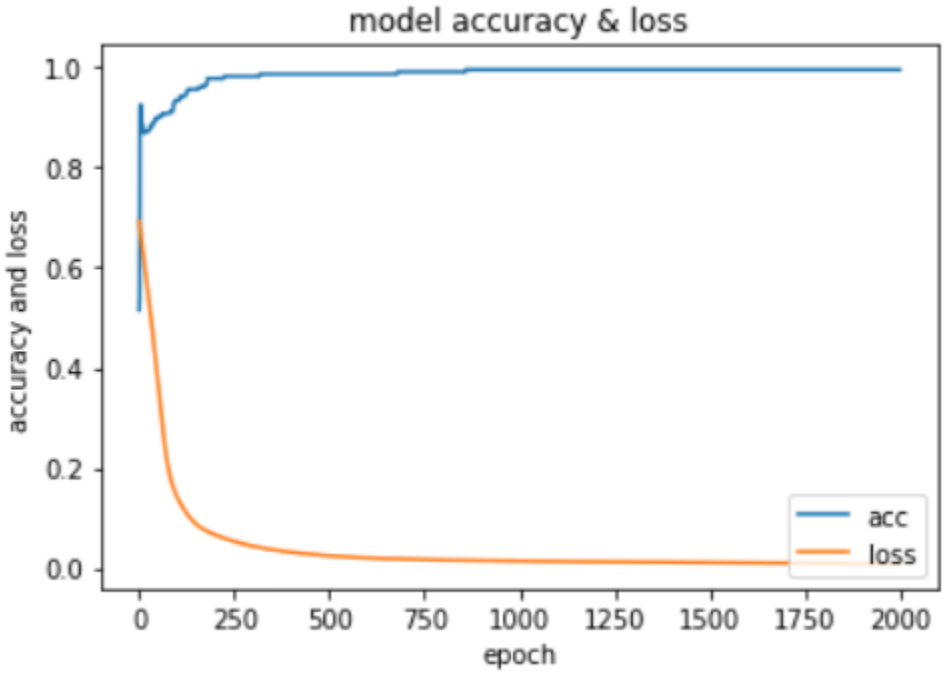
A sample of the training with the models accuracy = 99.56% and loss= .0087

Now that we are done creating our model. Let's save it.

```
#Save the model
model.save("ckd.model")
```

Visualize how well the model did on the training data set by plotting the loss and accuracy of the model.

```
#Visualize the models accuracy and loss
plt.plot(history.history["acc"])
plt.plot(history.history["loss"])
plt.title("model accuracy & loss")
plt.ylabel("accuracy and loss")
plt.xlabel("epoch")
plt.legend(['acc', 'loss'], loc='lower right')
plt.show()
```



The models loss (orange) & accuracy (blue)

Get the training and test data shape

```
print("-----------------------------------------------------")
print("Shape of training data: ", X_train.shape)
print("Shape of test data    : ", X_test.shape )
print("-----------------------------------------------------")
```

```
---------------------------------------------------------------------
Shape of training data:  (229, 6)
Shape of test data    :  (58, 6)
---------------------------------------------------------------------
```

Shape of training and testing data

Loop through any and all saved models. Then get each models accuracy, loss, prediction and original values on the test data.

```
for model_file in glob.glob("*.model"):
  print("Model file: ", model_file)
  model = load_model(model_file)
  pred = model.predict(X_test)
  pred = [1 if y>=0.5 else 0 for y in pred] #Threshold, transforming probabilities to either 0 or 1 depending if the probability is below or above 0.5
  scores = model.evaluate(X_test, y_test)
```

```
print()
print("Original  : {0}".format(", ".join([str(x) for x in y_test])))
print()
print("Predicted : {0}".format(", ".join([str(x) for x in pred])))
print()
print("Scores    : loss = ", scores[0], " acc = ", scores[1])
print("----------------------------------------------------------------")
print()
```

```
Model file:  ckd.model
58/58 [==============================] - 0s 3ms/step

Original  : 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0,
1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1

Predicted : 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0,
1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1

Scores    : loss =  0.010315061514747554  acc =  1.0
----------------------------------------------------------------
```

CHAPTER *3*

ANALYSIS AND INFERENCE:

Chronic kidney disease affects 8% to 16% of the population worldwide and is a leading cause of death. Optimal management of CKD includes cardiovascular risk reduction, treatment of albuminuria, avoidance of potential nephrotoxins, and adjustments to drug dosing. Patients also require monitoring for complications of CKD, such as hyperkalemia, metabolic acidosis, anemia, and other metabolic abnormalities. Diagnosis, staging, and appropriate referral of CKD by primary care clinicians are important in reducing the burden of CKD worldwide. And also we analysed how machine learning and the python contributes to biotechnology. We were able to create data sets by taking the reference of this model.

CHAPTER 4

CONCLUSION:

Chronic kidney disease affects 8% to 16% of the population worldwide and is a leading cause of death. Optimal management of CKD includes cardiovascular risk reduction, treatment of albuminuria, avoidance of potential nephrotoxins, and adjustments to drug dosing. Patients also require monitoring for complications of CKD, such as hyperkalemia, metabolic acidosis, anemia, and other metabolic abnormalities. Diagnosis, staging, and appropriate referral of CKD by primary care clinicians are important in reducing the burden of CKD worldwide.

In this Python machine learning project, we learned to detect the presence of Chronic kidney Disease in individuals using various factors. We used the sklearn library to prepare the dataset. This gives us an accuracy of 94.87%, which is great considering the number of lines of code in this python project.

This will help in a medical field for easy and early detection of chronic disease.

FUTURE SCOPE

There are other possible evolutionary techniques that may be used to improve results of the proposed classifiers. We can also evaluate and compare the performance of the used classifiers with other existing classifiers. CKD early detection helps in timely treatment of the patients suffering from the disease and also to avoid the disease from getting worse. Early prediction of the disease and timely treatment are the need for medical sector. New classifiers can be used and their performance can be evaluated to find better solutions of the objective function in future work.

There is a possibility of introducing the trending research area called image processing in this detection in future which is highly advanced and there is possibility of easy detection by comparing images .

References

http://www.ijetajournal.org/volume-3/issue-4/IJETA-V3I4P4.pdf

https://www.kaggle.com/mansoordaku/ckdisease

Chronic Kidney Disease Diagnosis and Treatment 2020 Edition by Junwei Yang, Weichun He , Springer

Python: The Complete Reference

APPENDIX

```
#Import Libraries
import glob
from keras.models import Sequential, load_model
import numpy as np
import pandas as pd
import keras as k
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
import matplotlib.pyplot as plt
 #load the data
    from google.colab import files #Only use for Google Colab
    uploaded = files.upload()      #Only use for Google Colab
    df = pd.read_csv("kidney_disease.csv")

    #Print the first 5 rows
    df.head()

#Get the shape of the data (the number of rows & columns)
df.shape
#Create a list of columns to retain
columns_to_retain = ["sg", "al", "sc", "hemo",
                         "pcv", "wbcc", "rbcc", "htn", "classification"
]

#columns_to_retain = df.columns, Drop the columns that are not in colum
ns_to_retain
df = df.drop([col for col in df.columns if not col in columns_to_retain
], axis=1)

# Drop the rows with na or missing values
df = df.dropna(axis=0)
#Transform non-numeric columns into numerical columns
for column in df.columns:
        if df[column].dtype == np.number:
            continue
        df[column] = LabelEncoder().fit_transform(df[column])
df.head()
df.shape
```

```
#Create a list of columns to retain
columns_to_retain = ["sg", "al", "sc", "hemo",
                          "pcv", "wbcc", "rbcc", "htn", "classification"
]

#columns_to_retain = df.columns, Drop the columns that are not in colum
ns_to_retain
df = df.drop([col for col in df.columns if not col in columns_to_retain
], axis=1)

# Drop the rows with na or missing values
df = df.dropna(axis=0)
#Transform non-numeric columns into numerical columns
for column in df.columns:
        if df[column].dtype == np.number:
            continue
        df[column] = LabelEncoder().fit_transform(df[column])
df.head()
df.shape
#Split the data
X = df.drop(["classification"], axis=1)
y = df["classification"]
#Feature Scaling
x_scaler = MinMaxScaler()
x_scaler.fit(X)
column_names = X.columns
X[column_names] = x_scaler.transform(X)
#Split the data into 80% training and 20% testing
X_train,  X_test, y_train, y_test = train_test_split(
        X, y, test_size= 0.2, shuffle=True)
#Build The model

model = Sequential()
model.add(Dense(256, input_dim=len(X.columns),              kernel_init
ializer=k.initializers.random_normal(seed=13), activation="relu"))
model.add(Dense(1, activation="hard_sigmoid"))
#Compile the model
model.compile(loss='binary_crossentropy',
                  optimizer='adam', metrics=['accuracy'])
#Train the model
history = model.fit(X_train, y_train,
                    epochs=2000,
                    batch_size=X_train.shape[0])
#Save the model
model.save("ckd.model")
#Visualize the loss and accuracy of the model
plt.plot(history.history['accuracy'])
plt.plot(history.history['loss'])
```

```
plt.title('model accuracy and loss')
plt.ylabel('accuracy and loss')
plt.xlabel('epoch')
print("------------------------------------------------------------")
print("Shape of training data: ", X_train.shape)
print("Shape of test data    : ", X_test.shape )
print("------------------------------------------------------------")
for model_file in glob.glob("*.model"):
  print("Model file: ", model_file)
  model = load_model(model_file)
  pred = model.predict(X_test)
  pred = [1 if y>=0.5 else 0 for y in pred] #Threshold, transforming pr
obabilities to either 0 or 1 depending if the probability is below or a
bove 0.5
  scores = model.evaluate(X_test, y_test)
  print()
  print("Original  : {0}".format(", ".join([str(x) for x in y_test])))
  print()
  print("Predicted : {0}".format(", ".join([str(x) for x in pred])))
  print()
  print("Scores    : loss = ", scores[0], " acc = ", scores[1])
  print("------------------------------------------------------------")
  print()
```