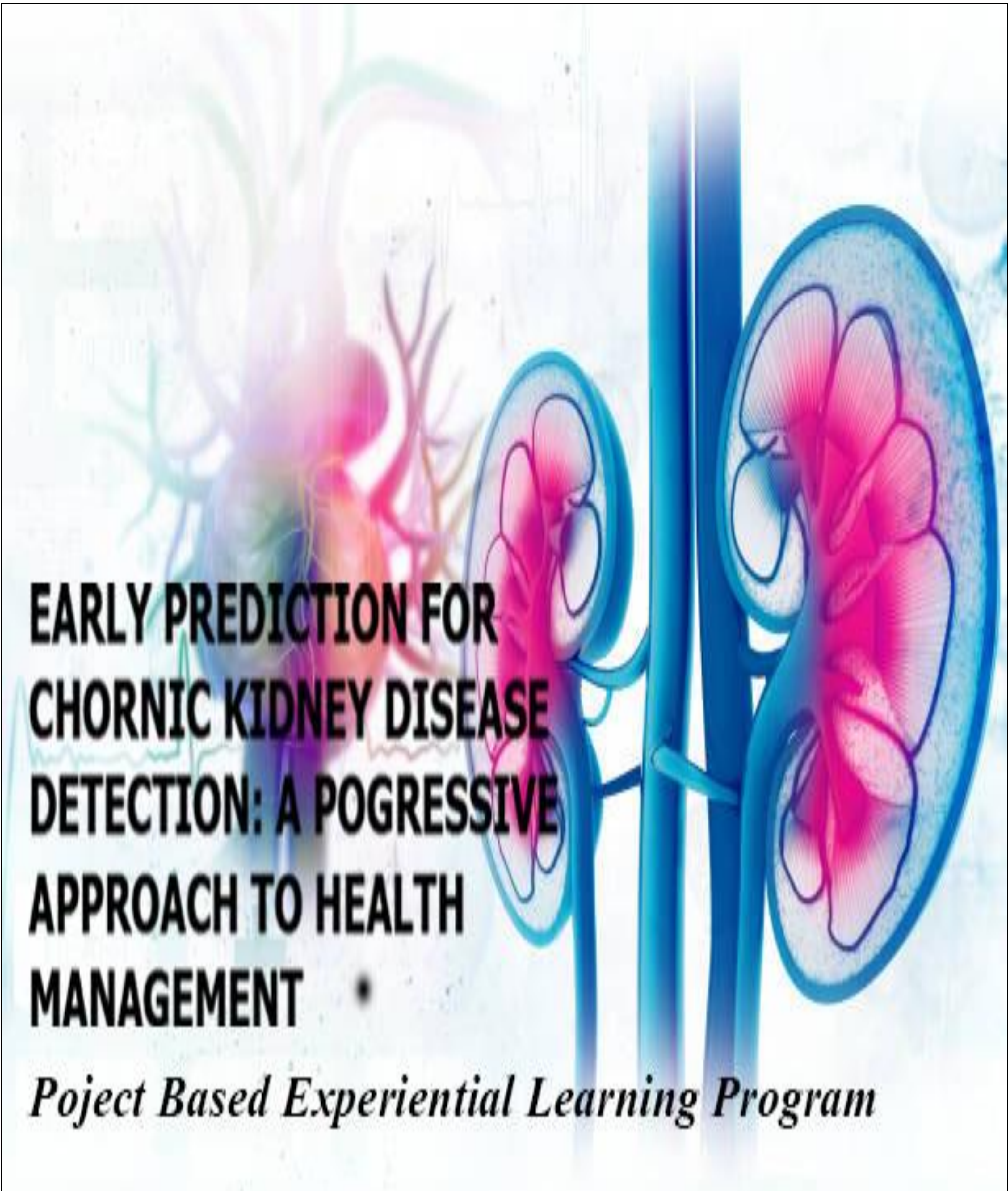


# EARLY PREDICTION FOR CHORNIC KIDNEY DISEASE DETECTION: A POGRESSIVE APPROACH TO HEALTH MANAGEMENT

*Poject Based Experiential Learning Program*

**THIRUVALLUVAR GOVERNMENT ARTS COLLEGE**

**RASIPURAM – 637 408**

**Team Leader: Sathya J S (20UCS2323)**

**Team Member: Bhuvaneswari A (20UCS2317)**

**Team Member: Girija V (20UCS2318)**

**Team Member: Monasri R (20UCS2319)**

**Team Member: Sivajothi S (20UCS2324)**

# <u>Early Prediction for Chronic Kidney Disease:</u>

**Detection: A**

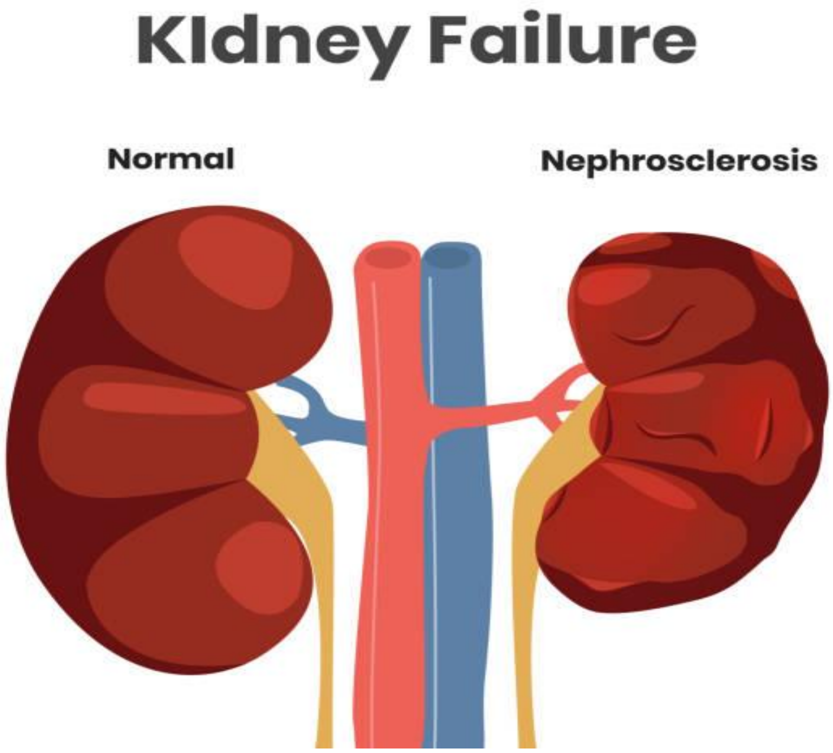
**Progressive Approach to Health Management**

**SPECIFY THE BUSINESS PROBLEM:**

**THE INTRODUCTION FOR CHRONIC KIDNEY DISEASE:**

CKD is a condition in which the kidneys are damagedand 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.



## <u>**CHORNIC KIDNEY DISEASE PREDICTION USING MACHINE LEARNING:**</u>

Predication is done using the machine learning tecgnique,SVM.

In this classification problem SVM classifies the output into two class with CKD and without CKD.

The main objective of this stuty was to predict patients with CKD using less number attributes while maintaining a higher accuracy**.**

## CHRONIC KIDNEY DISEASE PREDICTION:

Glomerular filtration rate(GFR)is the most accurate  test to dertermine your kidney function and the degree of chronic kidney disease.

Blood creatinine level,age,gender,and other characteristics can be used to calculate it.In most cases it is better to get sick early.

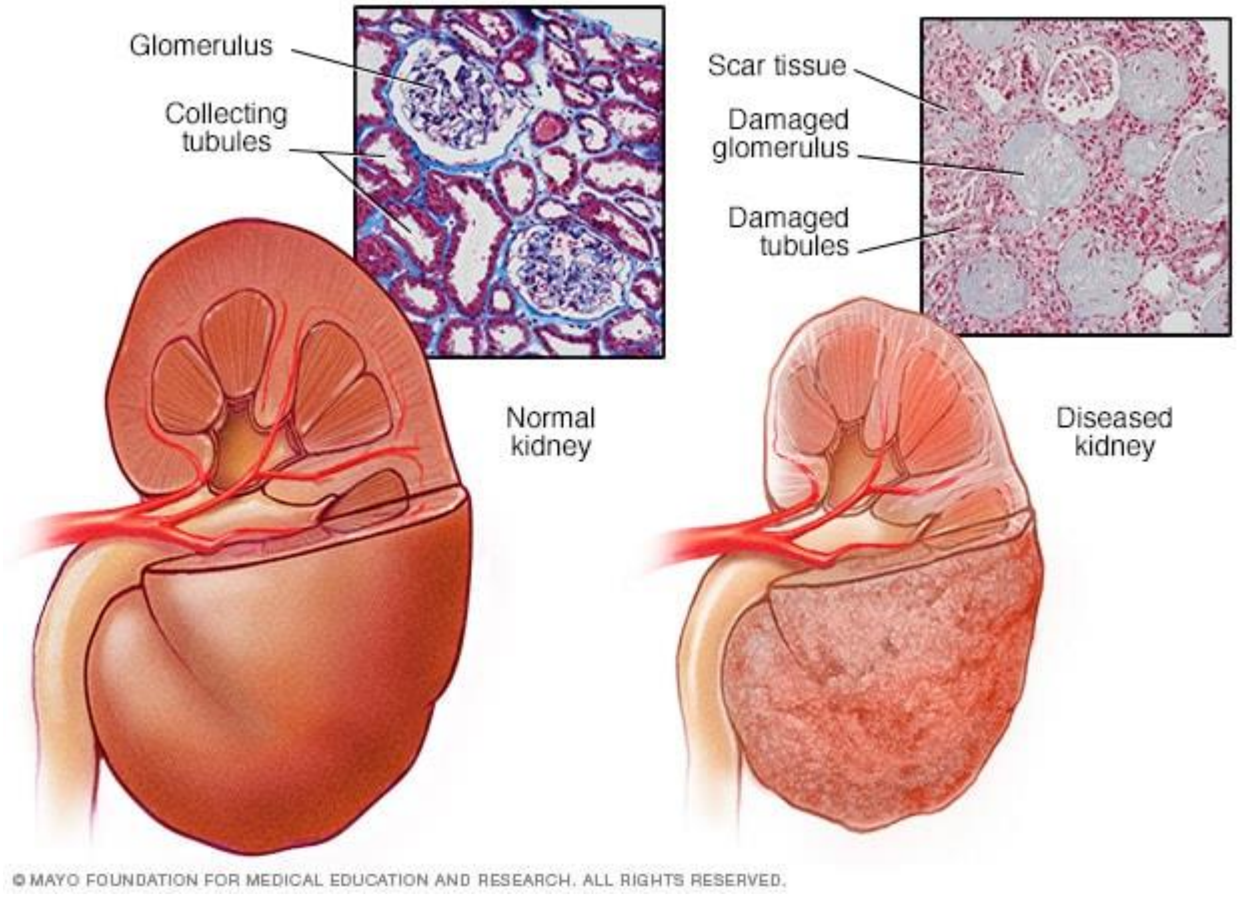
## OBJECTIVE OF CHRONIC KIDNEY DISEASE:

One of the objectives is to reduce premature mortaily from non-communication disease by third in 2030.

chronic kidney disease(CKD)is among the signifiant contributor to morbidity and mortality from non-communicable diseases that can affected 10-15% of the global population.

These strategies seek to:prevent and control risk facters for CKD.raise awareness of CKD and its complications.

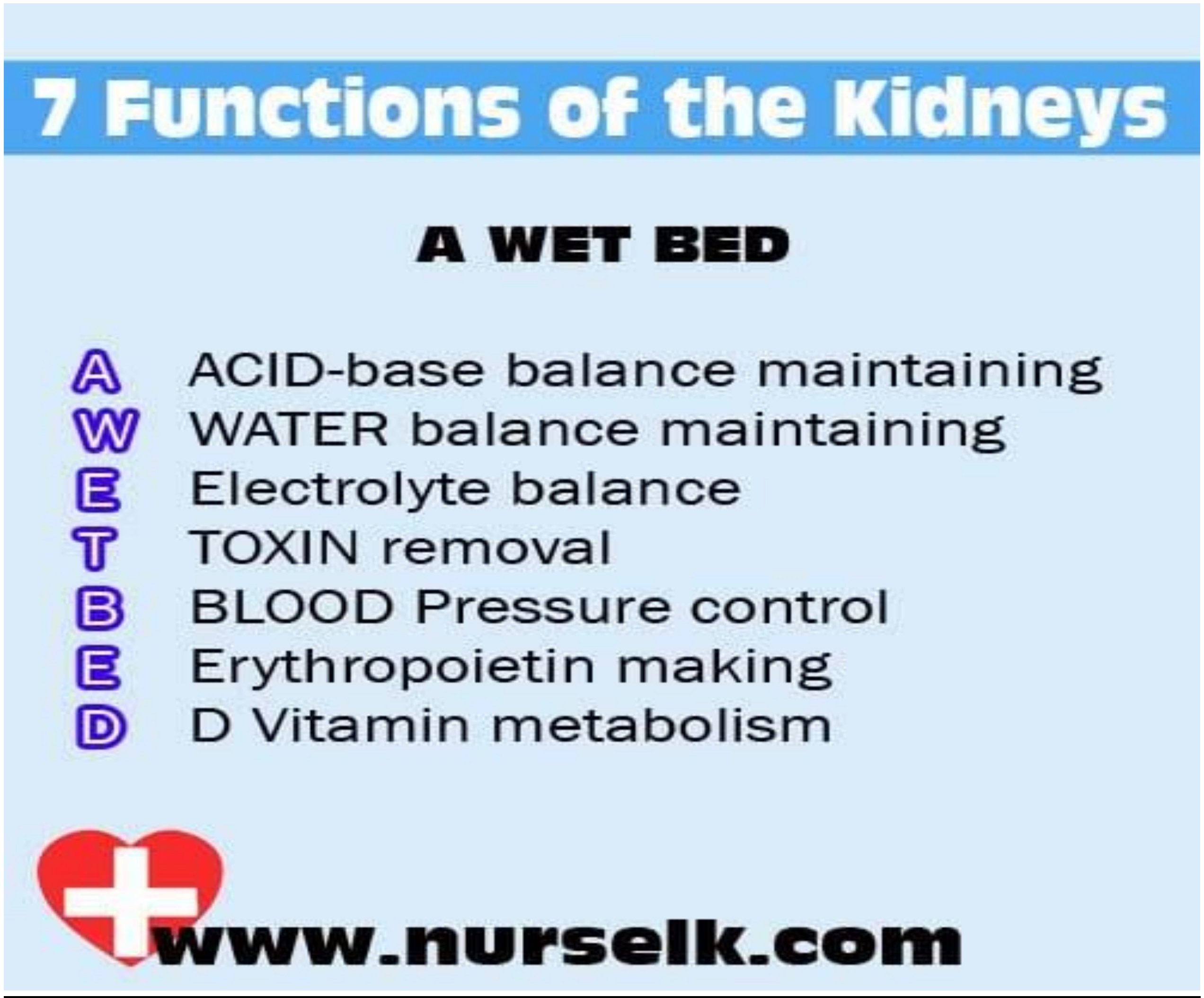
To detect the various diseases through the examining symptoms of patient's using different techniques of machine learning models.



## <u>7 MAIN FUNCTIONS OF THE KIDNEY:</u>

- Remove waste produts from the body.
- Remove drugs from the body.
- Balance the body's fluids.
- Release hormones that regulate blood pressure.
- Produce an active form of vitamin D that promotes strong,healthy bones.

- Control the production of red blood cells.



7 Functions of the Kidneys

A WET BED

| | |
|---|---|
| A | ACID-base balance maintaining |
| W | WATER balance maintaining |
| E | Electrolyte balance |
| T | TOXIN removal |
| B | BLOOD Pressure control |
| E | Erythropoietin making |
| D | D Vitamin metabolism |

www.nurselk.com

**<u>ROLE OF KIDNEY:</u>**

The main functions of the kidney are filtration and excretion of metabolic waste products from the bloodstream,redulation of electrolytes,acidity and blood cell production.

## **DETECT CHRONIC KIDNEY DISEASE EARLY:**

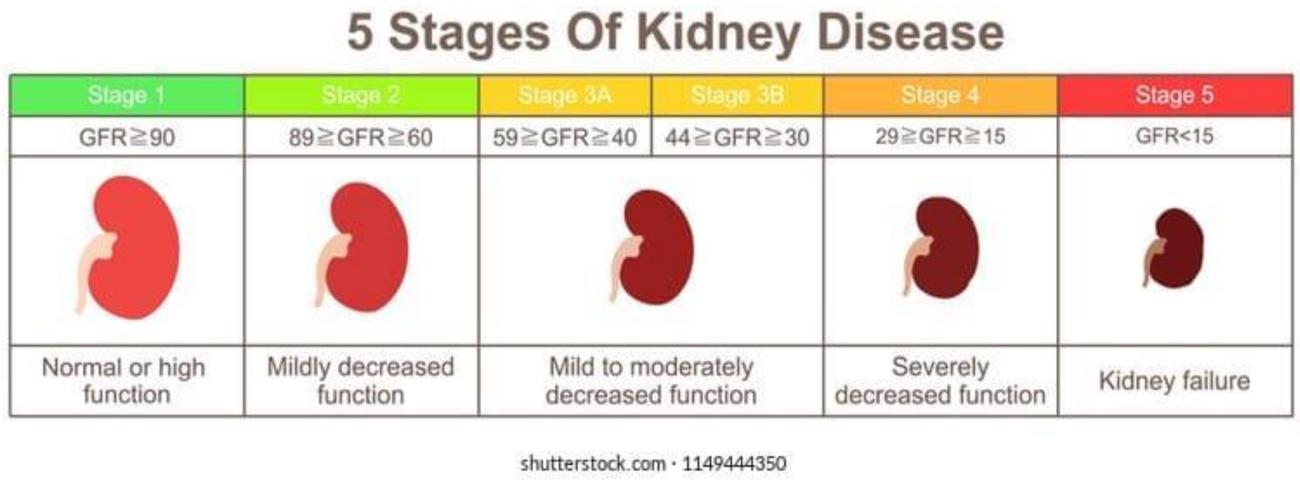
The only way to find out if people have CKD is through simple blood and urine tests.

The urine test checks for protein,which may indicate kidney damage.

## **STAGES OF CHRONIC KIDNEY DISEASE:**

- Stage 2 mild CKD(GFR=60-89ml/min)
- Stage 3A moderate CKD(GFR=45-59ml/min)

- Stage 3B moderate CKD(GFR=30-44ml/min)
- Stage 4 severe CKD(GFR=15-29ml/min)
- Stage 5 end stage CKD(GFR<15ml/min)



5 Stages Of Kidney Disease

| Stage 1 | Stage 2 | Stage 3A | Stage 3B | Stage 4 | Stage 5 |
|---|---|---|---|---|---|
| GFR≧90 | 89≧GFR≧60 | 59≧GFR≧40 | 44≧GFR≧30 | 29≧GFR≧15 | GFR<15 |
| Normal or high function | Mildly decreased function | Mild to moderately decreased function | | Severely decreased function | Kidney failure |

shutterstock.com · 1149444350

## CONCLUSION TO KIDNEY FUNCTIONS:

In conclusion the kidneys are a vital organ critical to the human body.

fromfiltering waste from blood to produce red blood cells,it serves a crucial role.

With cells andissue that work together in synchronized form for common function

## Business Requirements

**Topics:**

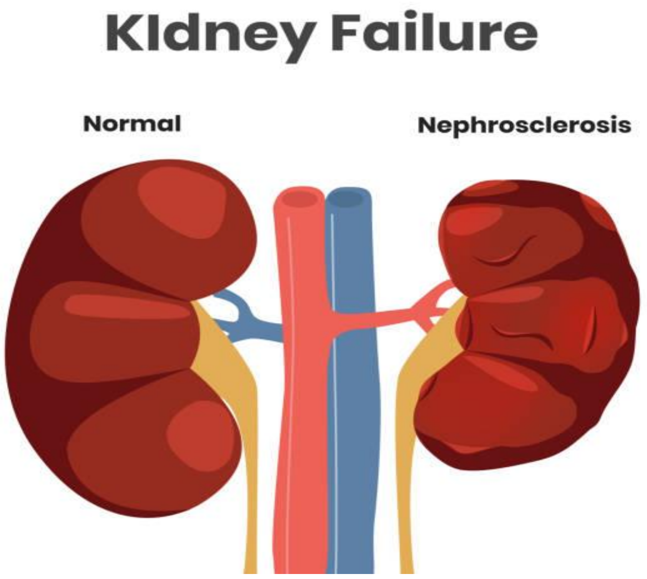
- ➢ The business requirements for a machine learning model chronic kidney diseas (ckd)
- ➢ predicting disease and not diseased
- ➢ Explanation for the machine learning model decision

## **1.The business requirements for a machine learning model to predict chronic kidney disease**

**Requirements:**

- ✓ Medical care center
- ✓ Nurses
- ✓ Laboratories
- ✓ Hospitals
- ✓ Lot environmentat patients sphere network
- ✓ Decision trees -Bio medical sensors

✓Wearable intelligent Devices
✓Personal area
✓Supoort vector machine
✓Multi layer perceptron
✓Bayes classifier



2.<u>Predicting   Disease</u>

<u>and not</u>

<u>Disease:</u>

**<u>CDK Symptoms:</u>**

*Changes in urination
*Fatigue

*Itching

*Swelling in your hands, legs, or feet

*Shortness of breath

*Pain in the small of your back

*Decreased appetite

*Puffiness around your eyes

*Abnormal levels of phosphorus, calcium, or vitamin D

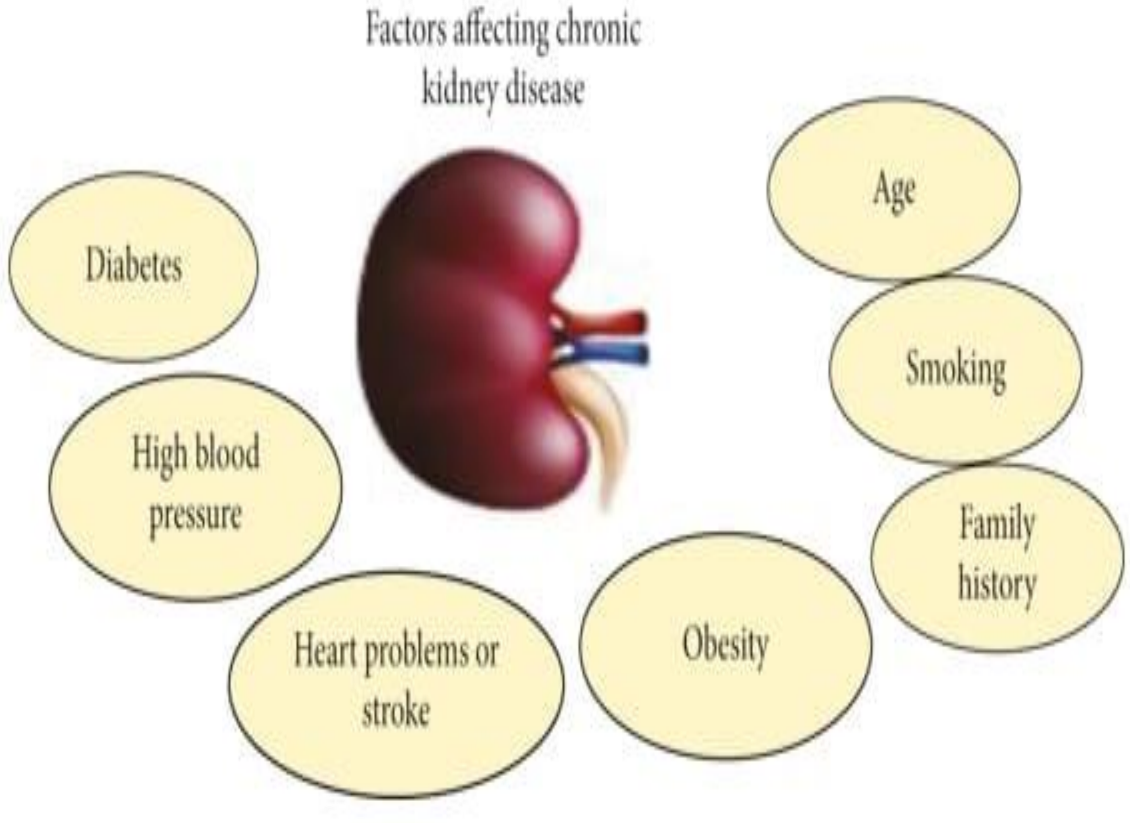
*Abnormal urine test

*High blood

pressure

<u>Not</u>

<u>Diseased</u>



<u>Sings of healthy kidney :</u>

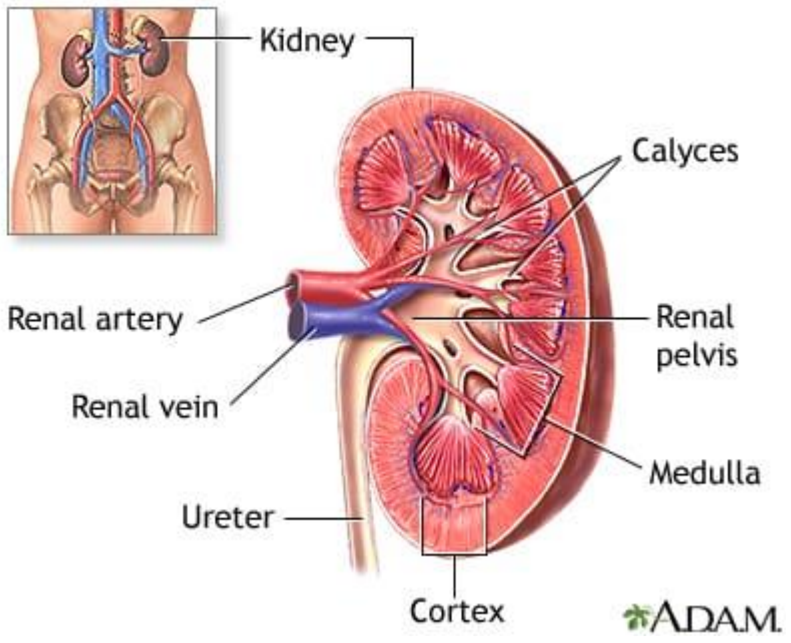
- No Changes in Frequency of Urination. One of the early signs of kidney disease is having to urinate more often.
- Healthy Urine. Healthy urine is directly related to healthy kidneys.…
- No Signs of Puffiness or Swelling...

-No Muscle Cramping…

-Your Skin is Clear and Healthy…

-Los Angeles Kidney Specialist.

-if you have a GFR number of 60 or more together with a normal urine alumbinium test, you are in the normal range.



## <u>3.Explanation for machine learning model decision</u>

-Early diagnosis arded by virtual parameter analytics using affordable computer aided diagnosis costs but improve <u>patient management and outcomes</u>

-Better and cheaper and more effective <u>screeping tools </u>can reduce the burden of

disease by identifying at -risk individuals at an early stage

-Traditionaly, ckd severity detection and to measure any are used <u>biomarkers</u> and

<u>computation</u> there are methods.

-Dialysis and transplantation treatments are needed However, early Adverse CKD through diagnosis and appropriate treatments. Reduce the effects.

-A hemoglobin level of less than 13g and a specific gravity equal to or less than

1.015 have high implications for the diagnosis of CKD(chronic kidney disease)

-Anemia working group (KDIGO)[43];less than

Hemoglobin level over 15 years of age for men, 12g

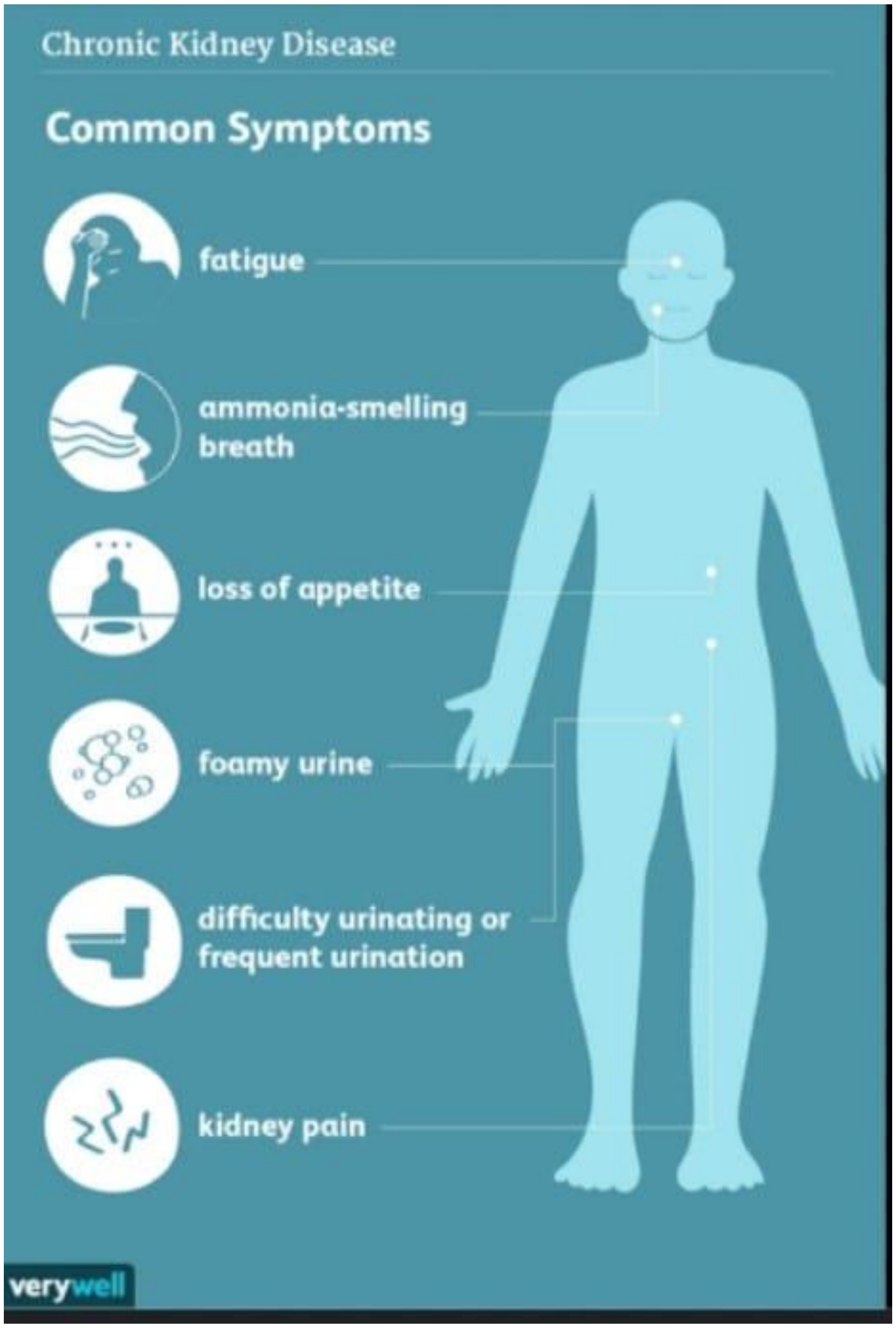
/dL Low hemoglobin levels over 15 years of age for

men,12g/dl low hemoglobin For 15 years olds with

size more women also blood due to diabetes.

affected,usually their kidneys At least half of the

process it is also lost informs.



## LITERATURE SURVEY :-

Delaying diseases progression and reducing the risk of mortality are key goals in treatment of chronic kidney diseases.

- This systematic literature review explored treatments evaluated in patience with CKD since 1990.

There is currently an unmet need for effective treatment for CKD that slows diseases progression prevents development.

## DATA SOURCES AND SEARCHES:

Using the terms listed in the supplementary materials, we searched **MEDLINE**, Embase and cochrane Library peers articles publshed between 1990 and November 2,2020 that reported results from prospective, parallel design randomized people in the world.

## ACCORDING TO CURRENT ESTIMATIONS:-

- CKD is more common people aged in 65 years.
- CKD is slightly more common in women 14% and men 12%

## MANAGE CKD:-

- Use medicines as directed to slow the decline in kidney functions.
- Stop smoking or do not starts smoking.

## CERTAIN MEDICINES:-

- Over the counter pain like ibuprofen and neproxen, which are also called non-steroidal anti inflammatory drugs.
- **SOME ANTIBIOTICS:-**
- Certain herbal supplements.
- Excessive alcohol intake.

**REFERENCES:-**

1. Bikbov B, et al. Global regional and national burden of CKD, 1990-2017:

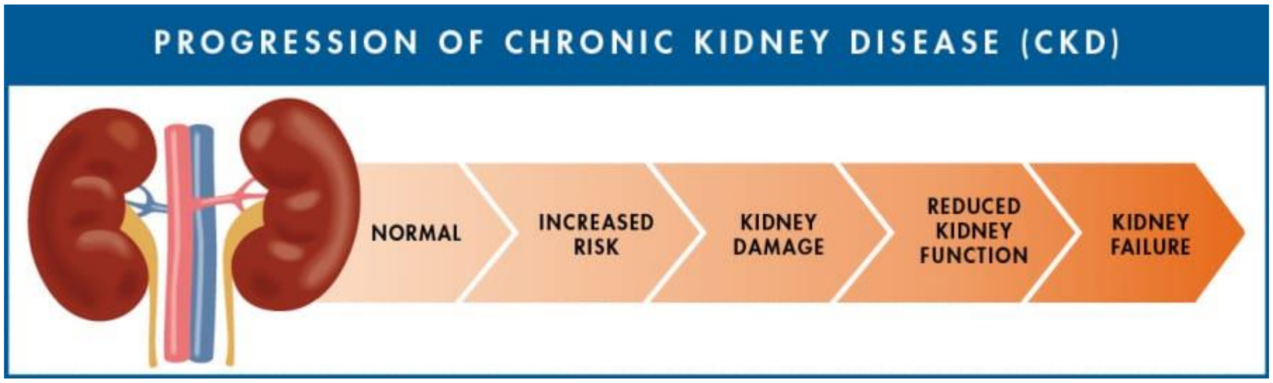
**SYSTEMATIC ANALYSIS FOR THE GLOBAL BURDEN OF DISEASE STUDY :-**

**Background :-**

- We estimate the global, regional and national burden of CKD, as well as the burden of cardoivascular disease and gout attributable to impaired kidney function, for the Global Burden of Diseases, Injuries and Risk Factors Study 2017.

**Methods :-**

- The main data sources we used were published literature, vital registration systems, end stages kidney diseases registries and household survey**lancet, 2012- Elsevier.**



PREVALENCE OF CHRONIC DISEASE IN CHINA:-

Background :

- The prevalence of CKD is high in developing . countries
- We aimed to measure the prevalence of CKD in China with such a survey.

**Methods :-**

- We did a cross-sectional survey of nationally representative sample of Chinese adults.

3. **Robert G Nelson, Morgan E Grams, Shoshanna H Ballew, Yingving Sang.**

**DEVELOPMENT OF RISK PREDICTION EQUATIONS FOR INCIDENT CKD DEVELOPMENT OF RISK PREDICTIOIN EQUATIONS FOR INCIDENT CKD :-**

a. Early identification of individuals at elavated risk of developing CKD could improve clinic through enhanced surveillance and better management and underlying health conditions

**SOCIAL IMPACT OR BUSINESS IMPOCT CKD:**

SOCIAL IMPACT FOR CKD:

Psychosocial factors including depression, anxiety and lower social support are common in patients with chronic kidney disease (CKD).

SOCIAL RISK FACTORS FOR CKD:

Food Insecurity :

A household-level experiencing an emotional or financial crisis, in which family members have uncertain or inadequate access to food

Housing Instability:

Due to either emotional or financial problems, family members face uncertainty, such as having trouble Paying rent, overcrowding, moving frequently, staying with relatives, or spending the bulk of household income on housing.

Unreliable Transportation :

_Unable to access affordable, convenient, reliable transportation, which leads to difficulties getting Places, including, school, grocery stores, medical appointments, and more.

Safety Fears:

__Concerns about domestic or neighbourhood violence, which includes victimization, witnessing violence and crime.

Inadequate Access To Utilities :

__Limited or unreliable access to water, electricity, phone, internet, WIFI, and gas for the household, including lack of refrigeration or plumbing.

IMPACT OF CKD IN QUALITY OF LIFE :

Introduction:

_Even though there is no statistically significant association between stages of CKD and it is decreased in patients with early stages of the disease.

Methods:

_A cross sectional study was conducted at the nephrology clinic of TASH. A total of 256 patients were recruited through systematic random sampling.

_ Student’s unpaired t-test and ANOVA were conducted to compare two groups and more than two groups in the analysis of QOL , respectively.

Results:

_whereas absence of CKD complications (β 2.75; 95%CI: 0.56–4.94, p = 0.014), high family income (β10.10; 95%CI: 5.10–15.10, p<0.001) and hemoglobin ≥11g/dl (β 4.54, 95%CI: 2.01–7.08, p = 0.001) were predictors of better QOL in the mental component summary.

Conclusion:

_In this setting, QoL decreased in CKD patients in the early stages of the disease.

BUSINESS IMPACT FOR CKD:

The Increasing Health &Economic Burden Of Kidney Disease:

*Approximately 26 million Americans have some evidence of chronic kidney disease and are at risk to develop kidney failure.

Another 20 million are at increased risk for developing kidney disease.

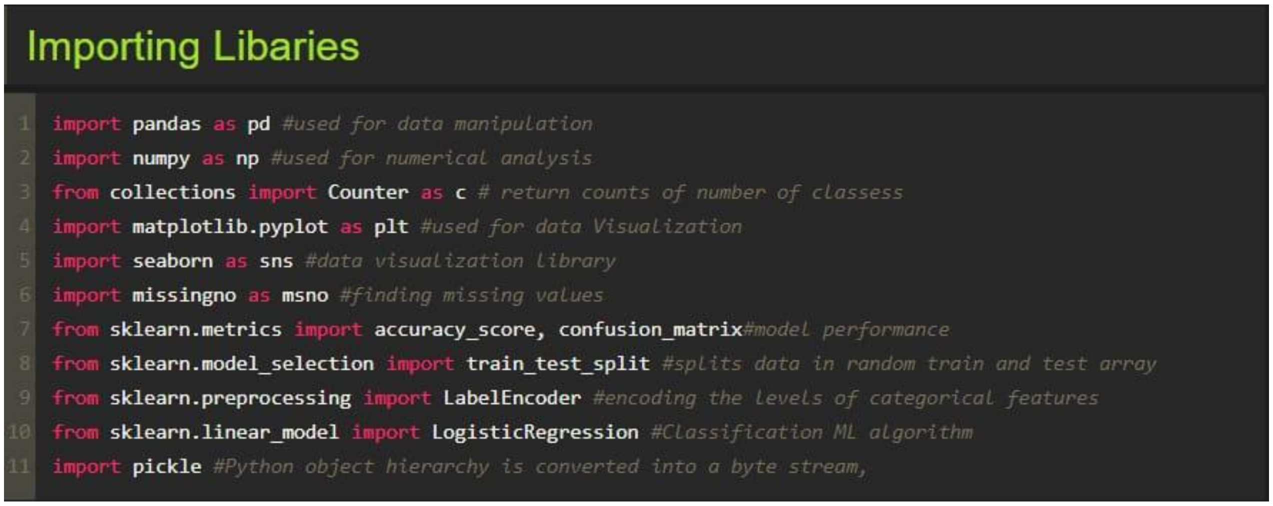
*When chronic kidney disease progresses, it may lead to kidney failure Or end stage renal disease (ESRD).

Currently, approximately 485,000 Americans have been diagnosed with kidney failure and require ongoing, expensive and life-altering treatments – such as frequent dialysis treatments or kidney transplantation.

More Research & Information about Kidney Disease is Needed:

*Primary care physicians and nephrologists need evidence-based guidelines to effectively treat early stage kidney disease and to help prevent its progression. Government guidance is needed on tracking data based on patient race and ethnicity ln order to quantify the considerable racial disparities in kidney disease incidence, and ensure that appropriate measures are designed.

**Data Collection :**
**Collect the dataset:**
There are many popular open sources for collecting the data. Eg:
kaggle.com, UCI Repository, etc.
Importing the libraries:

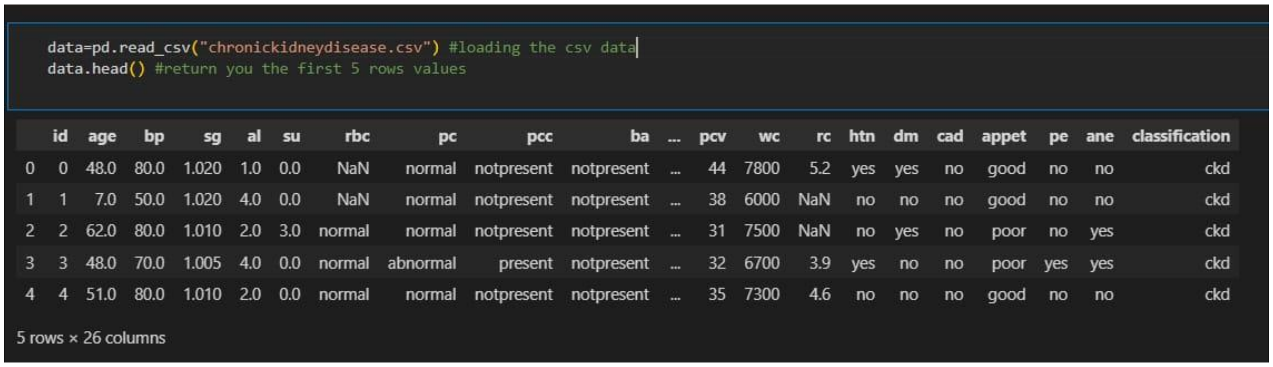


Importing Libaries

```python
import pandas as pd #used for data manipulation
import numpy as np #used for numerical analysis
from collections import Counter as c # return counts of number of classess
import matplotlib.pyplot as plt #used for data Visualization
import seaborn as sns #data visualization library
import missingno as msno #finding missing values
from sklearn.metrics import accuracy_score, confusion_matrix#model performance
from sklearn.model_selection import train_test_split #splits data in random train and test array
from sklearn.preprocessing import LabelEncoder #encoding the levels of categorical features
from sklearn.linear_model import LogisticRegression #Classification ML algorithm
import pickle #Python object hierarchy is converted into a byte stream,
```

**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



```python
data=pd.read_csv("chronickidneydisease.csv") #loading the csv data
data.head() #return you the first 5 rows values
```

| | id | age | bp | sg | al | su | rbc | pc | pcc | ba | ... | pcv | wc | rc | htn | dm | cad | appet | pe | ane | classification |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 48.0 | 80.0 | 1.020 | 1.0 | 0.0 | NaN | normal | notpresent | notpresent | ... | 44 | 7800 | 5.2 | yes | yes | no | good | no | no | ckd |
| 1 | 1 | 7.0 | 50.0 | 1.020 | 4.0 | 0.0 | NaN | normal | notpresent | notpresent | ... | 38 | 6000 | NaN | no | no | no | good | no | no | ckd |
| 2 | 2 | 62.0 | 80.0 | 1.010 | 2.0 | 3.0 | normal | normal | notpresent | notpresent | ... | 31 | 7500 | NaN | no | yes | no | poor | no | yes | ckd |
| 3 | 3 | 48.0 | 70.0 | 1.005 | 4.0 | 0.0 | normal | abnormal | present | notpresent | ... | 32 | 6700 | 3.9 | yes | no | no | poor | yes | yes | ckd |
| 4 | 4 | 51.0 | 80.0 | 1.010 | 2.0 | 0.0 | normal | normal | notpresent | notpresent | ... | 35 | 7300 | 4.6 | no | no | no | good | no | no | ckd |

5 rows × 26 columns

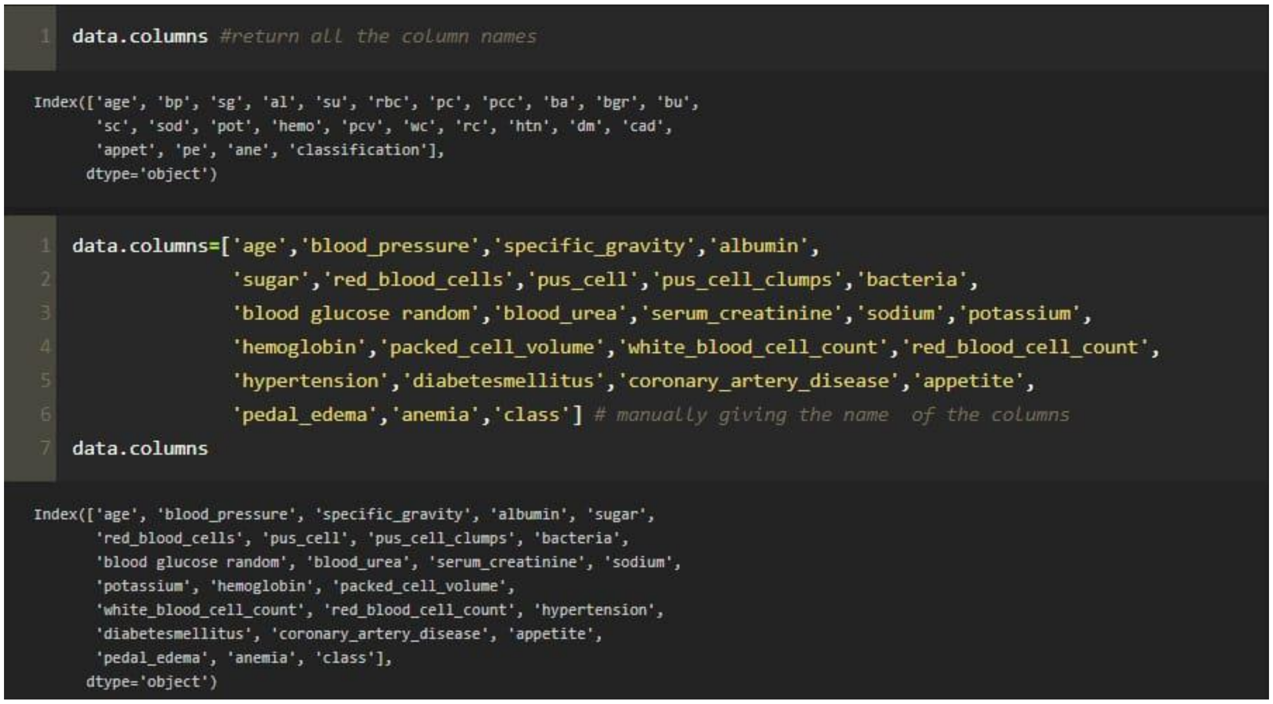
**Data Preparation**

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.

**Rename the columns**



```
data.columns #return all the column names
```

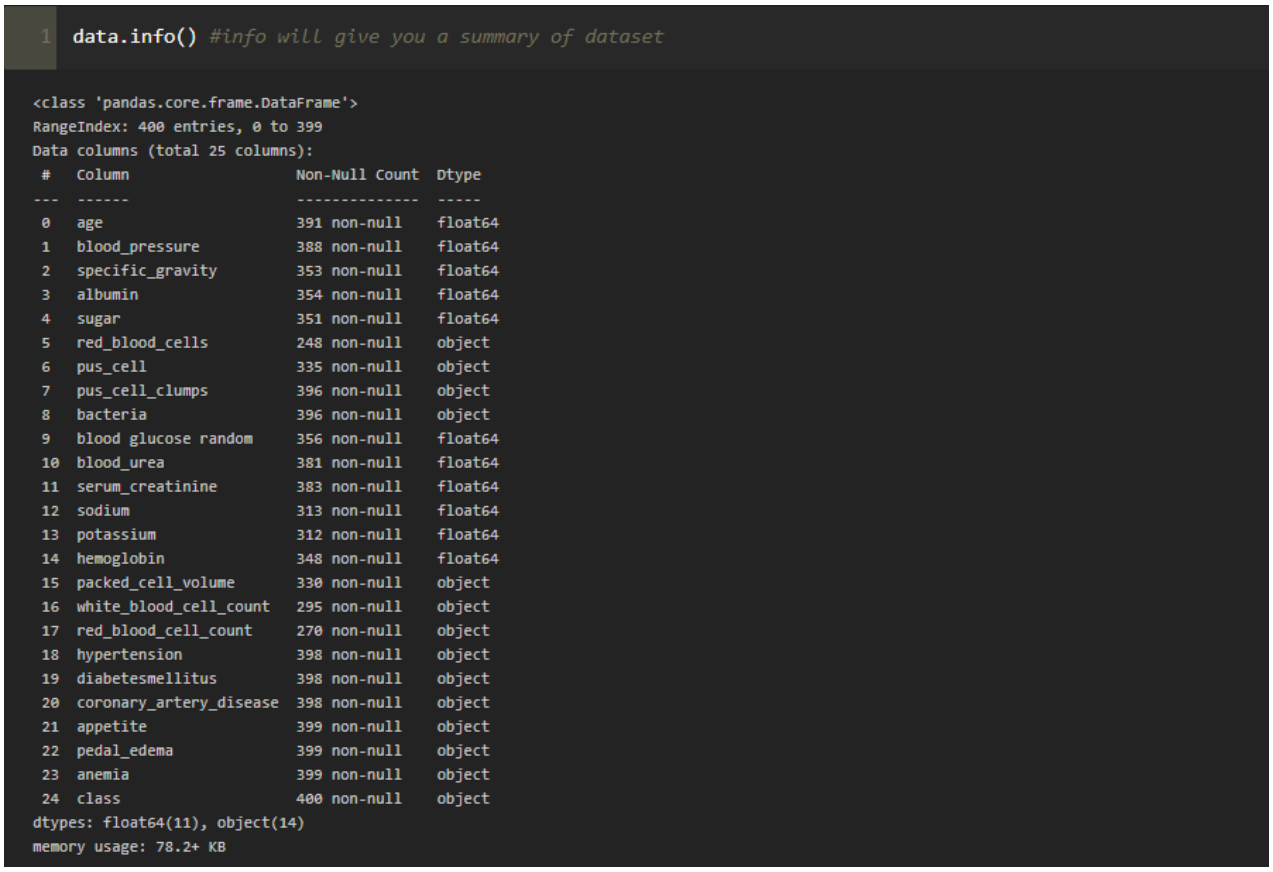
```
Index(['age', 'bp', 'sg', 'al', 'su', 'rbc', 'pc', 'pcc', 'ba', 'bgr', 'bu',
       'sc', 'sod', 'pot', 'hemo', 'pcv', 'wc', 'rc', 'htn', 'dm', 'cad',
       'appet', 'pe', 'ane', 'classification'],
      dtype='object')
```

```
data.columns=['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'] # manually giving the name  of the columns
data.columns
```

```
Index(['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'],
      dtype='object')
```

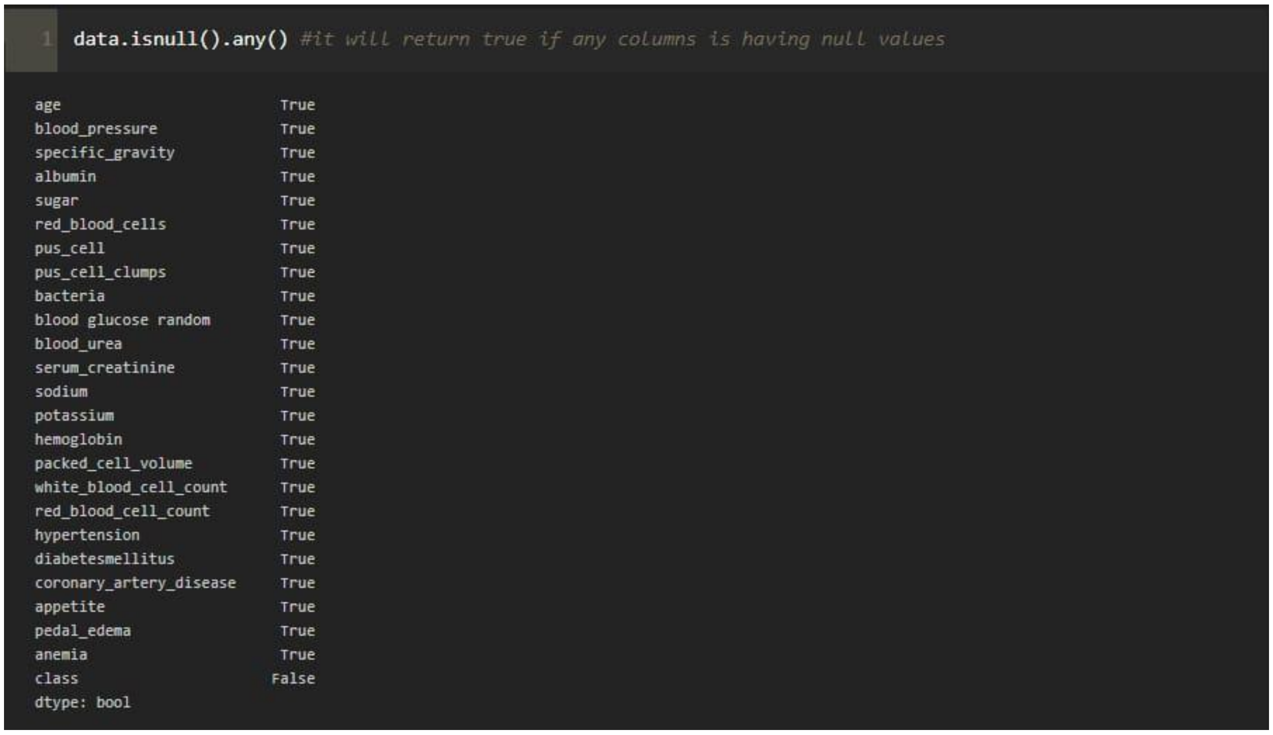
## Handling missing values

●
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.Let’s now check the count of null values after filling all null values using **isnull.sum()**



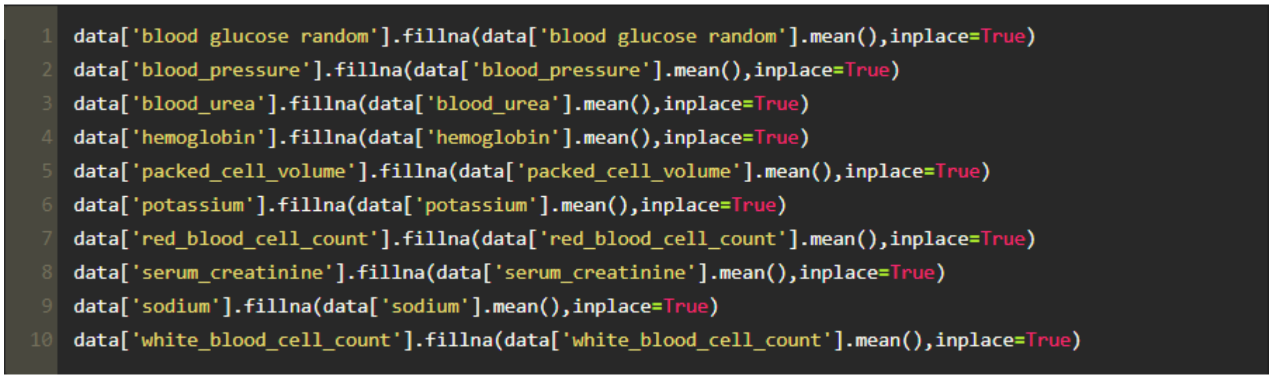
```
data.info() #info will give you a summary of dataset
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 25 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   age                      391 non-null    float64
 1   blood_pressure           388 non-null    float64
 2   specific_gravity         353 non-null    float64
 3   albumin                  354 non-null    float64
 4   sugar                    351 non-null    float64
 5   red_blood_cells          248 non-null    object 
 6   pus_cell                 335 non-null    object 
 7   pus_cell_clumps          396 non-null    object 
 8   bacteria                 396 non-null    object 
 9   blood glucose random     356 non-null    float64
 10  blood_urea               381 non-null    float64
 11  serum_creatinine         383 non-null    float64
 12  sodium                   313 non-null    float64
 13  potassium                312 non-null    float64
 14  hemoglobin               348 non-null    float64
 15  packed_cell_volume       330 non-null    object 
 16  white_blood_cell_count   295 non-null    object 
 17  red_blood_cell_count     270 non-null    object 
 18  hypertension             398 non-null    object 
 19  diabetesmellitus         398 non-null    object 
 20  coronary_artery_disease  398 non-null    object 
 21  appetite                 399 non-null    object 
 22  pedal_edema              399 non-null    object 
 23  anemia                   399 non-null    object 
 24  class                    400 non-null    object 
dtypes: float64(11), object(14)
memory usage: 78.2+ KB
```

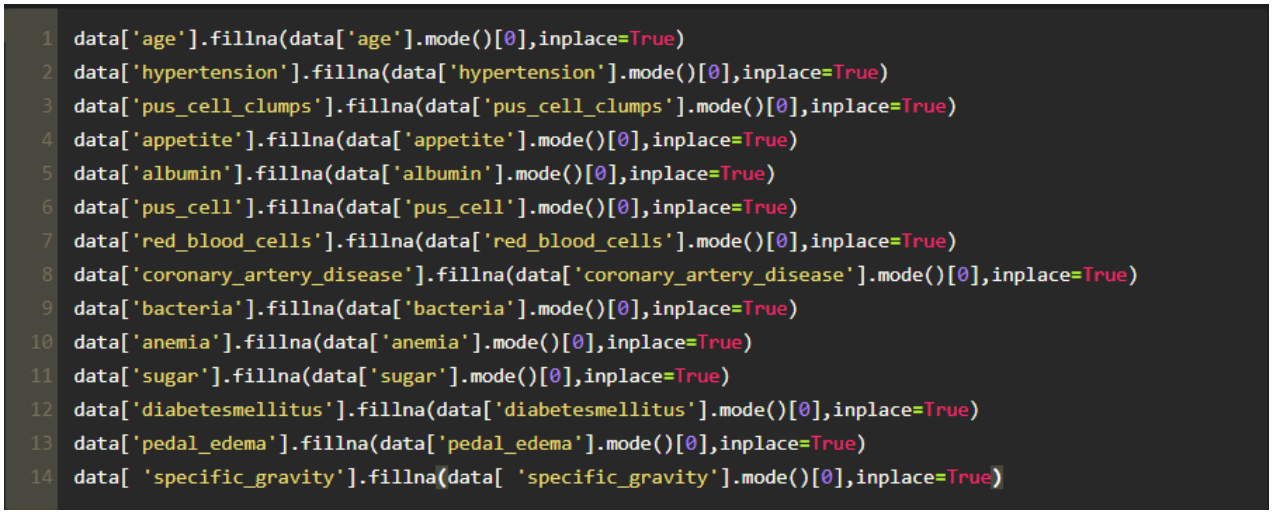


```
data.isnull().any() #it will return true if any columns is having null values
```

```
age                         True
blood_pressure              True
specific_gravity            True
albumin                     True
sugar                       True
red_blood_cells             True
pus_cell                    True
pus_cell_clumps             True
bacteria                    True
blood glucose random        True
blood_urea                  True
serum_creatinine            True
sodium                      True
potassium                   True
hemoglobin                  True
packed_cell_volume          True
white_blood_cell_count      True
red_blood_cell_count        True
hypertension                True
diabetesmellitus            True
coronary_artery_disease     True
appetite                    True
pedal_edema                 True
anemia                      True
class                      False
dtype: bool
```



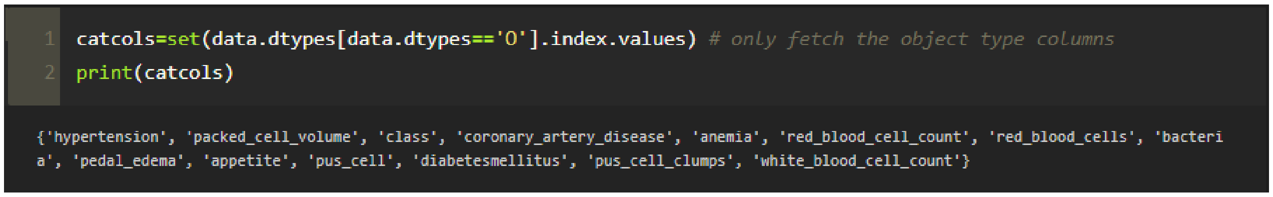
```python
data['blood glucose random'].fillna(data['blood glucose random'].mean(),inplace=True)
data['blood_pressure'].fillna(data['blood_pressure'].mean(),inplace=True)
data['blood_urea'].fillna(data['blood_urea'].mean(),inplace=True)
data['hemoglobin'].fillna(data['hemoglobin'].mean(),inplace=True)
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)
```



```python
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)
```

## Handling Categorical columns:

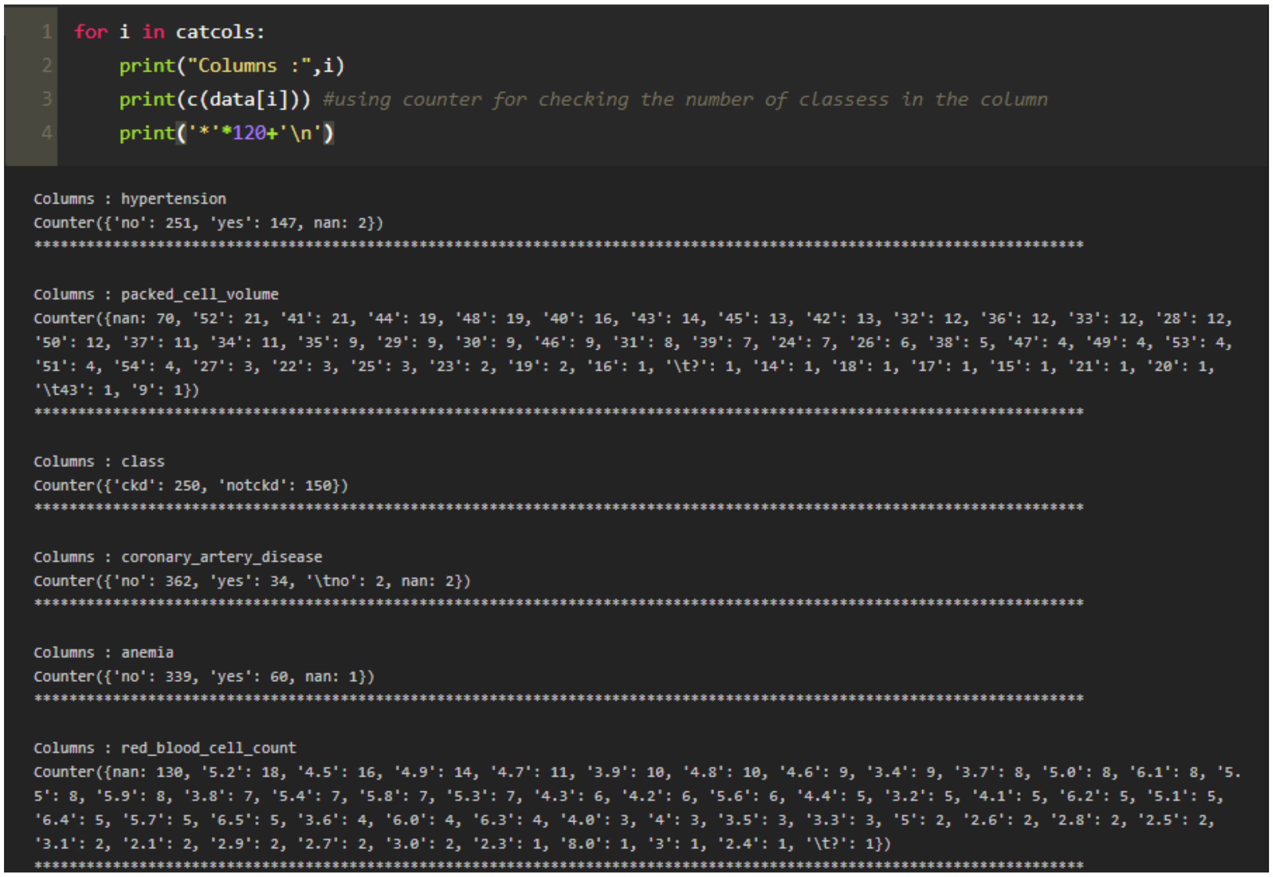
The below code is used for fetching all the object or categorical type of columns from
our data and we are storing it as **set** in variable **catcols.**



```python
catcols=set(data.dtypes[data.dtypes=='O'].index.values) # only fetch the object type columns
print(catcols)
```

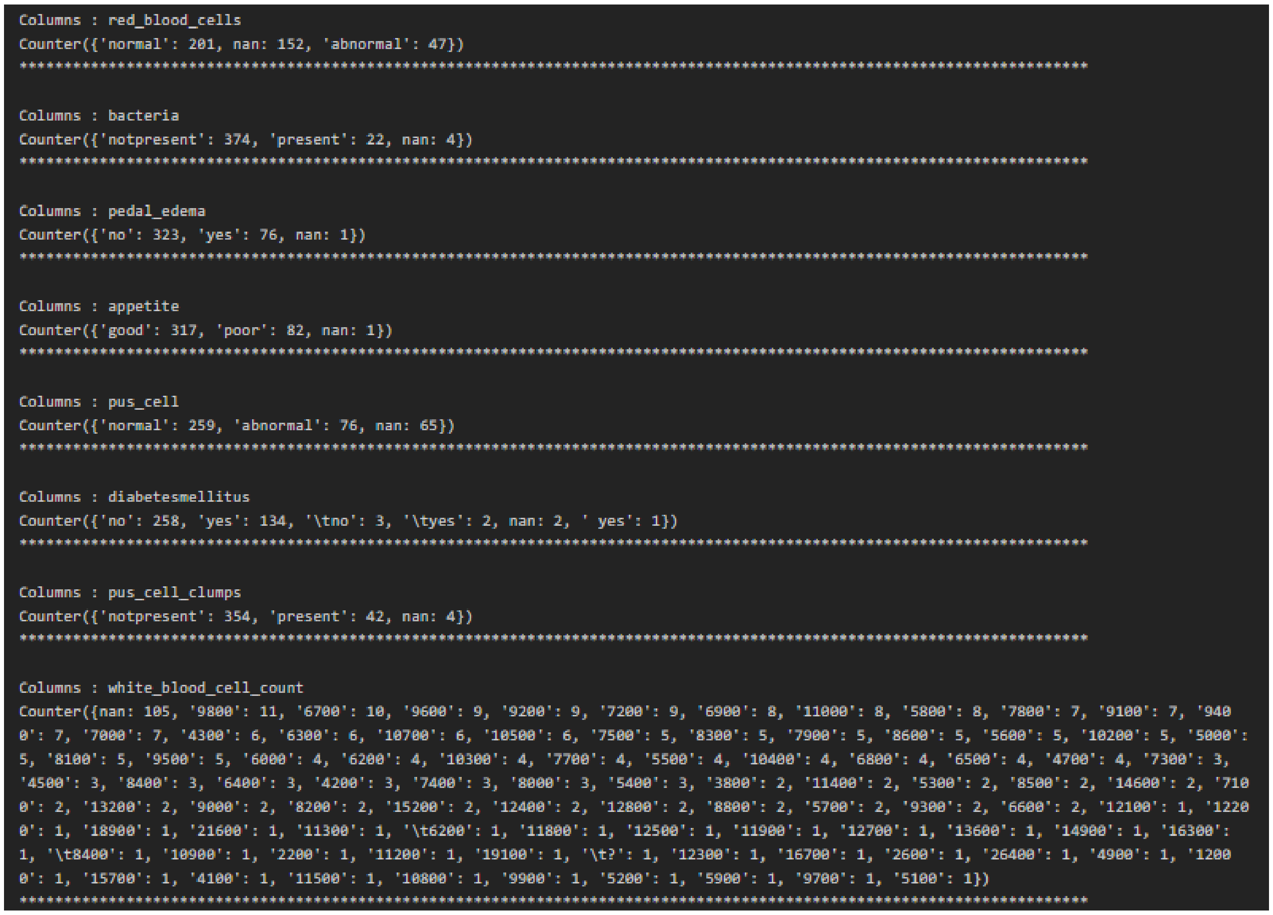
```
{'hypertension', 'packed_cell_volume', 'class', 'coronary_artery_disease', 'anemia', 'red_blood_cell_count', 'red_blood_cells', 'bacteri
a', 'pedal_edema', 'appetite', 'pus_cell', 'diabetesmellitus', 'pus_cell_clumps', 'white_blood_cell_count'}
```

As, you can observe that it gives us the same count of columns which we find
previously.



```python
for i in catcols:
    print("Columns :",i)
    print(c(data[i])) #using counter for checking the number of classess in the column
    print('*'*120+'\n')
```

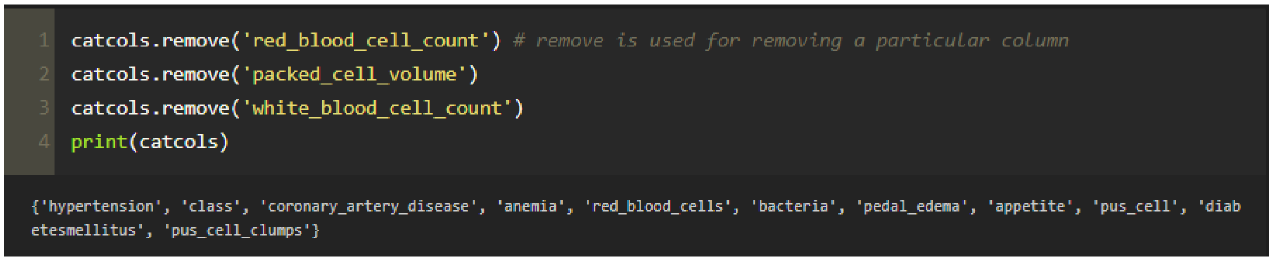
```
Columns : hypertension
Counter({'no': 251, 'yes': 147, nan: 2})
************************************************************************************************************************

Columns : packed_cell_volume
Counter({nan: 70, '52': 21, '41': 21, '44': 19, '48': 19, '40': 16, '43': 14, '45': 13, '42': 13, '32': 12, '36': 12, '33': 12, '28': 12,
'50': 12, '37': 11, '34': 11, '35': 9, '29': 9, '30': 9, '46': 9, '31': 8, '39': 7, '24': 7, '26': 6, '38': 5, '47': 4, '49': 4, '53': 4,
'51': 4, '54': 4, '27': 3, '22': 3, '25': 3, '23': 2, '19': 2, '16': 1, '\t?': 1, '14': 1, '18': 1, '17': 1, '15': 1, '21': 1, '20': 1,
'\t43': 1, '9': 1})
************************************************************************************************************************

Columns : class
Counter({'ckd': 250, 'notckd': 150})
************************************************************************************************************************

Columns : coronary_artery_disease
Counter({'no': 362, 'yes': 34, '\tno': 2, nan: 2})
************************************************************************************************************************

Columns : anemia
Counter({'no': 339, 'yes': 60, nan: 1})
************************************************************************************************************************

Columns : red_blood_cell_count
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})
************************************************************************************************************************
```



```
Columns : red_blood_cells
Counter({'normal': 201, nan: 152, 'abnormal': 47})
****************************************************************************************************************

Columns : bacteria
Counter({'notpresent': 374, 'present': 22, nan: 4})
****************************************************************************************************************

Columns : pedal_edema
Counter({'no': 323, 'yes': 76, nan: 1})
****************************************************************************************************************

Columns : appetite
Counter({'good': 317, 'poor': 82, nan: 1})
****************************************************************************************************************

Columns : pus_cell
Counter({'normal': 259, 'abnormal': 76, nan: 65})
****************************************************************************************************************

Columns : diabetesmellitus
Counter({'no': 258, 'yes': 134, '\tno': 3, '\tyes': 2, nan: 2, ' yes': 1})
****************************************************************************************************************

Columns : pus_cell_clumps
Counter({'notpresent': 354, 'present': 42, nan: 4})
****************************************************************************************************************

Columns : white_blood_cell_count
Counter({nan: 105, '9800': 11, '6700': 10, '9600': 9, '9200': 9, '7200': 9, '6900': 8, '11000': 8, '5800': 8, '7800': 7, '9100': 7, '940
0': 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, '710
0': 2, '13200': 2, '9000': 2, '8200': 2, '15200': 2, '12400': 2, '12800': 2, '8800': 2, '5700': 2, '9300': 2, '6600': 2, '12100': 1, '1220
0': 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, '1200
0': 1, '15700': 1, '4100': 1, '11500': 1, '10800': 1, '9900': 1, '5200': 1, '5900': 1, '9700': 1, '5100': 1})
****************************************************************************************************************
```

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.
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.



```
catcols.remove('red_blood_cell_count') # remove is used for removing a particular column
catcols.remove('packed_cell_volume')
catcols.remove('white_blood_cell_count')
print(catcols)
```

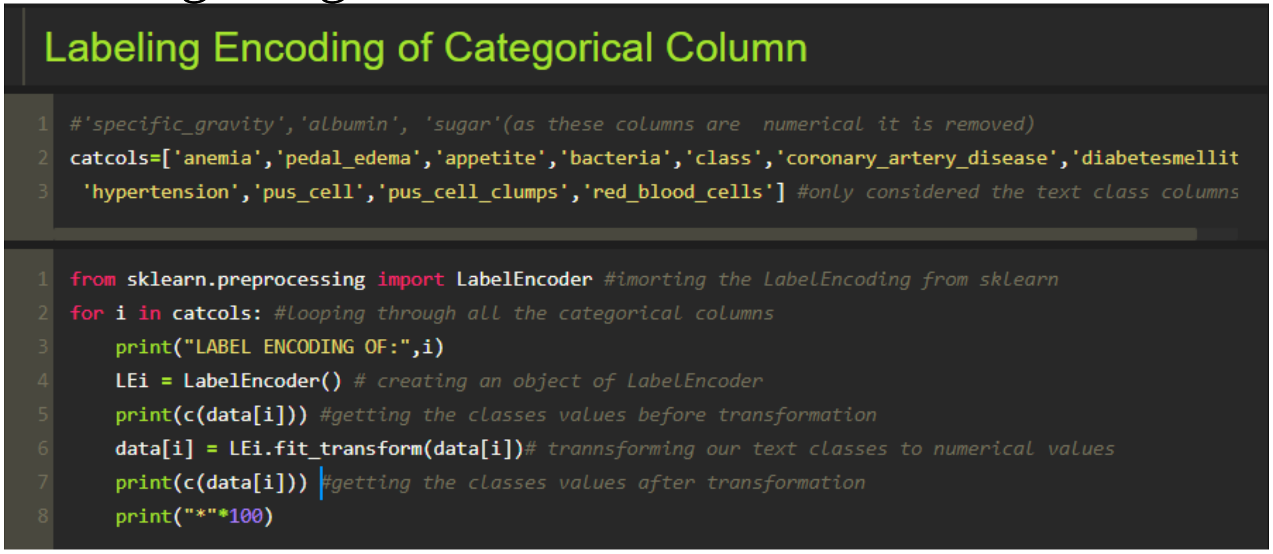
```
{'hypertension', 'class', 'coronary_artery_disease', 'anemia', 'red_blood_cells', 'bacteria', 'pedal_edema', 'appetite', 'pus_cell', 'diab
etesmellitus', 'pus_cell_clumps'}
```

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.

## Label Encoding for categorical columns:

Typically, any structured dataset includes multiple columns with combination of
numerical as well as categorical variables. A machine can only understand the
numbers. It cannot understand the text. That’s essentially the case with Machine
Learning algorithms too.We need to convert each text category to numbers in order
for the machine to process those using mathematical equations.
How should we handle categorical variables? There are Multiple way to handle, but
will see one of it is LabelEncoding.

**Label Encoding** is a popular encoding technique for handling categorical variables.

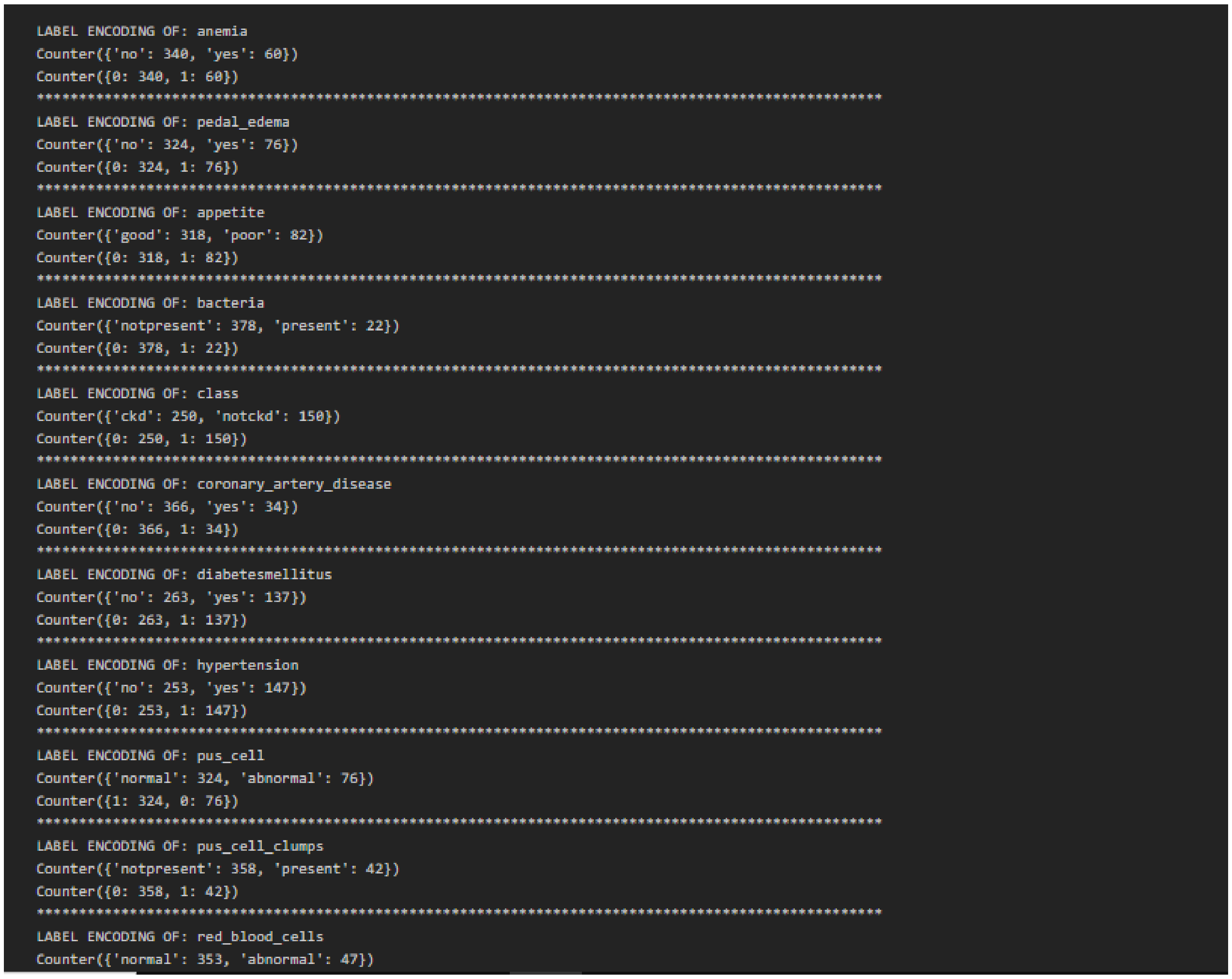


## Labeling Encoding of Categorical Column

```
#'specific_gravity','albumin', 'sugar'(as these columns are  numerical it is removed)
catcols=['anemia','pedal_edema','appetite','bacteria','class','coronary_artery_disease','diabetesmellit
 'hypertension','pus_cell','pus_cell_clumps','red_blood_cells'] #only considered the text class columns
```

```
from sklearn.preprocessing import LabelEncoder #imorting the LabelEncoding from sklearn
for i in catcols: #looping through all the categorical columns
    print("LABEL ENCODING OF:",i)
    LEi = LabelEncoder() # creating an object of LabelEncoder
    print(c(data[i])) #getting the classes values before transformation
    data[i] = LEi.fit_transform(data[i])# trannsforming our text classes to numerical values
    print(c(data[i])) #getting the classes values after transformation
    print("*"*100)
```

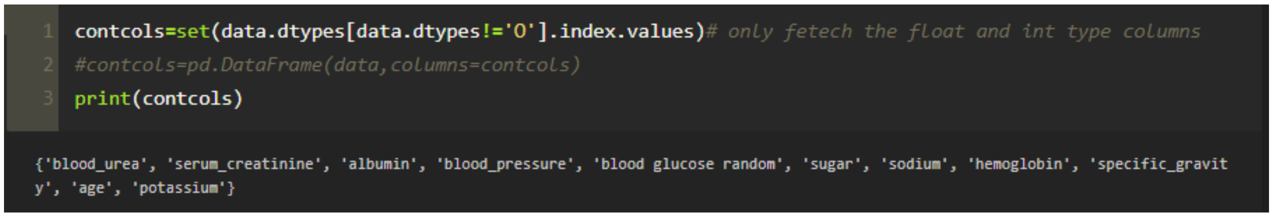
In this technique, each label is assigned a unique integer based on alphabetical
ordering.
Let’s see how to implement label encoding in Python using the scikit-learn library.
we have to convert only the text class category columns; we first select it then we will
implement Label Encoding to it.In the above code we are looping through all the selected text class categorical
columns and performing label encoding.



```
LABEL ENCODING OF: anemia
Counter({'no': 340, 'yes': 60})
Counter({0: 340, 1: 60})
****************************************************************************************************
LABEL ENCODING OF: pedal_edema
Counter({'no': 324, 'yes': 76})
Counter({0: 324, 1: 76})
****************************************************************************************************
LABEL ENCODING OF: appetite
Counter({'good': 318, 'poor': 82})
Counter({0: 318, 1: 82})
****************************************************************************************************
LABEL ENCODING OF: bacteria
Counter({'notpresent': 378, 'present': 22})
Counter({0: 378, 1: 22})
****************************************************************************************************
LABEL ENCODING OF: class
Counter({'ckd': 250, 'notckd': 150})
Counter({0: 250, 1: 150})
****************************************************************************************************
LABEL ENCODING OF: coronary_artery_disease
Counter({'no': 366, 'yes': 34})
Counter({0: 366, 1: 34})
****************************************************************************************************
LABEL ENCODING OF: diabetesmellitus
Counter({'no': 263, 'yes': 137})
Counter({0: 263, 1: 137})
****************************************************************************************************
LABEL ENCODING OF: hypertension
Counter({'no': 253, 'yes': 147})
Counter({0: 253, 1: 147})
****************************************************************************************************
LABEL ENCODING OF: pus_cell
Counter({'normal': 324, 'abnormal': 76})
Counter({1: 324, 0: 76})
****************************************************************************************************
LABEL ENCODING OF: pus_cell_clumps
Counter({'notpresent': 358, 'present': 42})
Counter({0: 358, 1: 42})
****************************************************************************************************
LABEL ENCODING OF: red_blood_cells
Counter({'normal': 353, 'abnormal': 47})
```

As you can see here, after performing label encoding alphabetical classes is
converted to numeric.

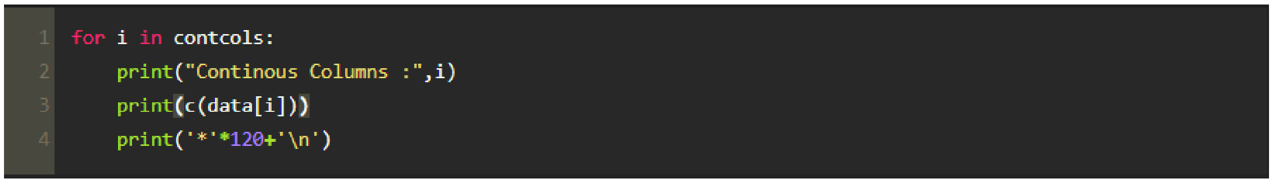
## Handling Numerical columns:



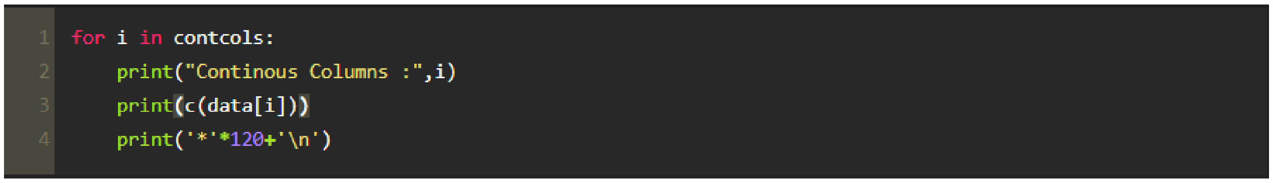
```
contcols=set(data.dtypes[data.dtypes!='O'].index.values)# only fetech the float and int type columns
#contcols=pd.DataFrame(data,columns=contcols)
print(contcols)
```

```
{'blood_urea', 'serum_creatinine', 'albumin', 'blood_pressure', 'blood glucose random', 'sugar', 'sodium', 'hemoglobin', 'specific_gravit
y', 'age', 'potassium'}
```

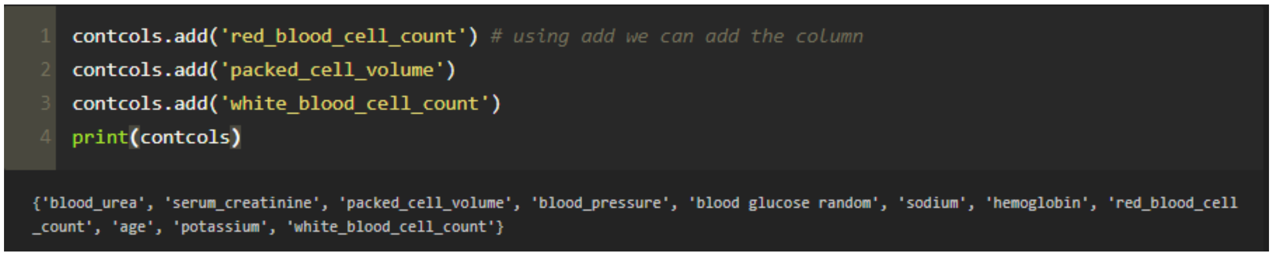
Same as we did with categorical columns, we are majing use of **dtypes** for finding the
continuous columns


```
for i in contcols:
    print("Continous Columns :",i)
    print(c(data[i]))
    print('*'*120+'\n')
```

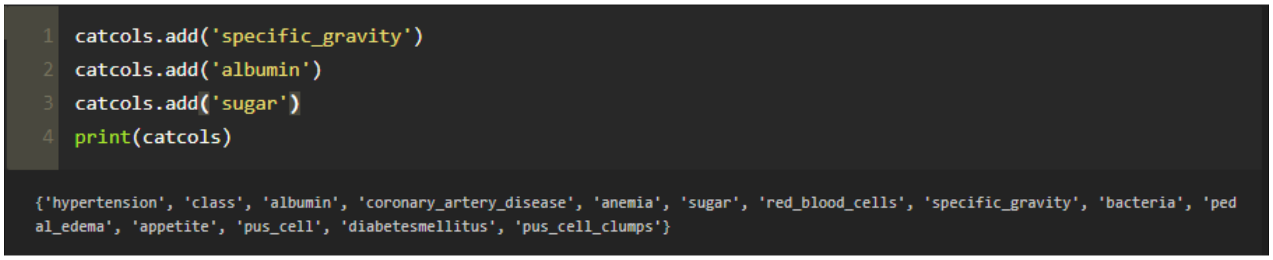
If we observe the output of the above code we can observe that some columns have
few values or you can say classes which can be considered as categorical columns.
So , let’s remove it and add the columns which we observed into their respective
variables.


```
for i in contcols:
    print("Continous Columns :",i)
    print(c(data[i]))
    print('*'*120+'\n')
```

With the help of add() function we can add an element.


```
contcols.add('red_blood_cell_count') # using add we can add the column
contcols.add('packed_cell_volume')
contcols.add('white_blood_cell_count')
print(contcols)
```

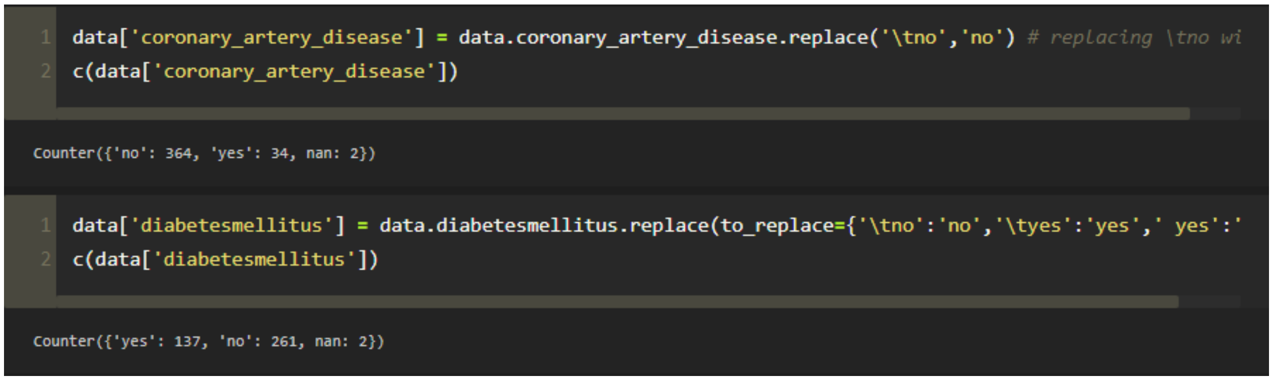
```
{'blood_urea', 'serum_creatinine', 'packed_cell_volume', 'blood_pressure', 'blood glucose random', 'sodium', 'hemoglobin', 'red_blood_cell
_count', 'age', 'potassium', 'white_blood_cell_count'}
```



```
catcols.add('specific_gravity')
catcols.add('albumin')
catcols.add('sugar')
print(catcols)
```

```
{'hypertension', 'class', 'albumin', 'coronary_artery_disease', 'anemia', 'sugar', 'red_blood_cells', 'specific_gravity', 'bacteria', 'ped
al_edema', 'appetite', 'pus_cell', 'diabetesmellitus', 'pus_cell_clumps'}
```

In our data some columns some unwanted classes so we have to rectify that also for
that we simply use **replace()**



```
data['coronary_artery_disease'] = data.coronary_artery_disease.replace('\tno','no') # replacing \tno wi
c(data['coronary_artery_disease'])
```

```
Counter({'no': 364, 'yes': 34, nan: 2})
```

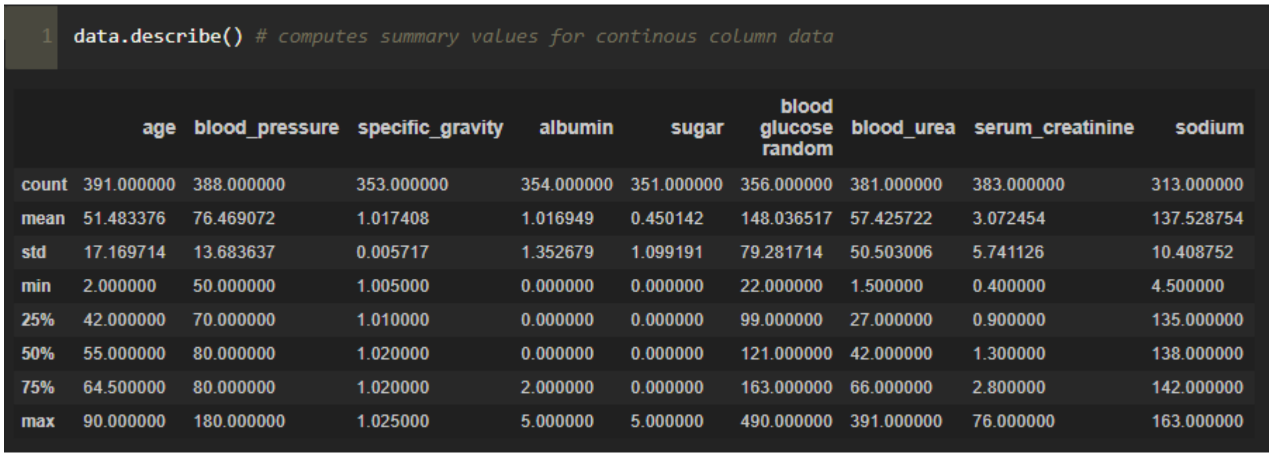
```
data['diabetesmellitus'] = data.diabetesmellitus.replace(to_replace={'\tno':'no','\tyes':'yes',' yes':'
c(data['diabetesmellitus'])
```

```
Counter({'yes': 137, 'no': 261, nan: 2})
```

## Exploratory Data Analysis:

### Descriptive statistical Analysis:

Descriptive analysis is to study the basic features of data with the statistical process.
Here pandas has a worthy function called describe. With this describe function we
can understand the unique, top and frequent values of categorical features. And we
can find mean, std, min, max and percentile values of continuous features.



```
data.describe() # computes summary values for continous column data
```

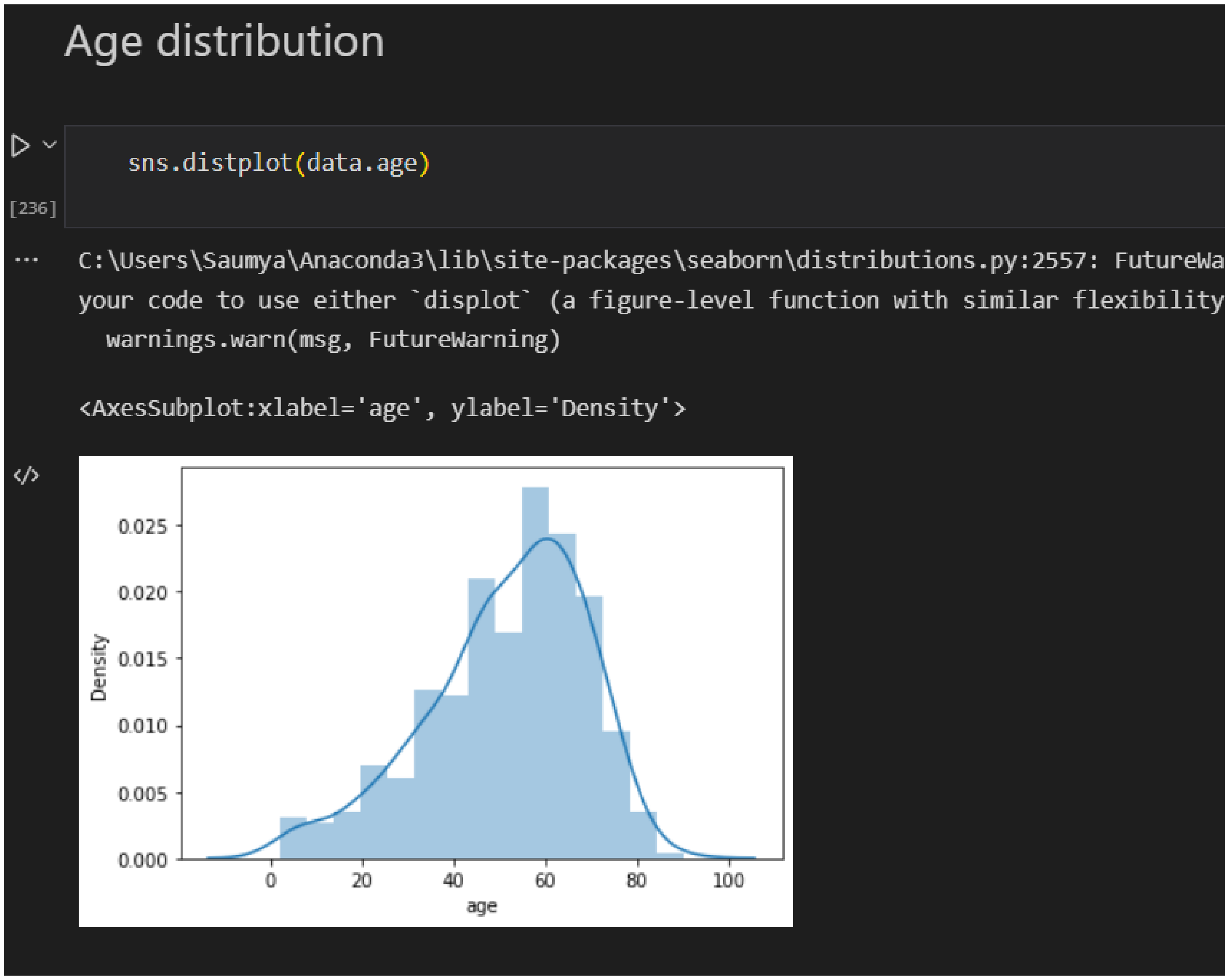
| | age | blood_pressure | specific_gravity | albumin | sugar | blood glucose random | blood_urea | serum_creatinine | sodium |
|---|---|---|---|---|---|---|---|---|---|
| count | 391.000000 | 388.000000 | 353.000000 | 354.000000 | 351.000000 | 356.000000 | 381.000000 | 383.000000 | 313.000000 |
| mean | 51.483376 | 76.469072 | 1.017408 | 1.016949 | 0.450142 | 148.036517 | 57.425722 | 3.072454 | 137.528754 |
| std | 17.169714 | 13.683637 | 0.005717 | 1.352679 | 1.099191 | 79.281714 | 50.503006 | 5.741126 | 10.408752 |
| min | 2.000000 | 50.000000 | 1.005000 | 0.000000 | 0.000000 | 22.000000 | 1.500000 | 0.400000 | 4.500000 |
| 25% | 42.000000 | 70.000000 | 1.010000 | 0.000000 | 0.000000 | 99.000000 | 27.000000 | 0.900000 | 135.000000 |
| 50% | 55.000000 | 80.000000 | 1.020000 | 0.000000 | 0.000000 | 121.000000 | 42.000000 | 1.300000 | 138.000000 |
| 75% | 64.500000 | 80.000000 | 1.020000 | 2.000000 | 0.000000 | 163.000000 | 66.000000 | 2.800000 | 142.000000 |
| max | 90.000000 | 180.000000 | 1.025000 | 5.000000 | 5.000000 | 490.000000 | 391.000000 | 76.000000 | 163.000000 |

## Visual analysis:

Visual analysis is the process of using visual representations, such as charts, plots,
and graphs, to explore and understand data. It is a way to quickly identify patterns,
trends, and outliers in the data, which can help to gain insights and make informed
decisions.

## Univariate analysis:

In simple words, univariate analysis is understanding the data with a single feature.
Here we have displayed two different graphs such as distplot and countplot.The Seaborn package provides a wonderful function distplot. With the help of distplot,
we can find the distribution of the feature.

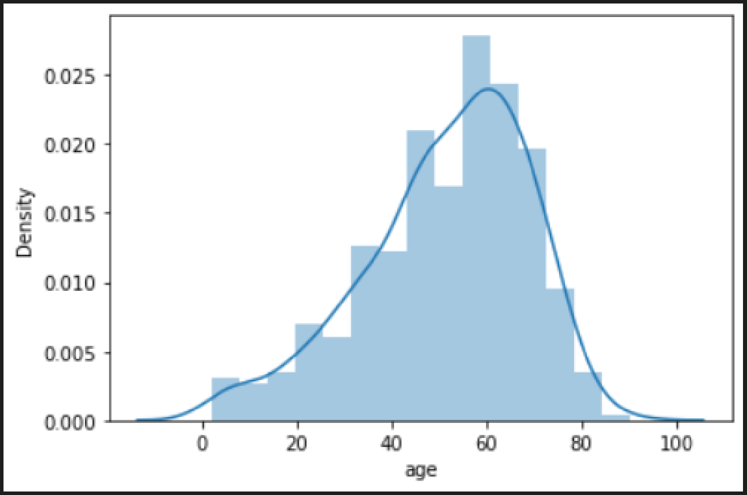


Age distribution

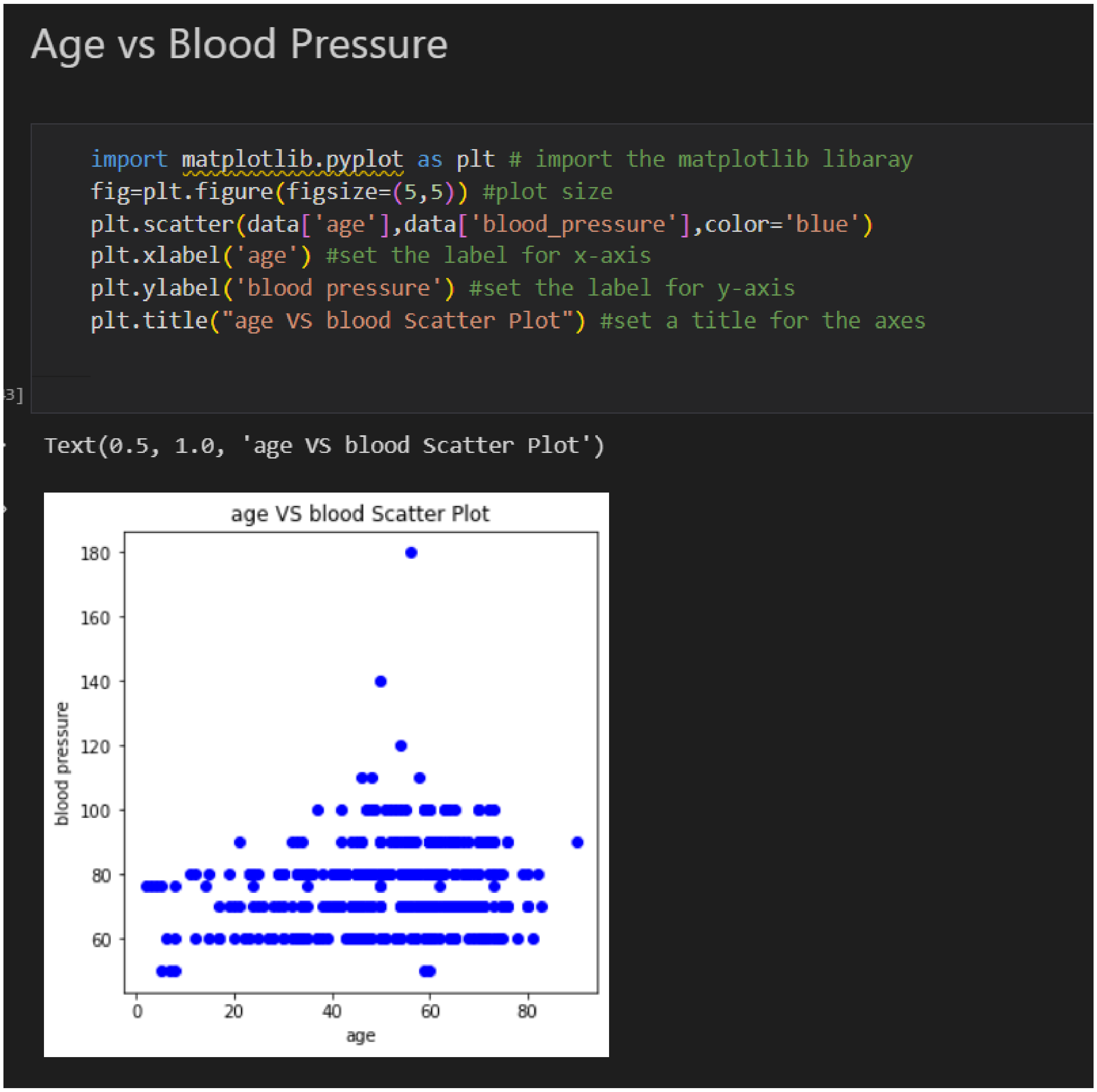
```
sns.distplot(data.age)
```

[236]

```
C:\Users\Saumya\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWa
your code to use either `displot` (a figure-level function with similar flexibility
  warnings.warn(msg, FutureWarning)

<AxesSubplot:xlabel='age', ylabel='Density'>
```



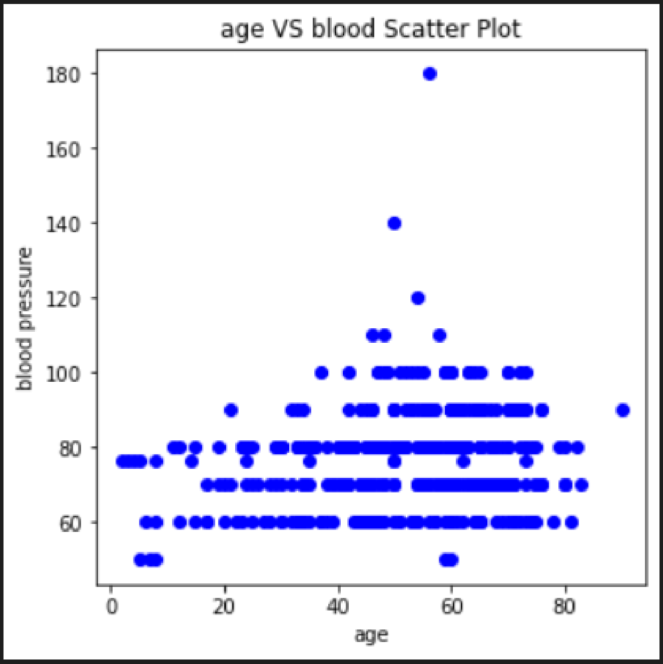
## Bivariate analysis:



## Age vs Blood Pressure

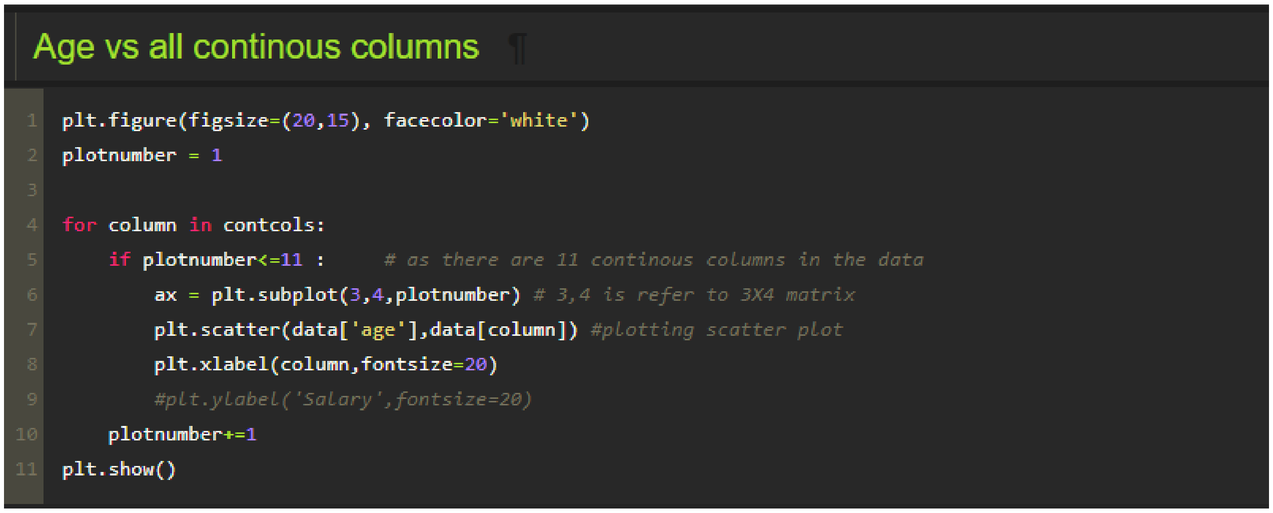
```
import matplotlib.pyplot as plt # import the matplotlib libaray
fig=plt.figure(figsize=(5,5)) #plot size
plt.scatter(data['age'],data['blood_pressure'],color='blue')
plt.xlabel('age') #set the label for x-axis
plt.ylabel('blood pressure') #set the label for y-axis
plt.title("age VS blood Scatter Plot") #set a title for the axes
```

```
Text(0.5, 1.0, 'age VS blood Scatter Plot')
```



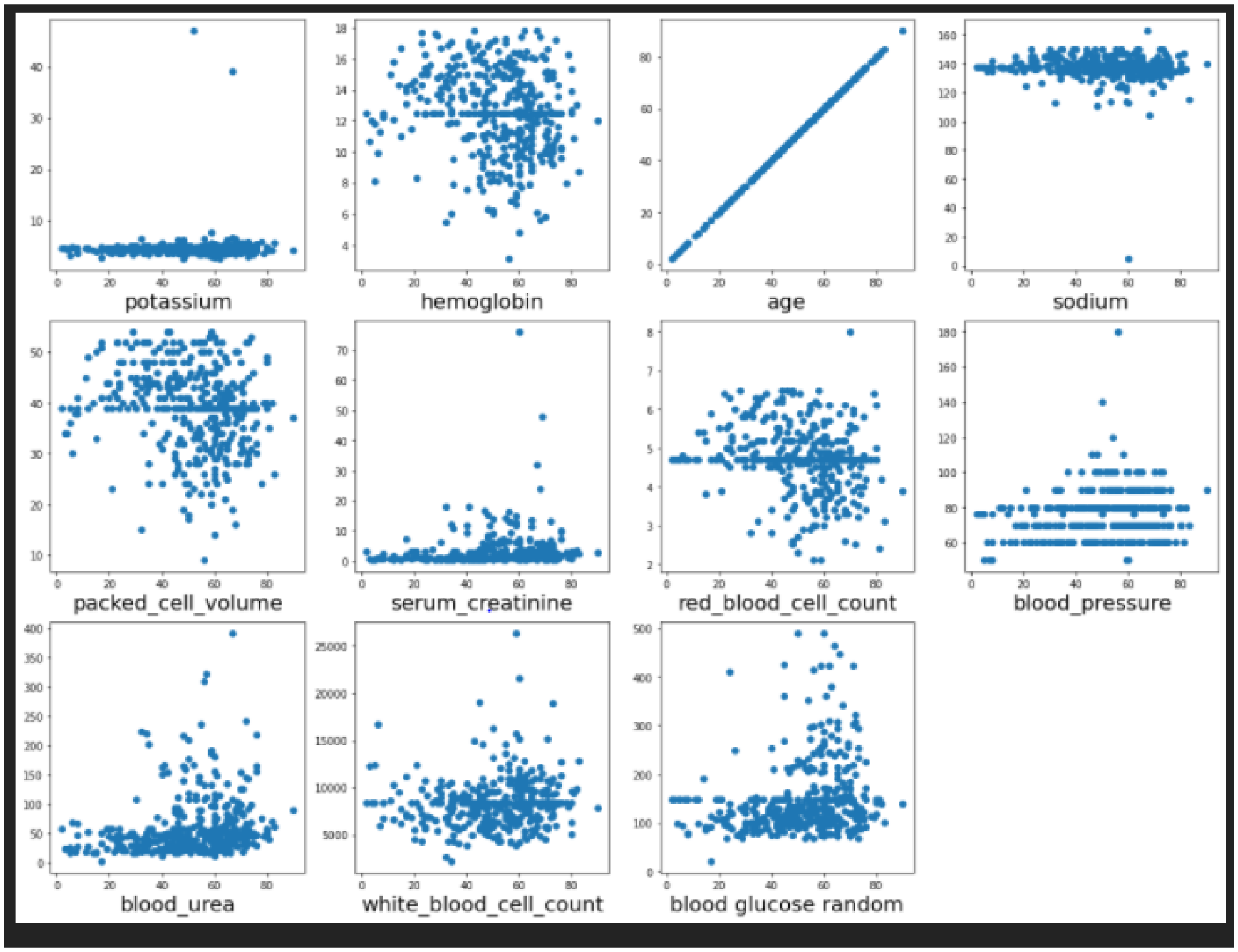
## Multivariate analysis:

## Age vs all continuous columns:



## Age vs all continous columns

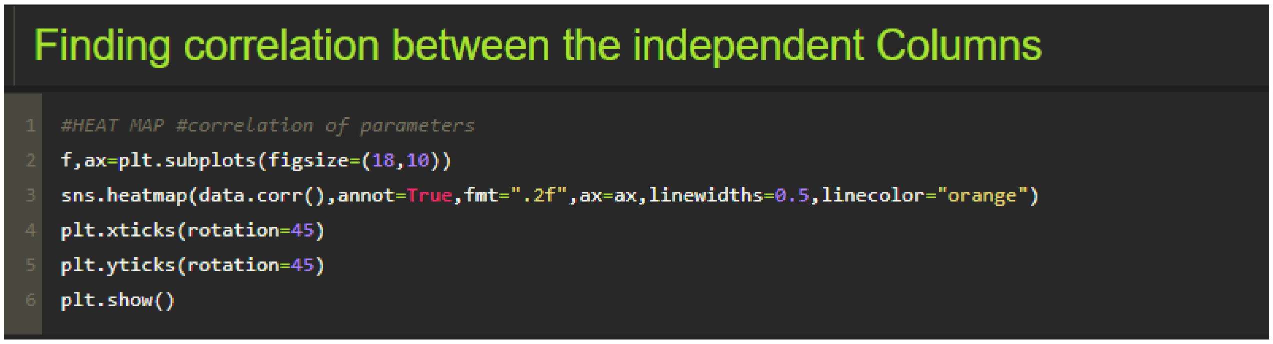
```
plt.figure(figsize=(20,15), facecolor='white')
plotnumber = 1

for column in contcols:
    if plotnumber<=11 :     # as there are 11 continous columns in the data
        ax = plt.subplot(3,4,plotnumber) # 3,4 is refer to 3X4 matrix
        plt.scatter(data['age'],data[column]) #plotting scatter plot
        plt.xlabel(column,fontsize=20)
        #plt.ylabel('Salary',fontsize=20)
    plotnumber+=1
plt.show()
```



As you can observe with the scatter plot many of features are correlated with age.

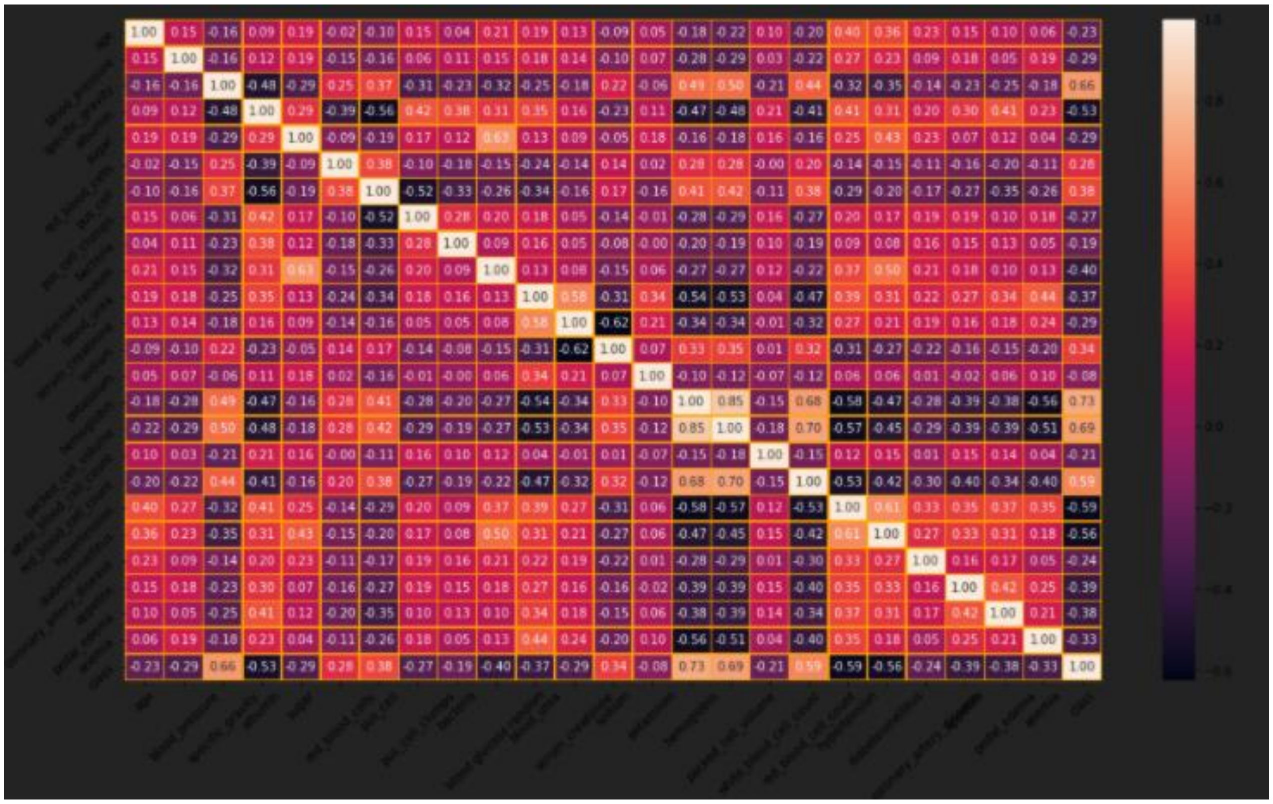
## Finding correlation between the independent Columns:

Correlation is a statistical relationship between two variables and it could be positive,
meaning both variables move in the same direction, or negative, meaning that when
one variable's value increases, the other variables' values decrease.
With the help of seaborn heatmap we will be plotting the heatmap and for finding the
correlation between variable we have **corr()** available.

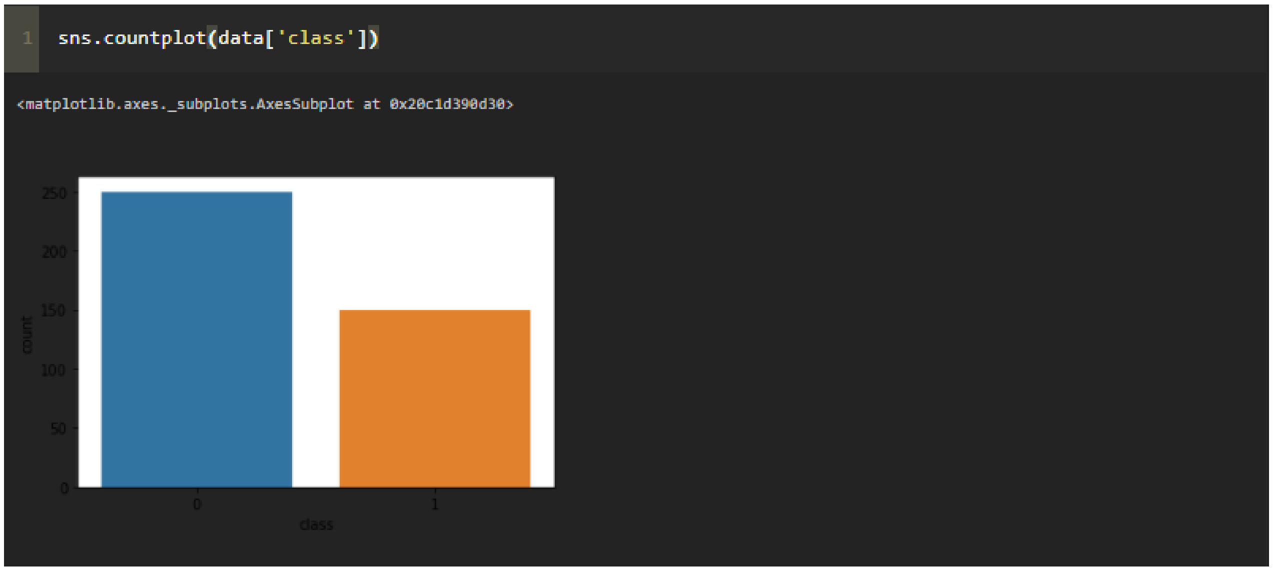


Finding correlation between the independent Columns

```
#HEAT MAP #correlation of parameters
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()
```



If you observe the heatmap, lighter the colour the correlation between that two
variables will be high.
And correlation plays a very important role for extracting the correct features for build
our model.
Now, let’s observe the count of our target data classes, by using seaborn countplot

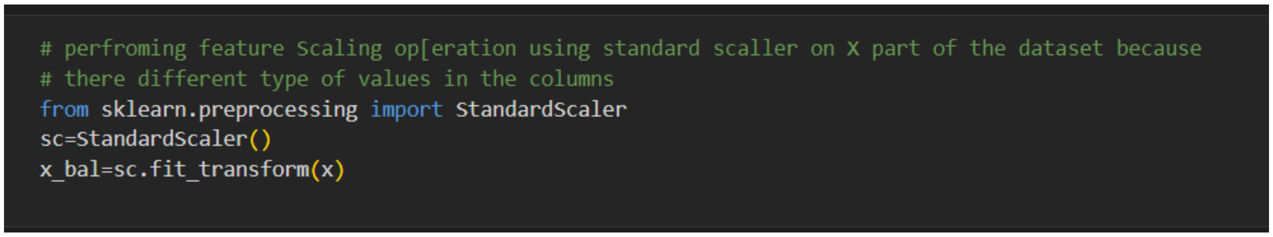


```
sns.countplot(data['class'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x20c1d390d30>
```

## Scaling the Data:

Scaling is one the important process, we have to perform on the dataset, because of
data measures in different ranges can leads to mislead in prediction
Models such as KNN, Logistic regression need scaled data, as they follow
distance-based method and Gradient Descent concept.

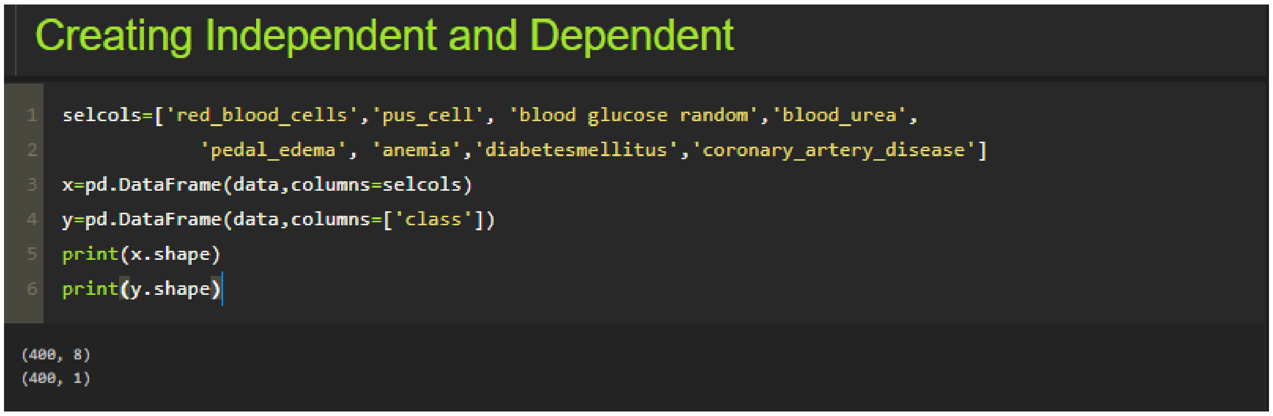


```
# perfroming feature Scaling op[eration using standard scaller on X part of the dataset because
# there different type of values in the columns
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_bal=sc.fit_transform(x)
```

We will perform scaling only on the input values.Once the dataset is scaled, it will be converted into an array and we need to convert it back to a dataframe.

**Separate independent and dependent variable:**

Now let’s split the Dataset into train and test sets
Changes: first split the dataset into x and y and then split the data set
Here x and y variables are created. On x variable, df is passed with dropping the
target variable. And on y target variable is passed. For splitting training and testing
data we are using the train_test_split() function from sklearn. As parameters, we are
passing x, y, test_size, random_state.



## Creating Independent and Dependent

```
selcols=['red_blood_cells','pus_cell', 'blood glucose random','blood_urea',
             'pedal_edema', 'anemia','diabetesmellitus','coronary_artery_disease']
x=pd.DataFrame(data,columns=selcols)
y=pd.DataFrame(data,columns=['class'])
print(x.shape)
print(y.shape)
```

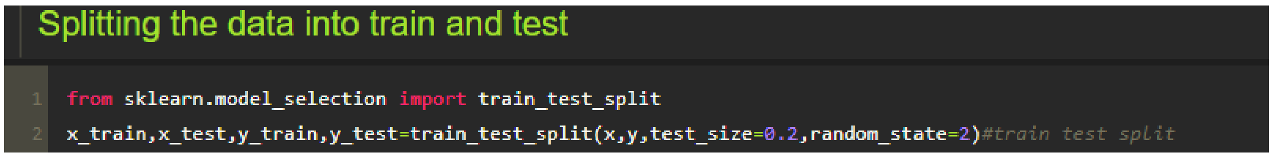
```
(400, 8)
(400, 1)
```

In the above code we are creating DataFrame of the independent variable **x** with our
selected columns and for dependent variable **y** we are only taking the **class** column.

Where **DataFrame is used to** represents a table of data with rows and columns.

## Splitting data into train and test

When you are working on a model and you want to train it, you obviously have a
dataset. But after training, we have to test the model on some test dataset. For this,
you will a dataset which is different from the training set you used earlier. But it might
not always be possible to have so much data during the development phase. In such
cases, the solution is to split the dataset into two sets, one for training and the other
for testing.


Splitting the data into train and test

```
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)#train test split
```

## Model Building:

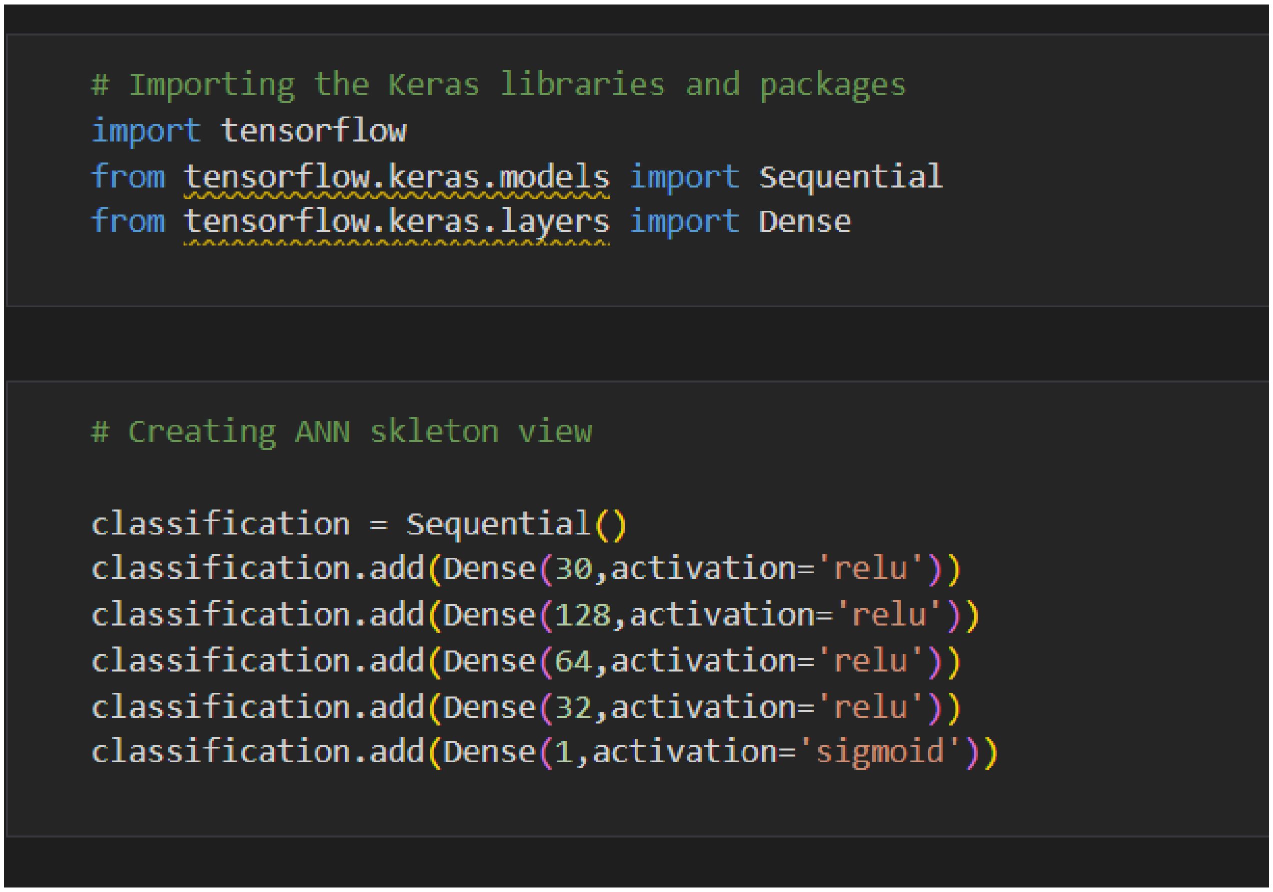
## Training the model in multiple algorithms:

Now our data is cleaned and it’s time to build the model. We can train our data on

different algorithms. For this project we are applying four classification algorithms.
The best model is saved based on its performance.

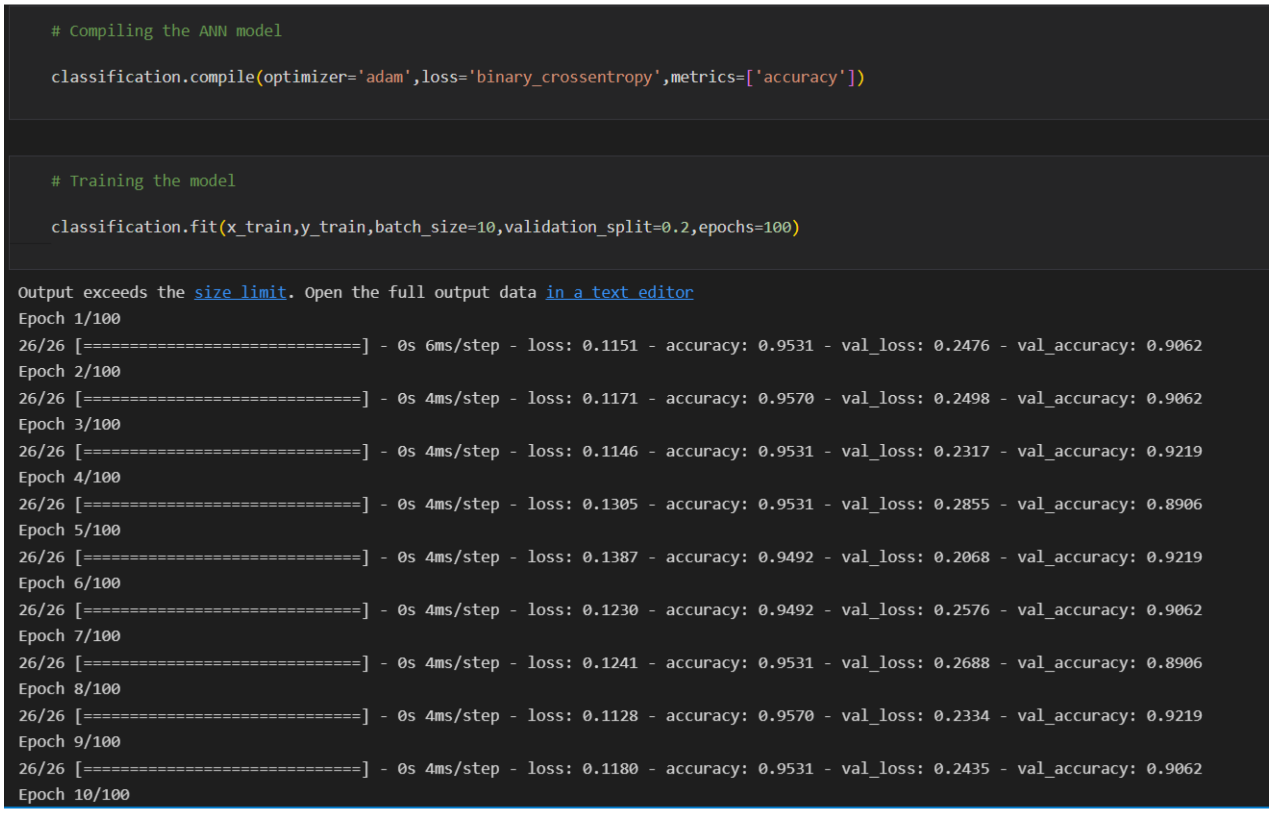
## ANN Model:

Building and training an Artificial Neural Network (ANN) using the Keras library with
TensorFlow as the backend. The ANN is initialised as an instance of the Sequential
class, which is a linear stack of layers. Then, the input layer and two hidden layers
are added to the model using the Dense class, where the number of units and
activation function are specified. The output layer is also added using the Dense
class with a sigmoid activation function. The model is then compiled with the Adam
optimizer, binary cross-entropy loss function, and accuracy metric. Finally, the model
is fit to the training data with a batch size of 100, 20% validation split, and 100 epochs.



```python
# Importing the Keras libraries and packages
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

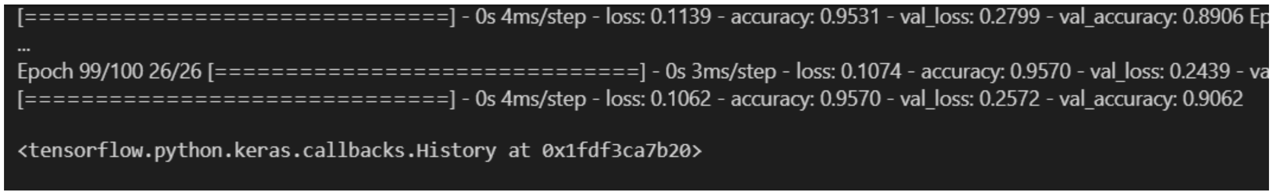
```python
# Creating ANN skleton view

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'))
```



```python
# Compiling the ANN model

classification.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

```python
# Training the model

classification.fit(x_train,y_train,batch_size=10,validation_split=0.2,epochs=100)
```

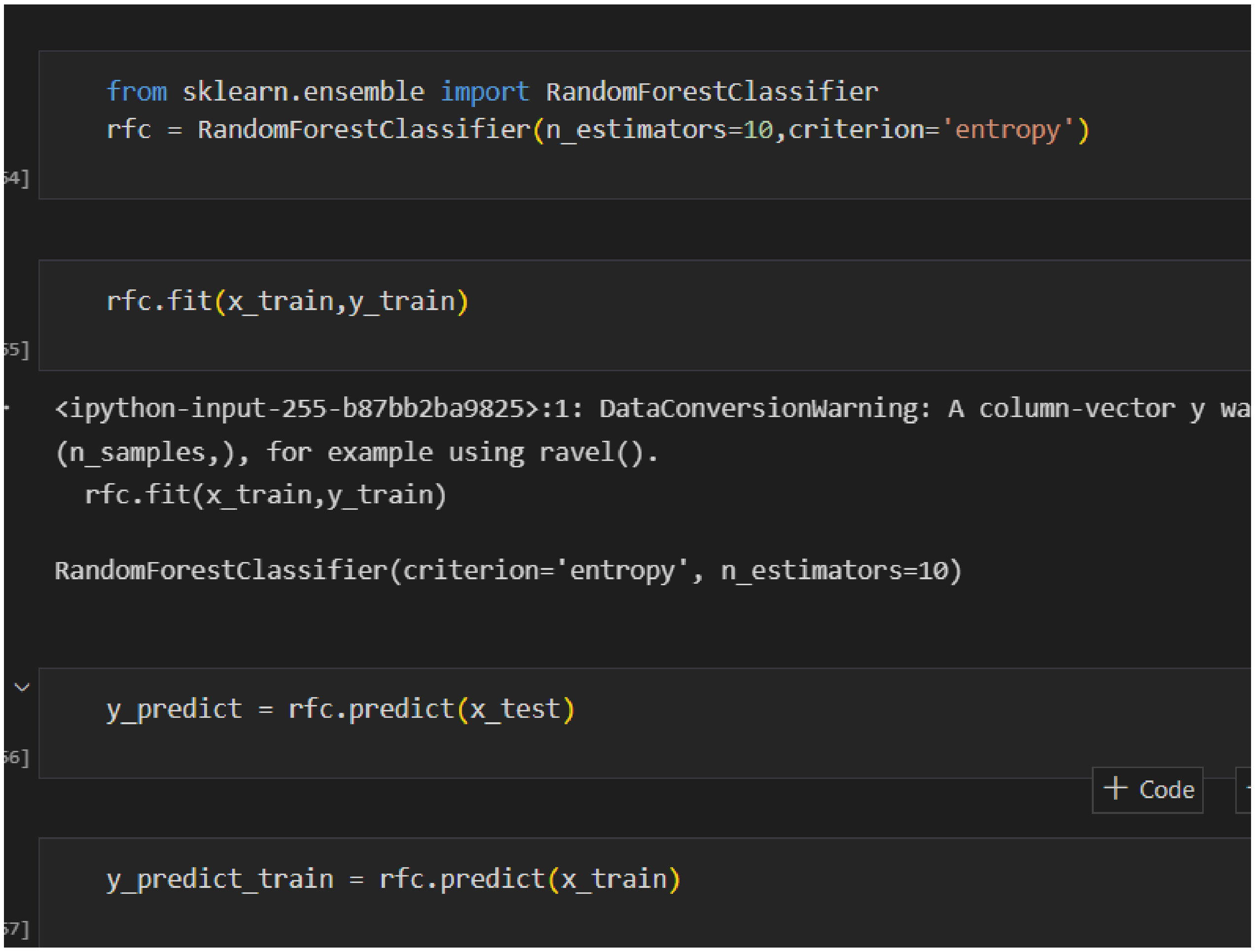
```
Output exceeds the size limit. Open the full output data in a text editor
Epoch 1/100
26/26 [==============================] - 0s 6ms/step - loss: 0.1151 - accuracy: 0.9531 - val_loss: 0.2476 - val_accuracy: 0.9062
Epoch 2/100
26/26 [==============================] - 0s 4ms/step - loss: 0.1171 - accuracy: 0.9570 - val_loss: 0.2498 - val_accuracy: 0.9062
Epoch 3/100
26/26 [==============================] - 0s 4ms/step - loss: 0.1146 - accuracy: 0.9531 - val_loss: 0.2317 - val_accuracy: 0.9219
Epoch 4/100
26/26 [==============================] - 0s 4ms/step - loss: 0.1305 - accuracy: 0.9531 - val_loss: 0.2855 - val_accuracy: 0.8906
Epoch 5/100
26/26 [==============================] - 0s 4ms/step - loss: 0.1387 - accuracy: 0.9492 - val_loss: 0.2068 - val_accuracy: 0.9219
Epoch 6/100
26/26 [==============================] - 0s 4ms/step - loss: 0.1230 - accuracy: 0.9492 - val_loss: 0.2576 - val_accuracy: 0.9062
Epoch 7/100
26/26 [==============================] - 0s 4ms/step - loss: 0.1241 - accuracy: 0.9531 - val_loss: 0.2688 - val_accuracy: 0.8906
Epoch 8/100
26/26 [==============================] - 0s 4ms/step - loss: 0.1128 - accuracy: 0.9570 - val_loss: 0.2334 - val_accuracy: 0.9219
Epoch 9/100
26/26 [==============================] - 0s 4ms/step - loss: 0.1180 - accuracy: 0.9531 - val_loss: 0.2435 - val_accuracy: 0.9062
Epoch 10/100
```



```
[==============================] - 0s 4ms/step - loss: 0.1139 - accuracy: 0.9531 - val_loss: 0.2799 - val_accuracy: 0.8906 Ep
...
Epoch 99/100 26/26 [==============================] - 0s 3ms/step - loss: 0.1074 - accuracy: 0.9570 - val_loss: 0.2439 - va
[==============================] - 0s 4ms/step - loss: 0.1062 - accuracy: 0.9570 - val_loss: 0.2572 - val_accuracy: 0.9062

<tensorflow.python.keras.callbacks.History at 0x1fdf3ca7b20>
```

## Random Forest model:

A function named random Forest is created and train and test data are passed as the
parameters. Inside the function, Random Forest Classifier algorithm is initialised and
training data is passed to the model with .fit() function. Test data is predicted with.
predict() function and saved in a new variable. For evaluating the model, a confusion
matrix and classification report is done.



```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=10,criterion='entropy')
```

```
rfc.fit(x_train,y_train)
```

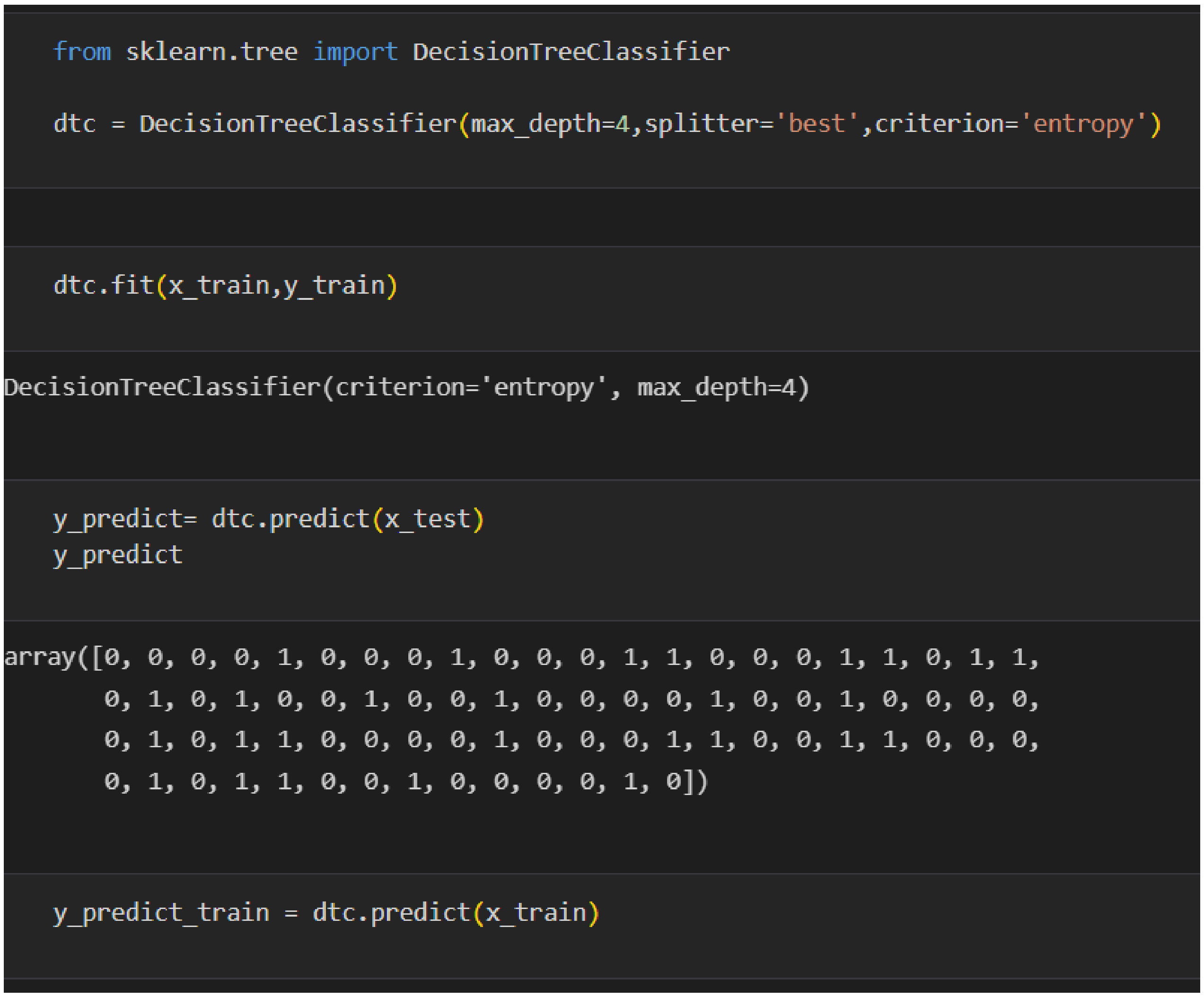
```
<ipython-input-255-b87bb2ba9825>:1: DataConversionWarning: A column-vector y wa
(n_samples,), for example using ravel().
  rfc.fit(x_train,y_train)

RandomForestClassifier(criterion='entropy', n_estimators=10)
```

```
y_predict = rfc.predict(x_test)
```

```
y_predict_train = rfc.predict(x_train)
```

## Decision tree model:

A function named decision Tree is created and train and test data are passed as the parameters. Inside the function, Decision Tree Classifier algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with. predict() function and saved in a new variable. For evaluating the model, a confusionmatrix and classification report is done.



```
from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier(max_depth=4,splitter='best',criterion='entropy')
```

```
dtc.fit(x_train,y_train)
```

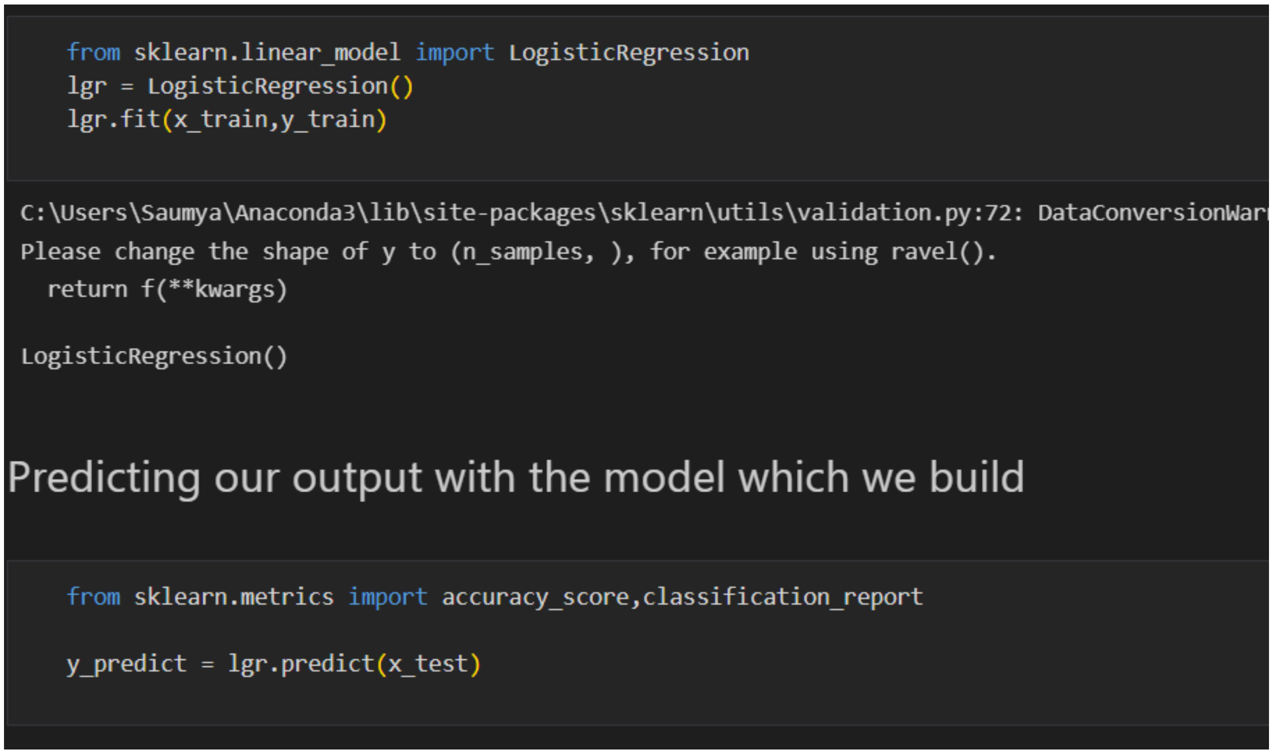
```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

```
y_predict= dtc.predict(x_test)
y_predict
```

```
array([0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1,
       0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0,
       0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0])
```

```
y_predict_train = dtc.predict(x_train)
```

## Logistic Regression:



```
from sklearn.linear_model import LogisticRegression
lgr = LogisticRegression()
lgr.fit(x_train,y_train)
```

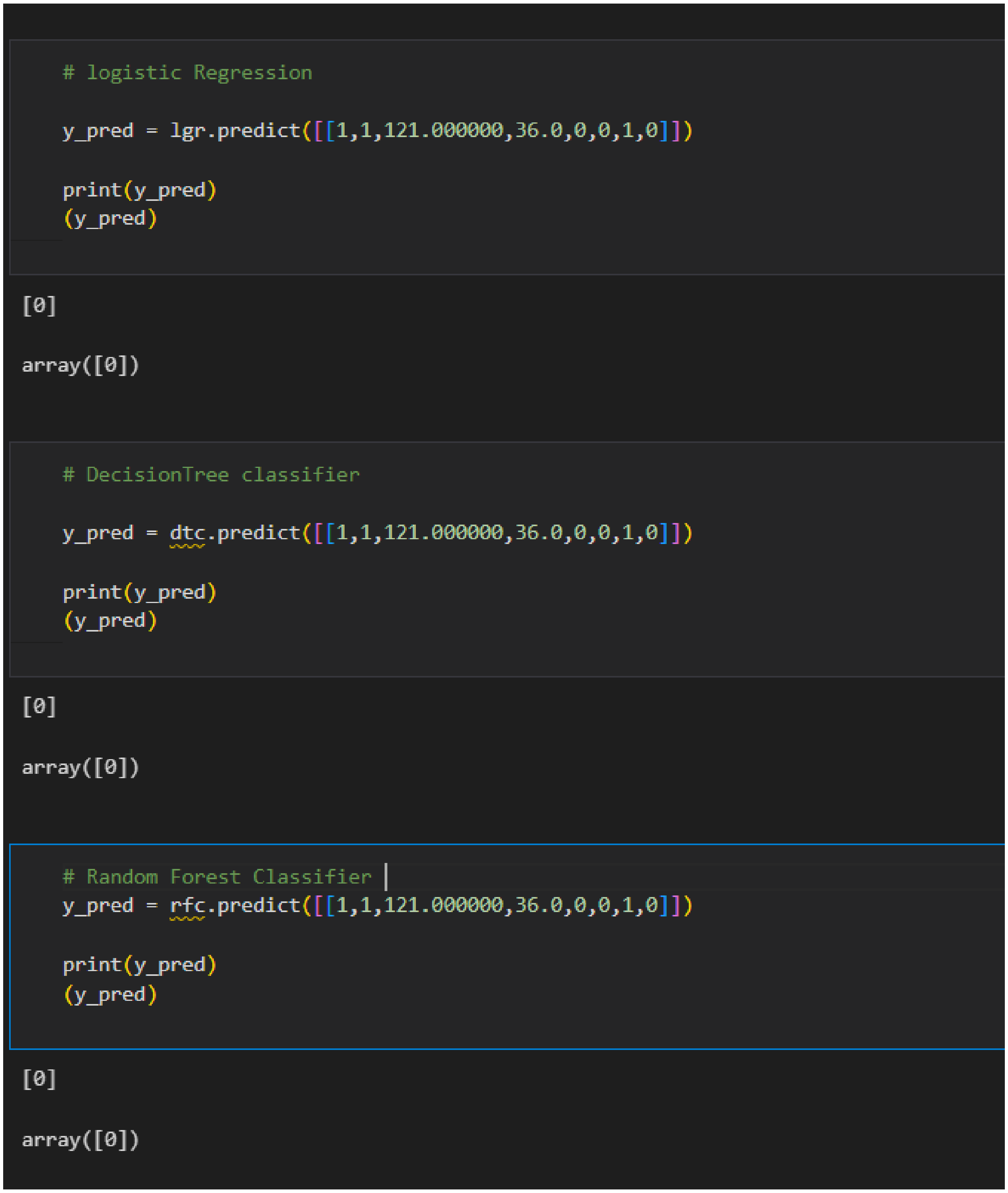
```
C:\Users\Saumya\Anaconda3\lib\site-packages\sklearn\utils\validation.py:72: DataConversionWar
Please change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

LogisticRegression()
```

Predicting our output with the model which we build

```
from sklearn.metrics import accuracy_score,classification_report

y_predict = lgr.predict(x_test)
```

## Testing the model:

 In ANN we first have to save the model to the test the inputs



```
# logistic Regression

y_pred = lgr.predict([[1,1,121.000000,36.0,0,0,1,0]])

print(y_pred)
(y_pred)
```

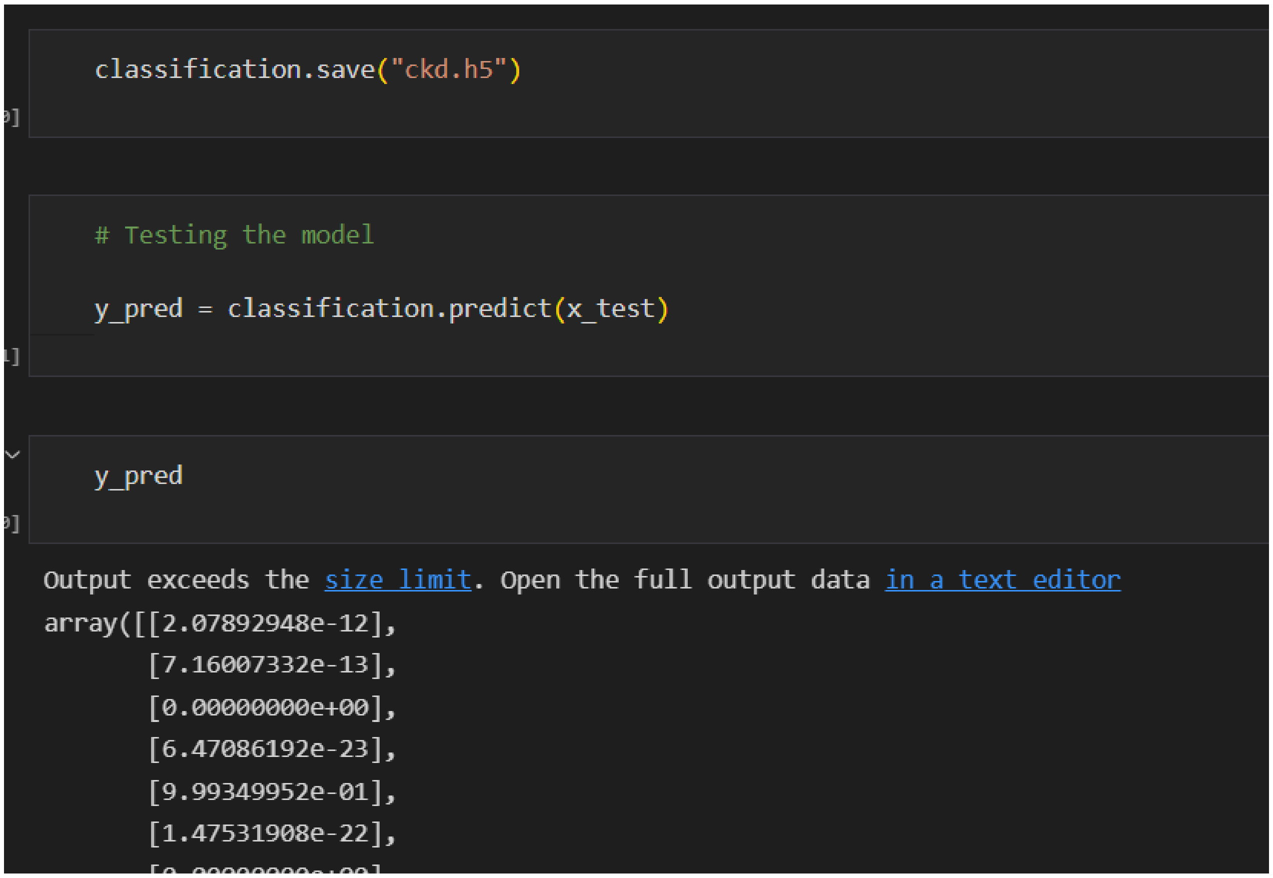
```
[0]

array([0])
```

```
# DecisionTree classifier

y_pred = dtc.predict([[1,1,121.000000,36.0,0,0,1,0]])

print(y_pred)
(y_pred)
```

```
[0]

array([0])
```

```
# Random Forest Classifier
y_pred = rfc.predict([[1,1,121.000000,36.0,0,0,1,0]])

print(y_pred)
(y_pred)
```

```
[0]

array([0])
```



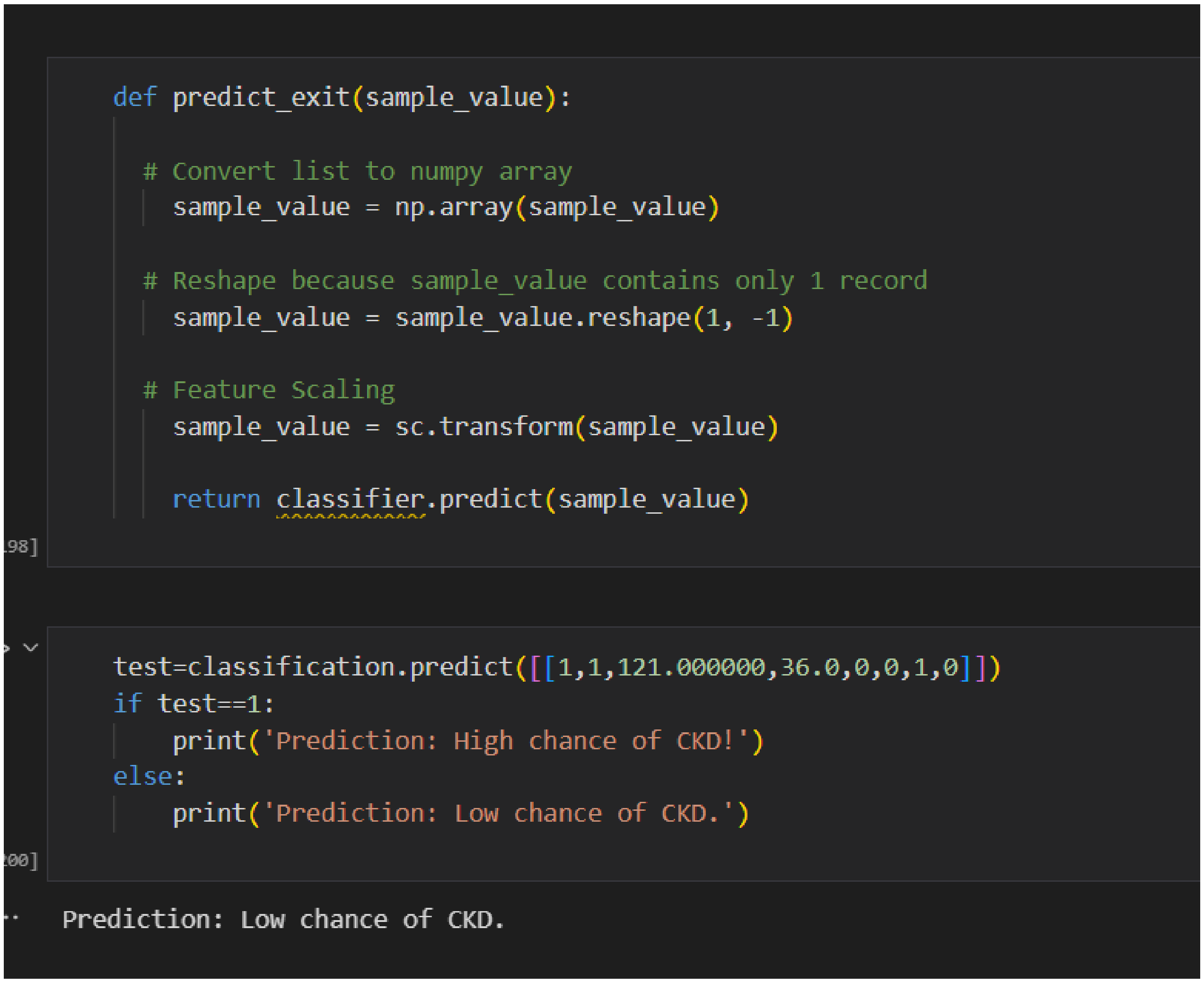
```
classification.save("ckd.h5")
```

```
# Testing the model

y_pred = classification.predict(x_test)
```

```
y_pred
```

```
Output exceeds the size limit. Open the full output data in a text editor
array([[2.07892948e-12],
       [7.16007332e-13],
       [0.00000000e+00],
       [6.47086192e-23],
       [9.99349952e-01],
       [1.47531908e-22],
```

This code defines a function named "predict_exit" which takes in a sample_value as an input. The function then converts the input sample_value from a list to a numpy array. It reshapes the sample_value array as it contains only one record. Then, it applies feature scaling to the reshaped sample_value array using a scaler object 'sc' that should have been previously defined and fitted. Finally, the function returns the prediction of the classifier on the scaled sample_value.



```python
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)
```

```python
test=classification.predict([[1,1,121.000000,36.0,0,0,1,0]])
if test==1:
    print('Prediction: High chance of CKD!')
else:
    print('Prediction: Low chance of CKD.')
```

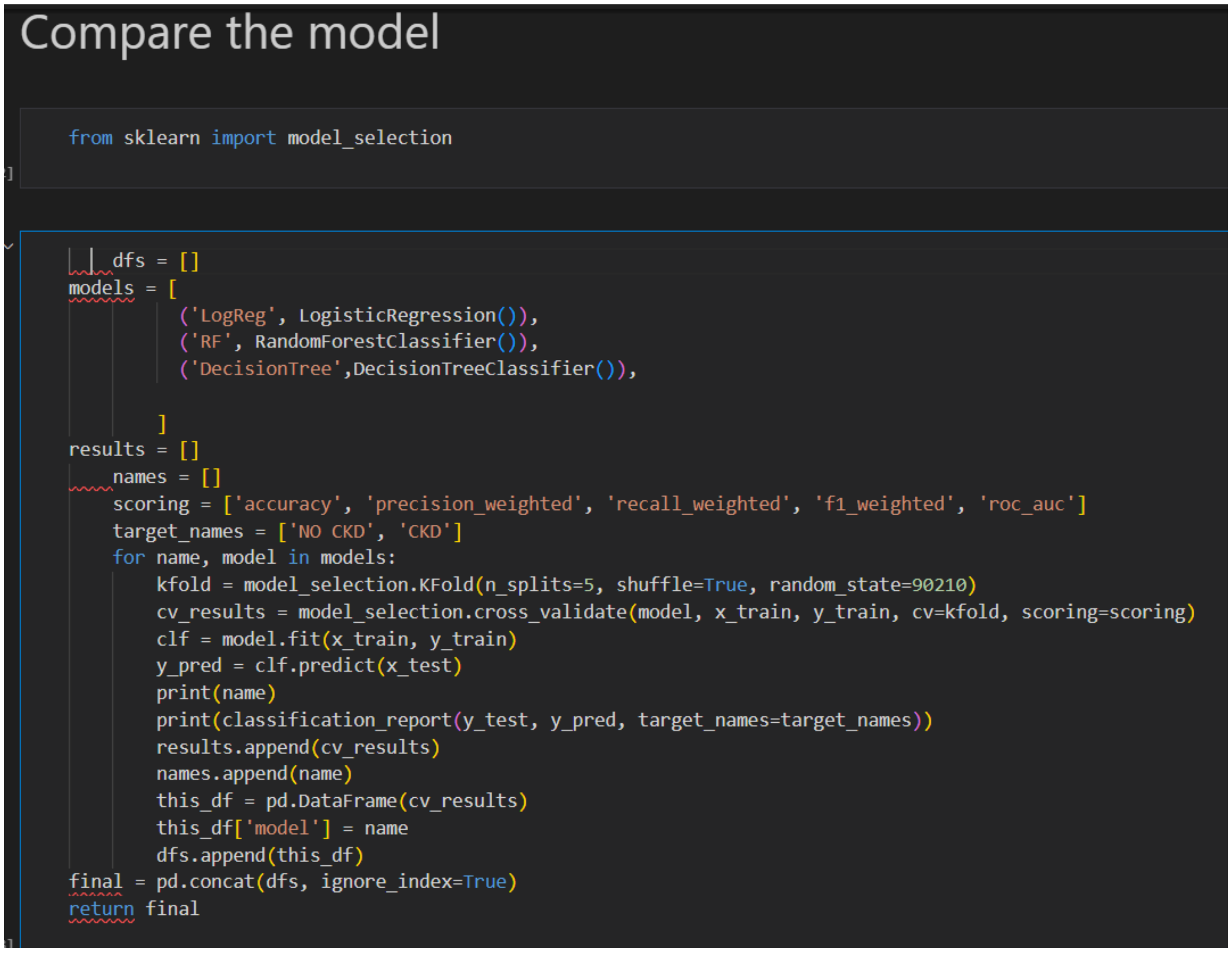
```
Prediction: Low chance of CKD.
```

## Performance Testing & Evaluate the results

### Testing model with multiple evaluation metrics

Multiple evaluation metrics means evaluating the model's performance on a test set
using different performance measures. This can provide a more comprehensive
understanding of the model's strengths and weaknesses. We are using evaluation

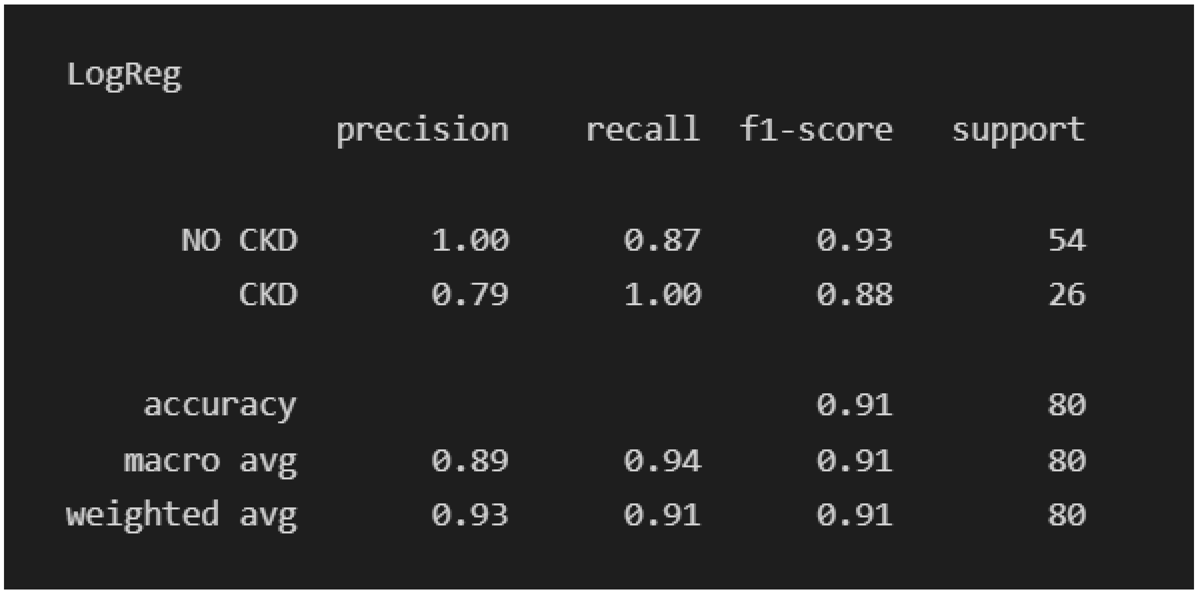
metrics for classification tasks including accuracy, precision, recall, support and
F1-score.



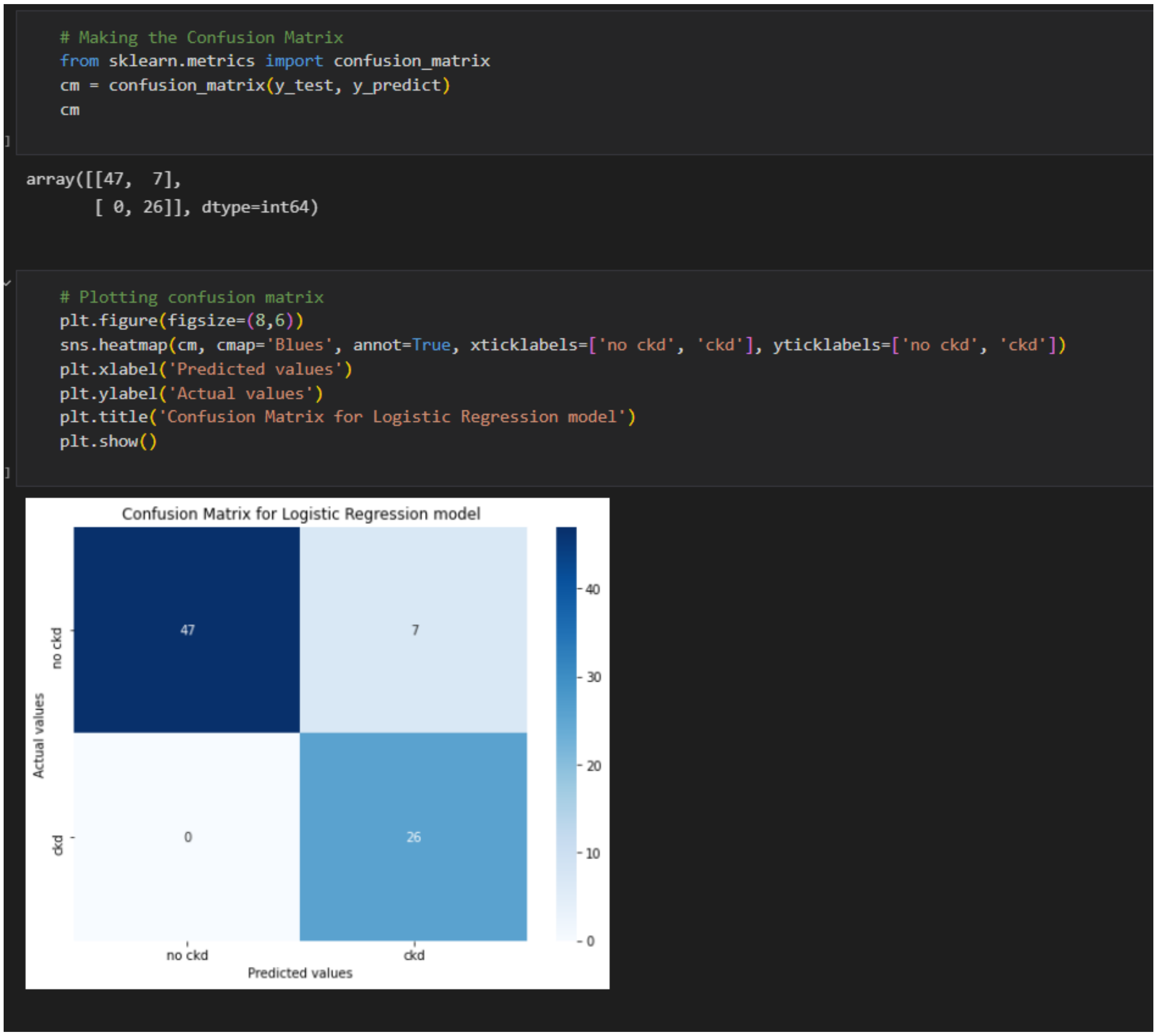
## Compare the model

```
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_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
```



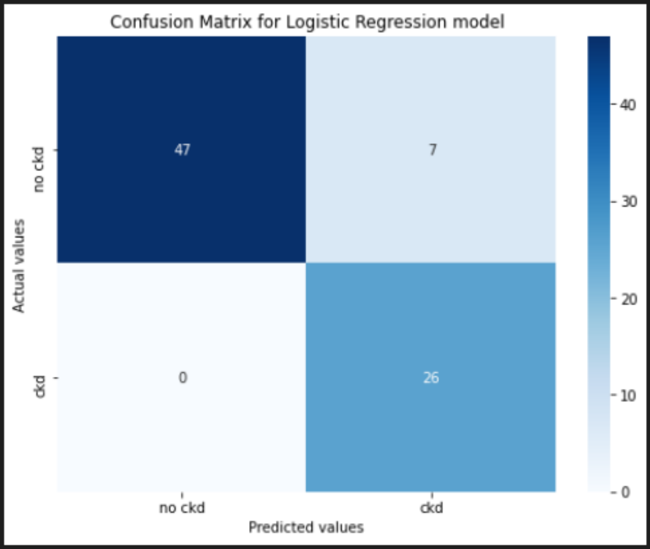
```
LogReg
              precision    recall  f1-score   support

      NO CKD       1.00      0.87      0.93        54
         CKD       0.79      1.00      0.88        26

    accuracy                           0.91        80
   macro avg       0.89      0.94      0.91        80
weighted avg       0.93      0.91      0.91        80
```



```python
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_predict)
cm
```

```
array([[47,  7],
       [ 0, 26]], dtype=int64)
```

```python
# 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 Logistic Regression model')
plt.show()
```

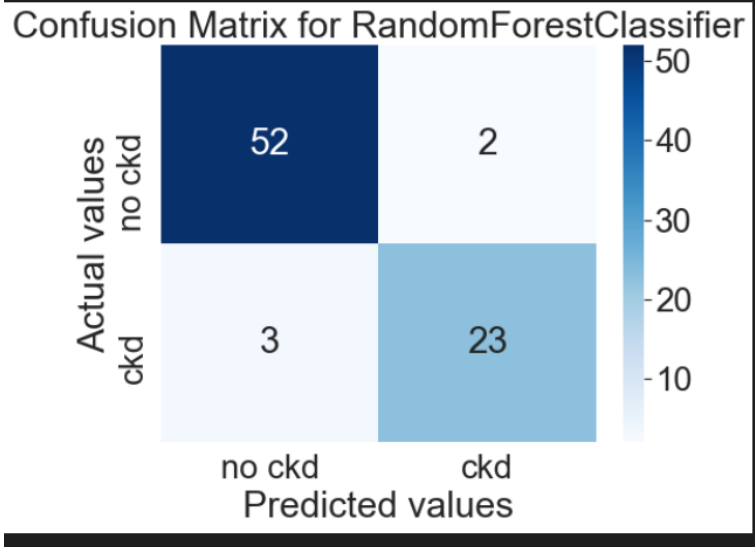


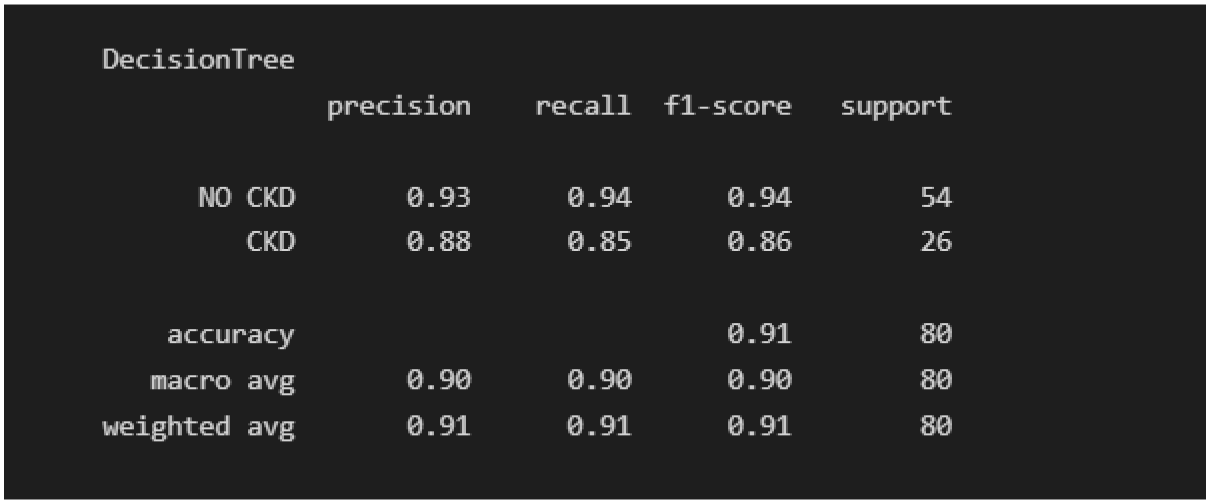
```
RF
              precision    recall  f1-score   support

      NO CKD       0.96      0.96      0.96        54
         CKD       0.92      0.92      0.92        26

    accuracy                           0.95        80
   macro avg       0.94      0.94      0.94        80
weighted avg       0.95      0.95      0.95        80
```

```python
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_predict)
cm
```

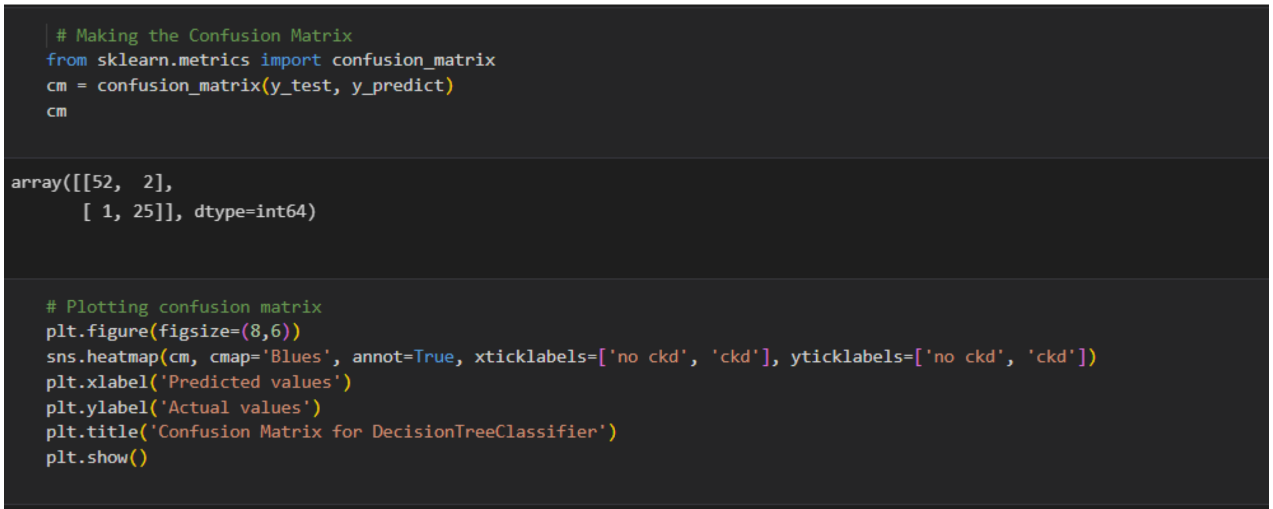
```
array([[52,  2],
       [ 3, 23]], dtype=int64)
```

```python
# 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()
```





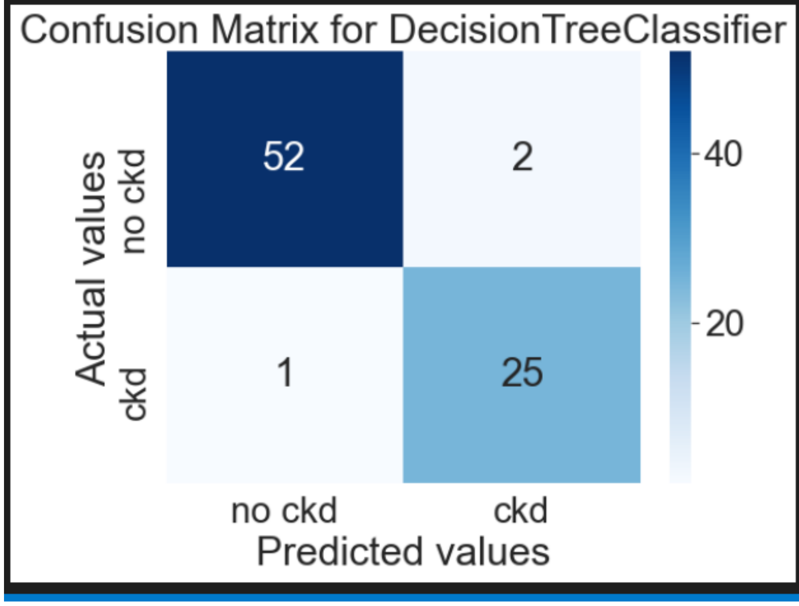
```
     DecisionTree
                  precision    recall  f1-score   support

          NO CKD       0.93      0.94      0.94        54
             CKD       0.88      0.85      0.86        26

        accuracy                           0.91        80
       macro avg       0.90      0.90      0.90        80
    weighted avg       0.91      0.91      0.91        80
```



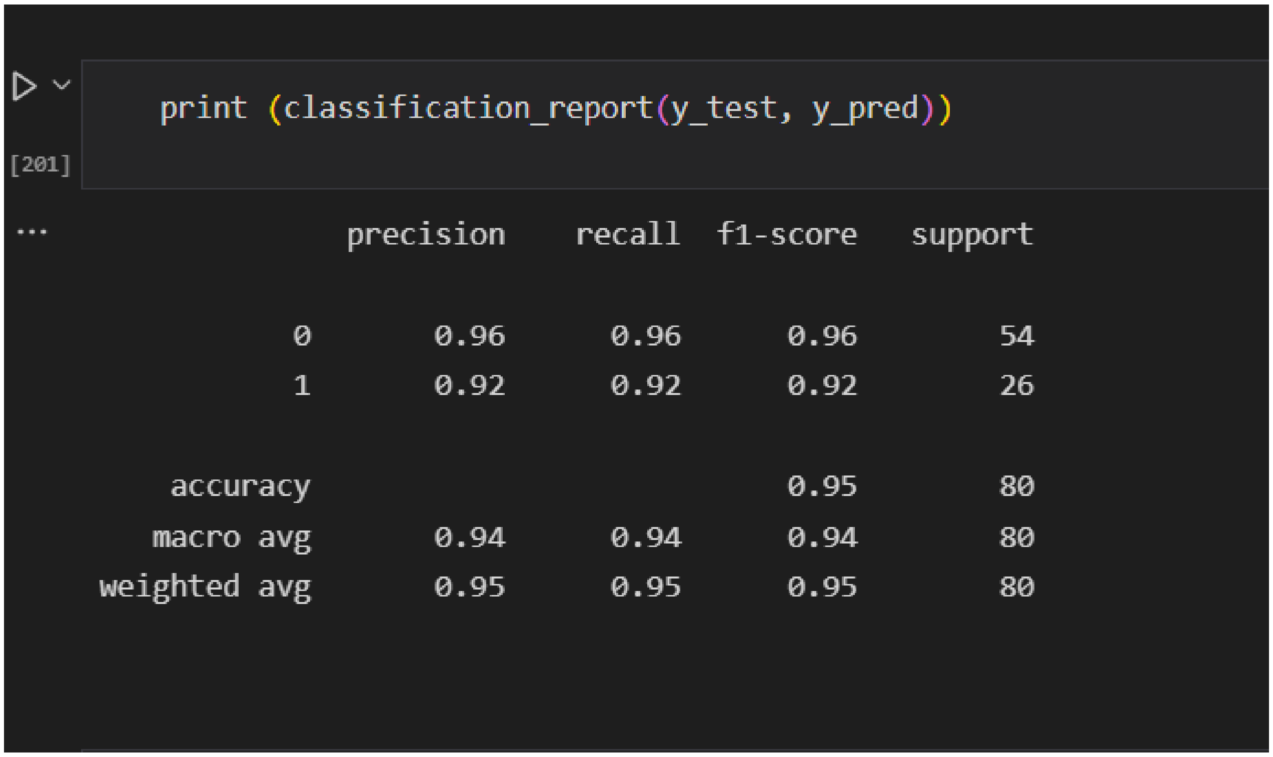
```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_predict)
cm
```

```
array([[52,  2],
       [ 1, 25]], dtype=int64)
```

```
# 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()
```

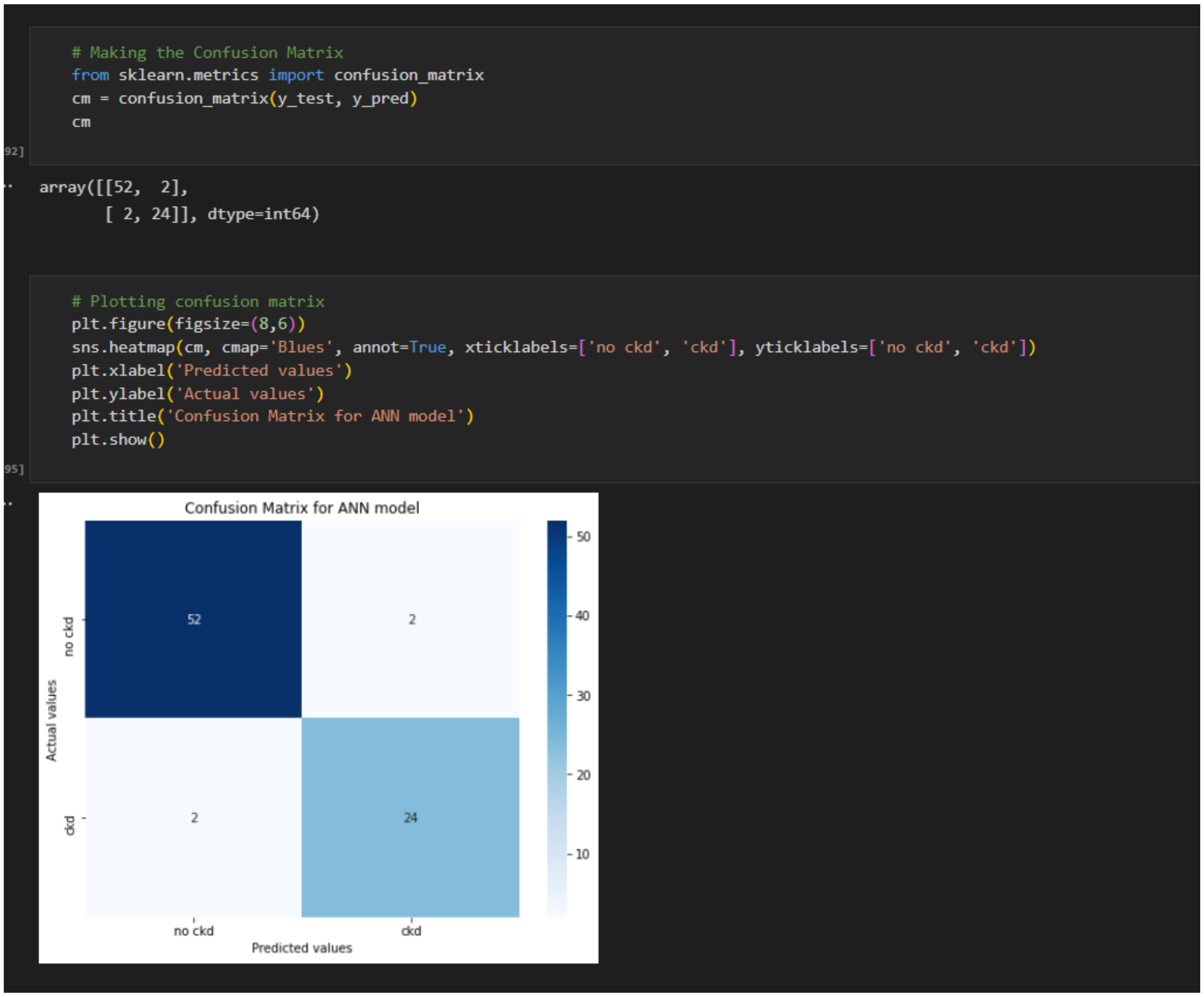


## For ANN



```
[201]
print (classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.96      0.96        54
           1       0.92      0.92      0.92        26

    accuracy                           0.95        80
   macro avg       0.94      0.94      0.94        80
weighted avg       0.95      0.95      0.95        80
```

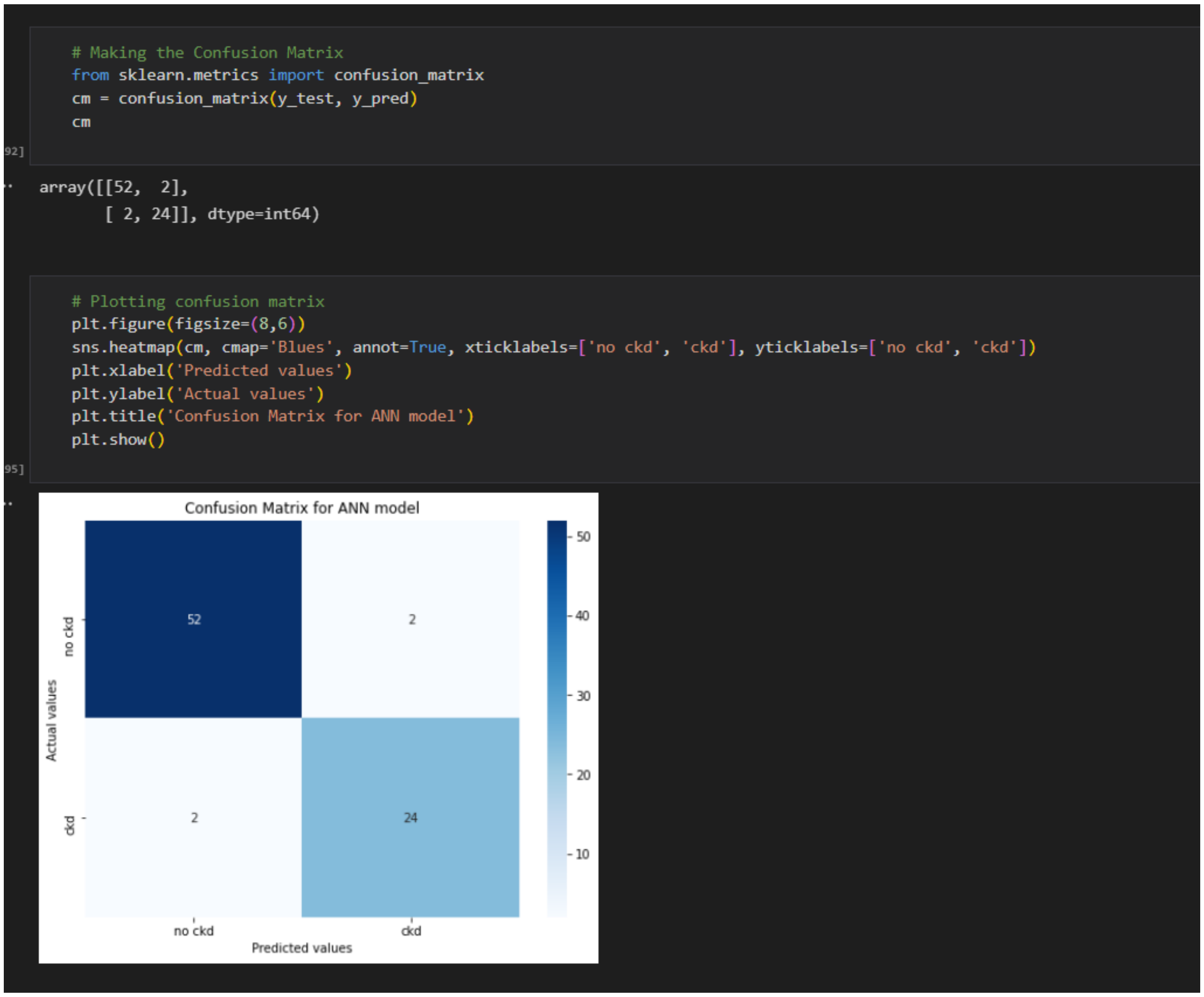


```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[52,  2],
       [ 2, 24]], dtype=int64)
```

```
# 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()
```

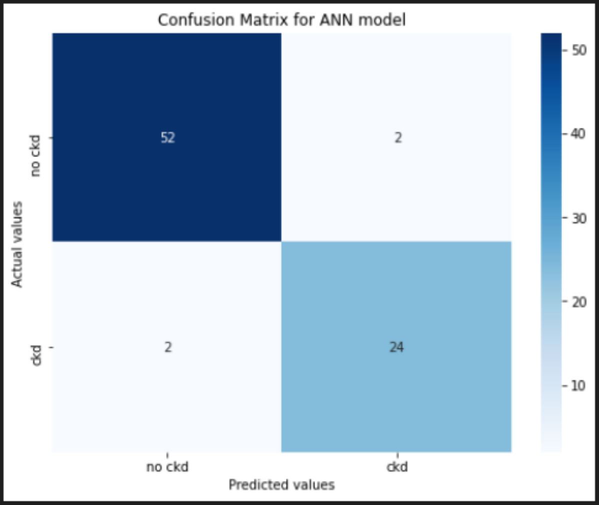
All above models are performing well for this dataset.



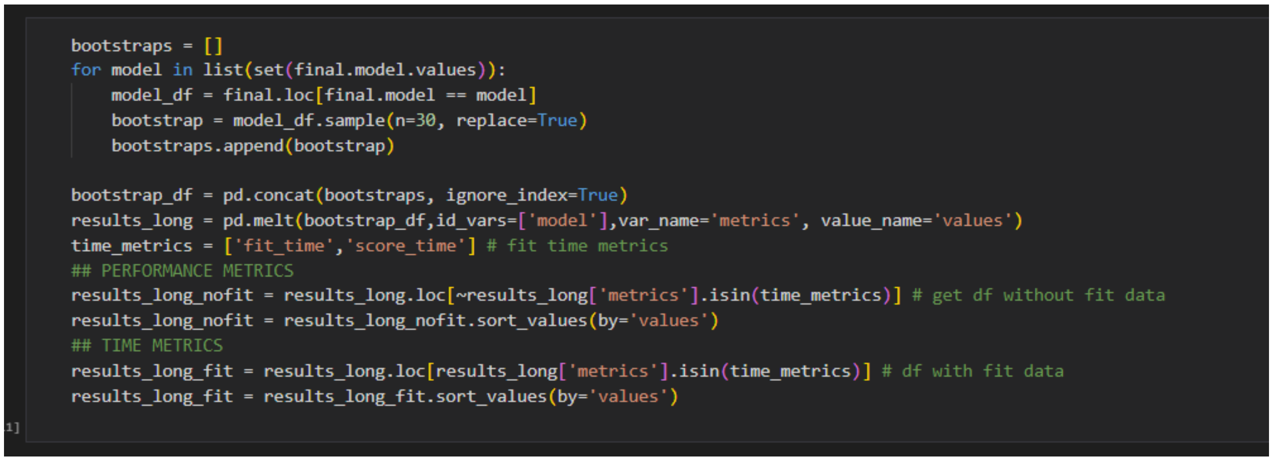
```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[52,  2],
       [ 2, 24]], dtype=int64)
```

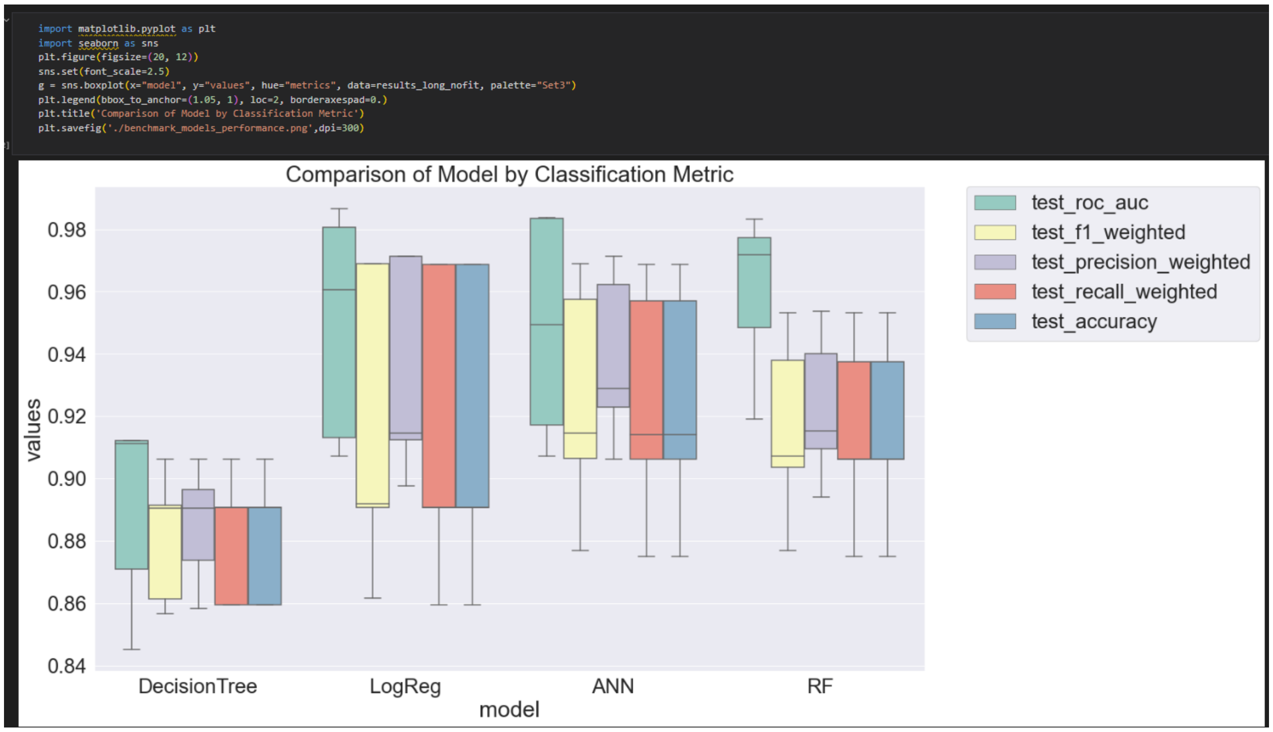
```
# Plotting confusion matrix
plt.figure(figsize=(8,6))
sns.heatmap(cm, cmap='Blues', annot=True, xticklabels=['no ckd', 'ckd'], yticklabels=['no ckd', 'ckd'])
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.title('Confusion Matrix for ANN model')
plt.show()
```



## Evaluate the results:



```
bootstraps = []
for model in list(set(final.model.values)):
    model_df = final.loc[final.model == model]
    bootstrap = model_df.sample(n=30, replace=True)
    bootstraps.append(bootstrap)

bootstrap_df = pd.concat(bootstraps, ignore_index=True)
results_long = pd.melt(bootstrap_df,id_vars=['model'],var_name='metrics', value_name='values')
time_metrics = ['fit_time','score_time'] # fit time metrics
## PERFORMANCE METRICS
results_long_nofit = results_long.loc[~results_long['metrics'].isin(time_metrics)] # get df without fit data
results_long_nofit = results_long_nofit.sort_values(by='values')
## TIME METRICS
results_long_fit = results_long.loc[results_long['metrics'].isin(time_metrics)] # df with fit data
results_long_fit = results_long_fit.sort_values(by='values')
```



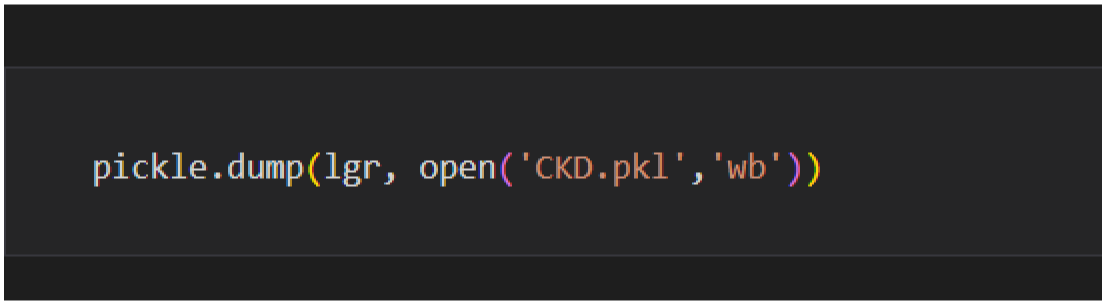
```python
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(20, 12))
sns.set(font_scale=2.5)
g = sns.boxplot(x="model", y="values", hue="metrics", data=results_long_nofit, palette="Set3")
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.title('Comparison of Model by Classification Metric')
plt.savefig('./benchmark_models_performance.png',dpi=300)
```

**A**mong all these 4 models logistic regression
has recall 1. So, we are going for logreg model.

## Model Deployment:

### Save the best model:

Saving the best model after comparing its
performance using different evaluation
metrics means selecting the model with the highest
performance and saving its
weights and configuration. This can be useful in
avoiding the need to retrain the
model every time it is needed and to be able to use it
in the future.



```
pickle.dump(lgr, open('CKD.pkl','wb'))
```

**Integrate with Web Framework:**

In this section, we will be building a web application
that is integrated to the model we
built. A UI is provided for the uses where he has to
enter the values for predictions.
The enter values are given to the saved model and
prediction is showcased on the
UI.This section has the following tasks

- Building HTML Pages
- Building server-side script
- Run the web application

**Building Html Pages:**

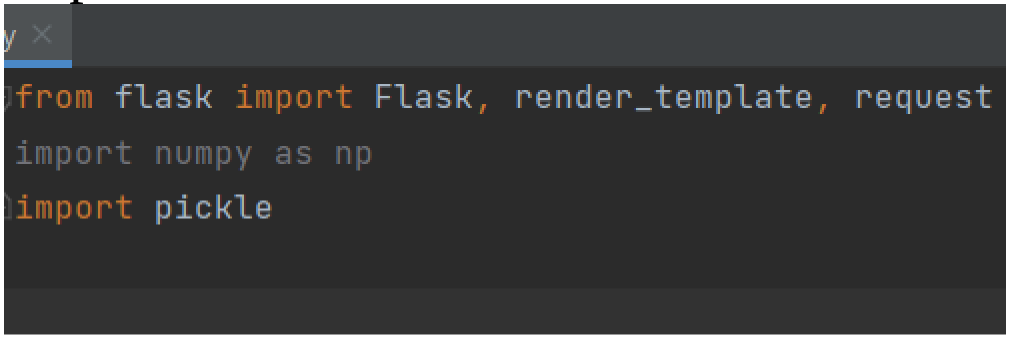
For this project create four HTML files namely

- home.html
- index1.html
- indexnew.html

● result.html
and save them in the templates folder.

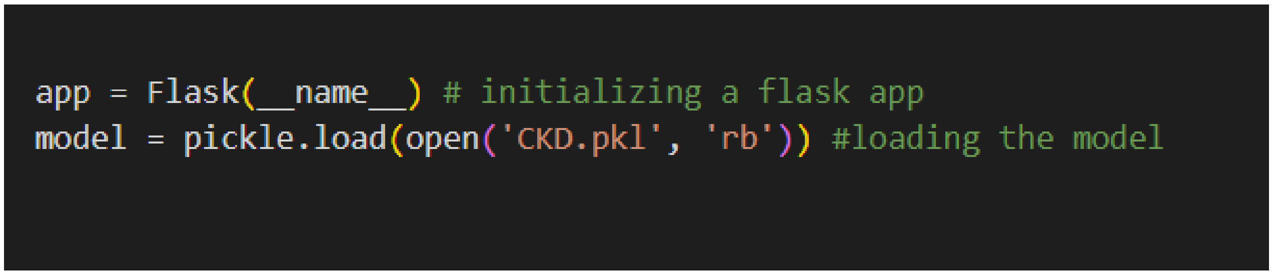
**Build Python code:**

Import the libraries:



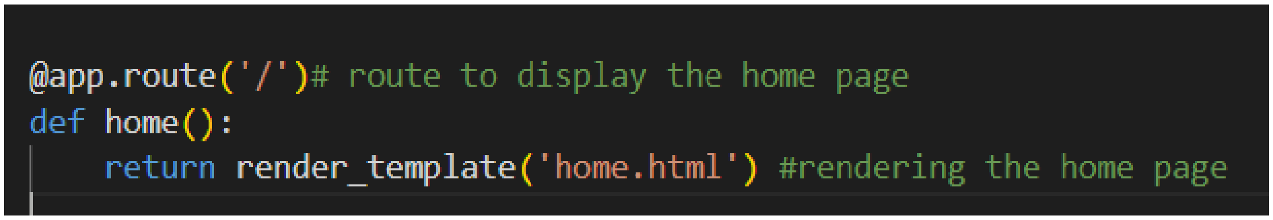
```
from flask import Flask, render_template, request
import numpy as np
import pickle
```

Load the saved model. Importing the flask module in the project is mandatory. An
object of Flask class is our WSGI application. Flask constructor takes the name of the
current module (__name__) as argument.



```
app = Flask(__name__) # initializing a flask app
model = pickle.load(open('CKD.pkl', 'rb')) #loading the model
```

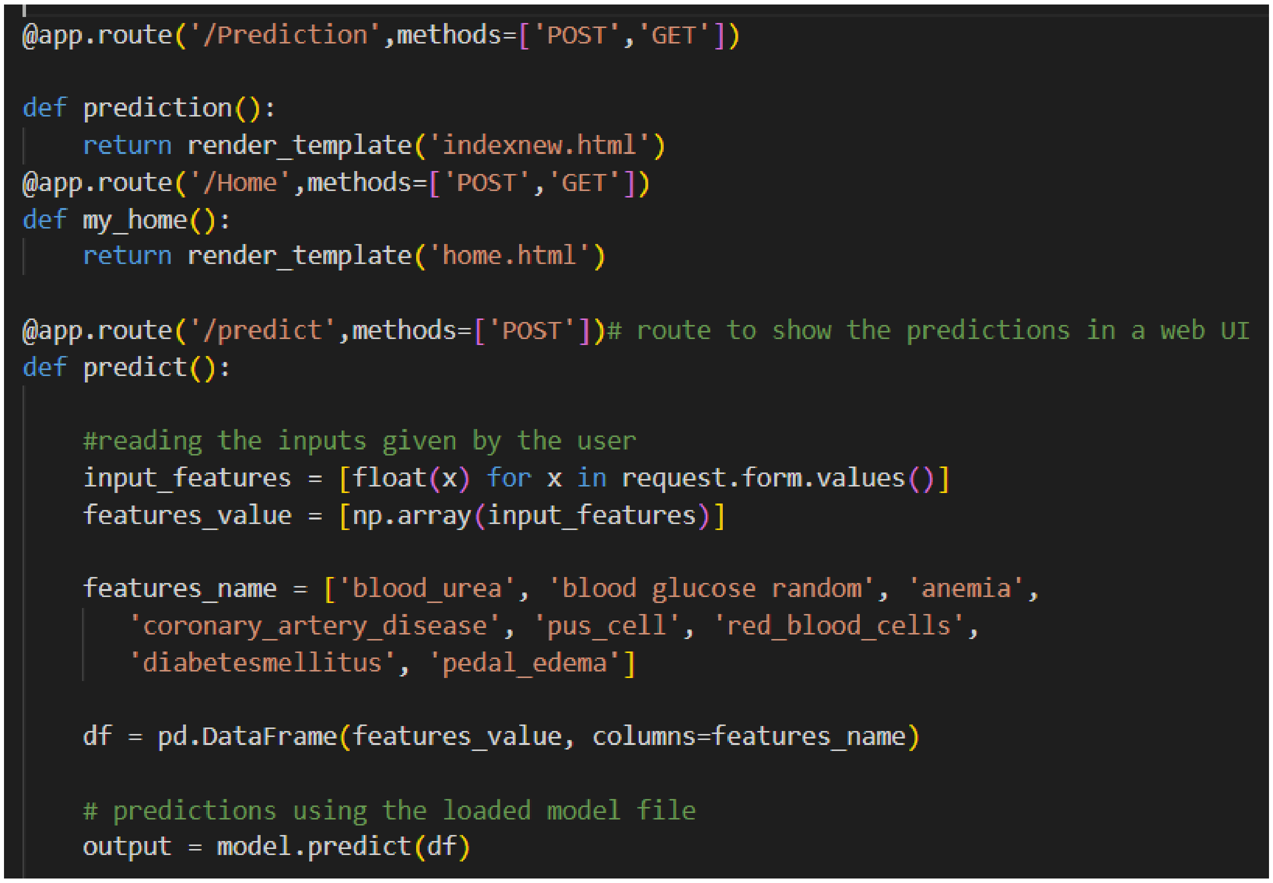
Render HTML page:



```
@app.route('/')# route to display the home page
def home():
    return render_template('home.html') #rendering the home page
```

Here we will be using a declared constructor to route to the HTML page which we
have created earlier.
In the above example, '/' URL is bound with the home.html function. Hence, when the
home page of the web server is opened in the browser, the html page will berendered. Whenever you enter the values from the html page the values can be
retrieved using POST Method.

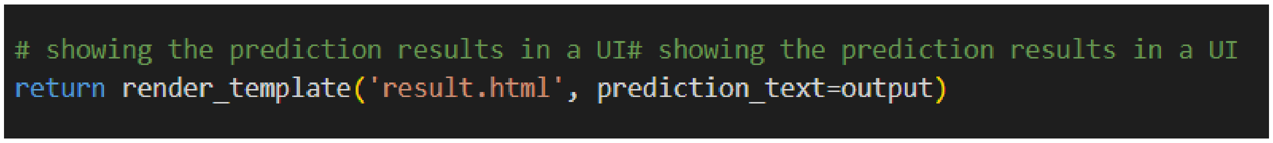
Retrieves the value from UI:



```python
@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'])# route to show the predictions in a web UI
def predict():

    #reading the inputs given by the user
    input_features = [float(x) for x in request.form.values()]
    features_value = [np.array(input_features)]

    features_name = ['blood_urea', 'blood glucose random', 'anemia',
       'coronary_artery_disease', 'pus_cell', 'red_blood_cells',
       'diabetesmellitus', 'pedal_edema']

    df = pd.DataFrame(features_value, columns=features_name)

    # predictions using the loaded model file
    output = model.predict(df)
```

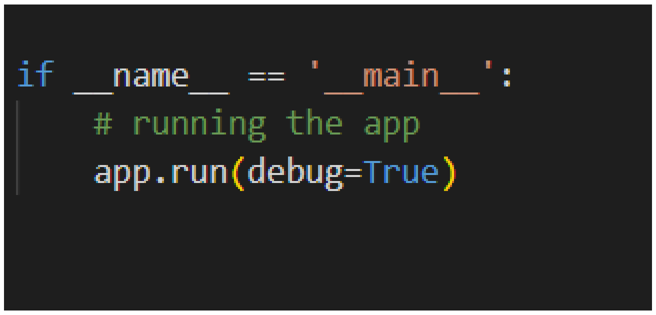
Here we are routing our app to predict() function. This function retrieves all the values

from the HTML page using Post request. That is stored in an array. This array is
passed to the model.predict() function. This function returns the prediction. And this
prediction value will be rendered to the text that we have mentioned in the
submit.html page earlier.



```python
# showing the prediction results in a UI# showing the prediction results in a UI
return render_template('result.html', prediction_text=output)
```

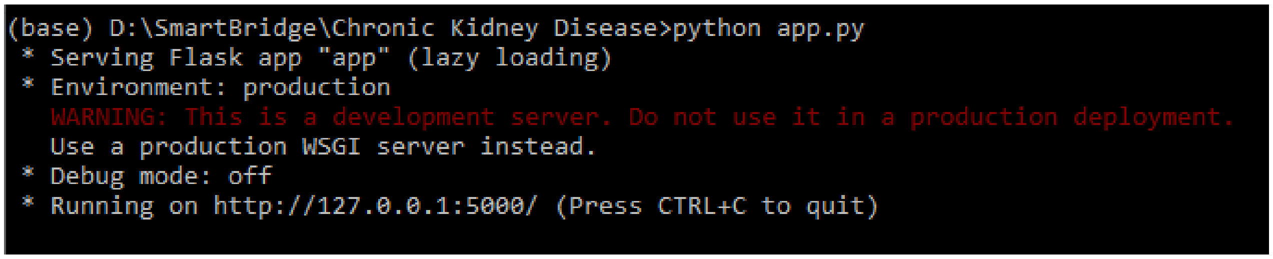
Main Function:



```python
if __name__ == '__main__':
    # running the app
    app.run(debug=True)
```

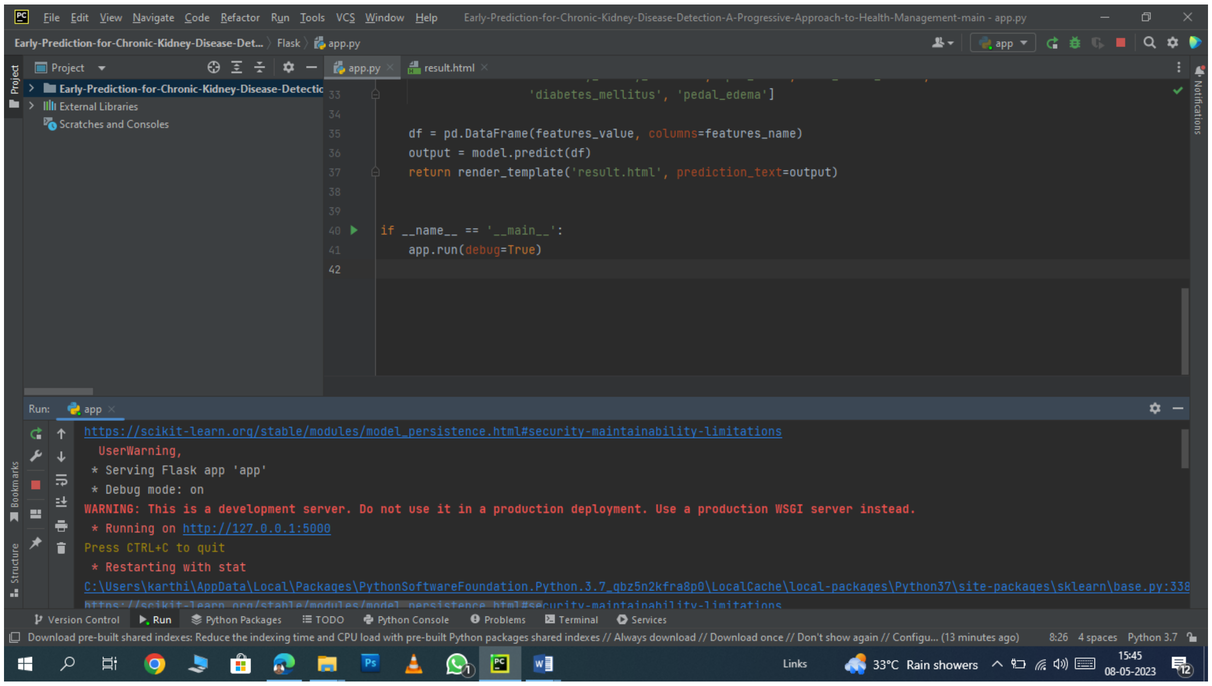
**Run the web application**
● Open anaconda prompt from the start menu
● Navigate to the folder where your python script is.
● Now type "python app.py" command●
Navigate to the localhost where you can view your web page.
● Click on the predict button from the top left corner, enter the inputs,

click on the submit button, and see the result/prediction on the web.



```
(base) D:\SmartBridge\Chronic Kidney Disease>python app.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

## Running the Coding of Kidney disease:



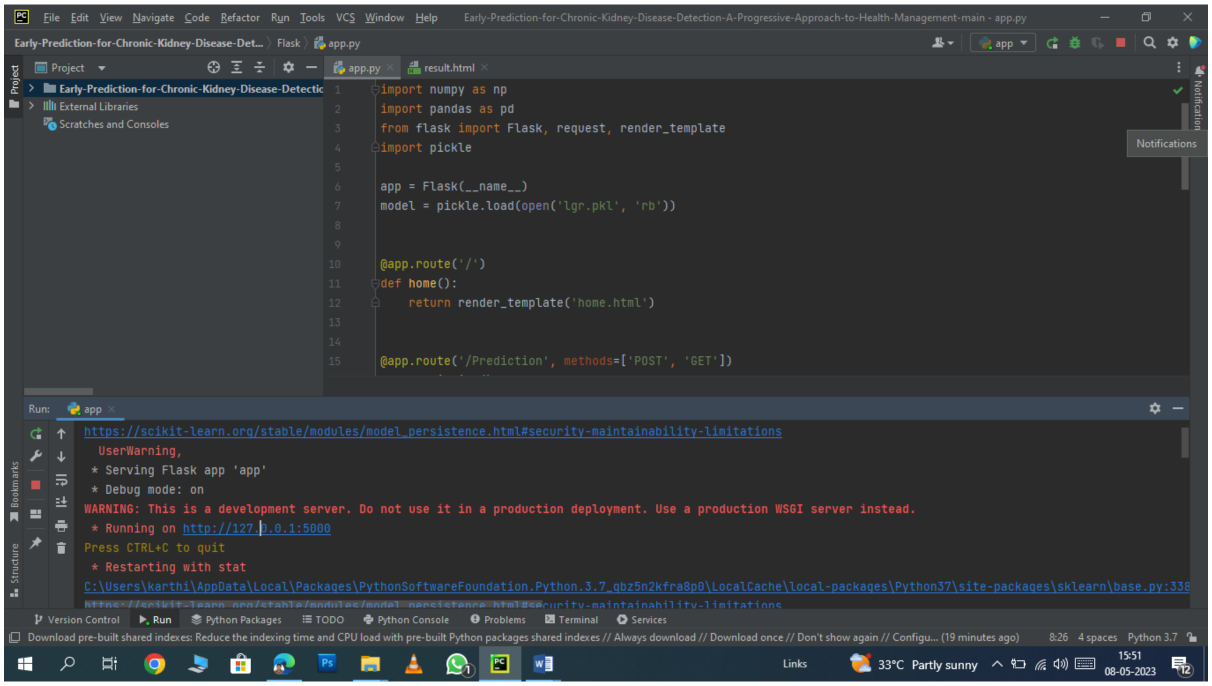
File Edit View Navigate Code Refactor Run Tools VCS Window Help
Early-Prediction-for-Chronic-Kidney-Disease-Detection-A-Progressive-Approach-to-Health-Management-main - app.py
Early-Prediction-for-Chronic-Kidney-Disease-Det... 〉 Flask 〉 app.py
Project
Early-Prediction-for-Chronic-Kidney-Disease-Detectic
External Libraries
Scratches and Consoles
app.py
result.html

```python
                    'diabetes_mellitus', 'pedal_edema']

    df = pd.DataFrame(features_value, columns=features_name)
    output = model.predict(df)
    return render_template('result.html', prediction_text=output)


if __name__ == '__main__':
    app.run(debug=True)
```

```
https://scikit-learn.org/stable/modules/model_persistence.html#security-maintainability-limitations
  UserWarning,
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
C:\Users\karthi\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.7_qbz5n2kfra8p0\LocalCache\local-packages\Python37\site-packages\sklearn\base.py:338
```

Version Control
Run
Python Packages
TODO
Python Console
Problems
Terminal
Services
Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built Python packages shared indexes // Always download // Download once // Don't show again // Configu... (13 minutes ago)
8:26 4 spaces Python 3.7
Links
33°C Rain showers
15:45
08-05-2023

Main.py is visible:

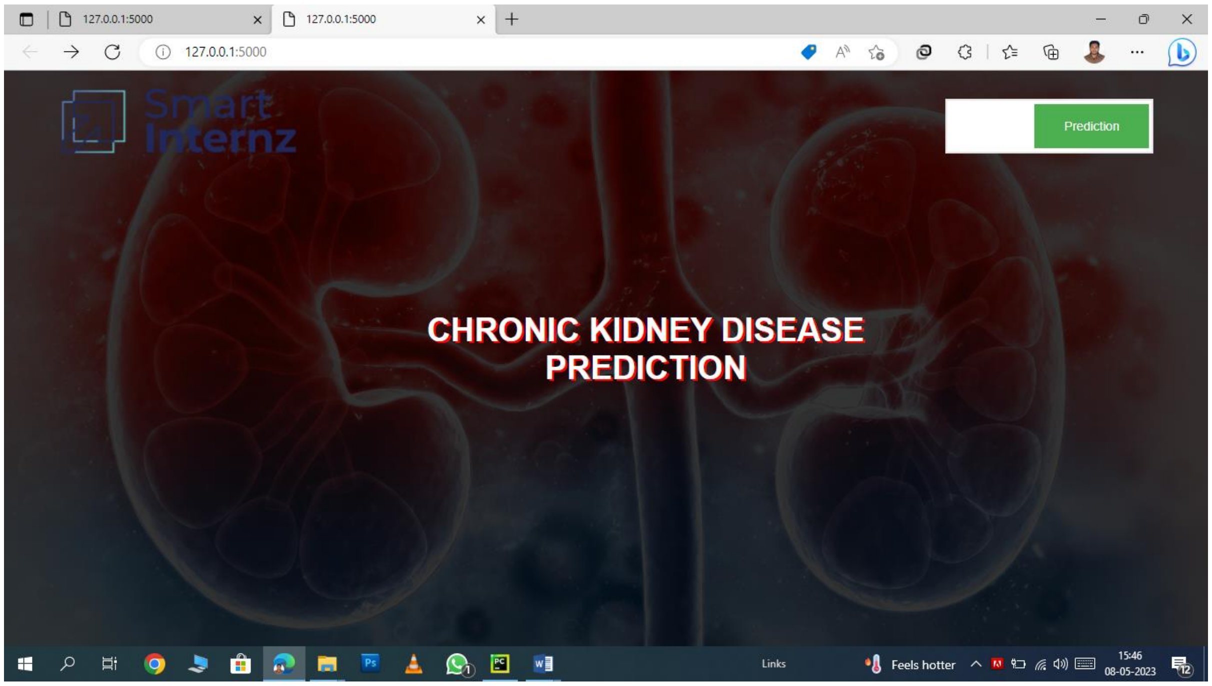


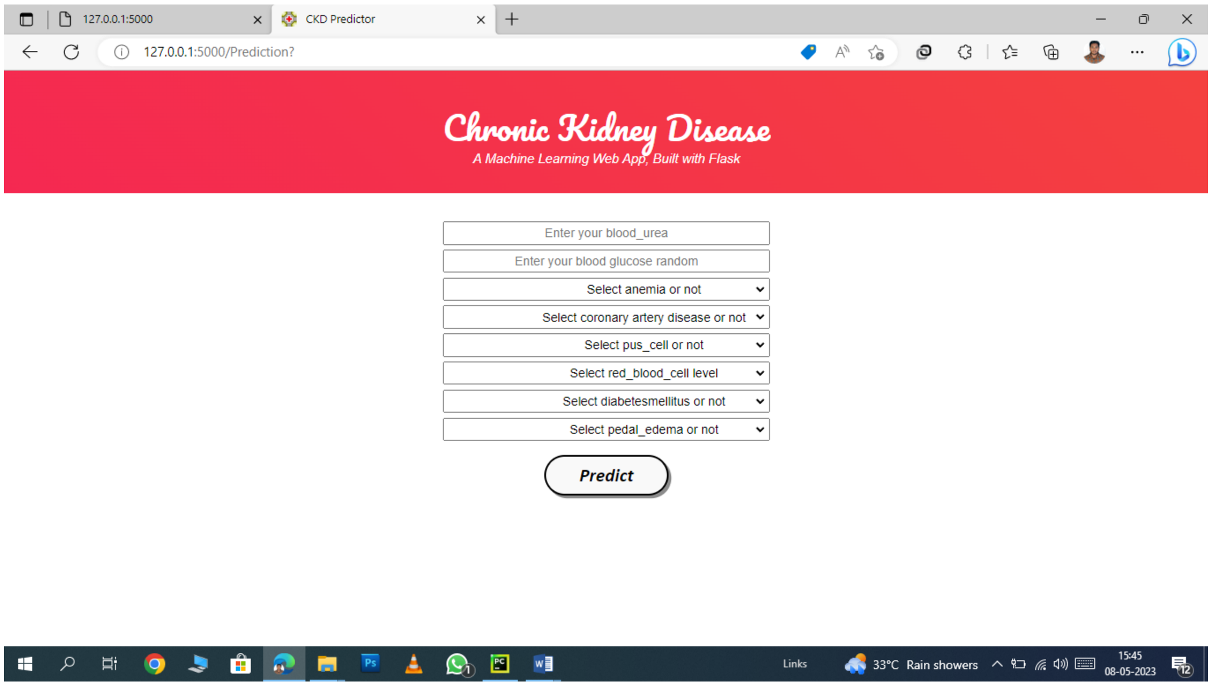
```python
import numpy as np
import pandas as pd
from flask import Flask, request, render_template
import pickle

app = Flask(__name__)
model = pickle.load(open('lgr.pkl', 'rb'))


@app.route('/')
def home():
    return render_template('home.html')


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

```
https://scikit-learn.org/stable/modules/model_persistence.html#security-maintainability-limitations
  UserWarning,
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
C:\Users\karthi\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.7_qbz5n2kfra8p0\LocalCache\local-packages\Python37\site-packages\sklearn\base.py:338
https://scikit-learn.org/stable/modules/model_persistence.html#security-maintainability-limitations
```

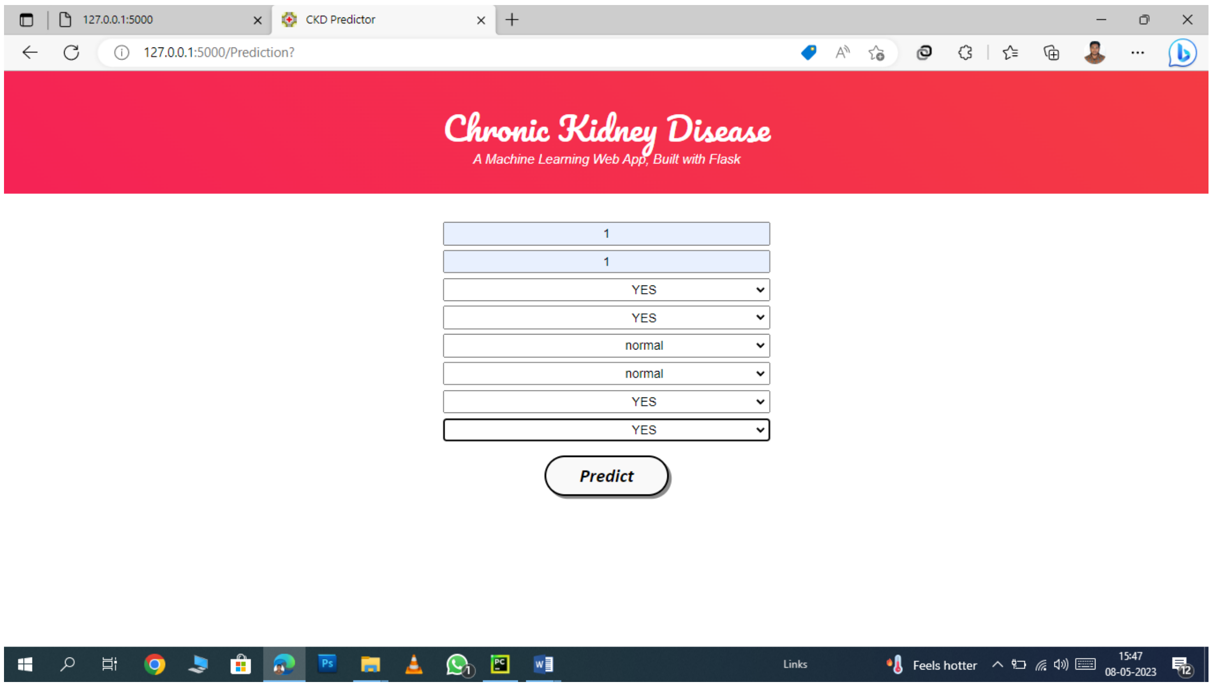
Now,Go the web browser and write the localhost url (http://127.0.0.1:5000) to get the
below result

## Home Page Of The Kidney Disease Project:

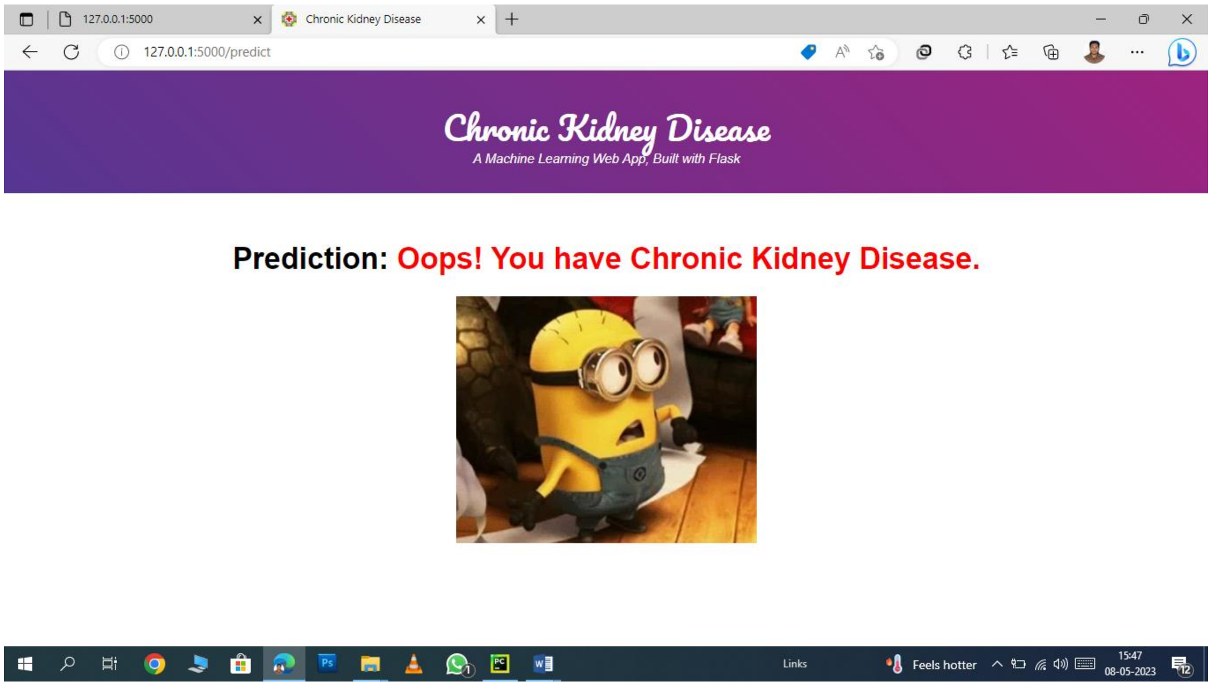

127.0.0.1:5000
127.0.0.1:5000
Smart Internz
Prediction
CHRONIC KIDNEY DISEASE PREDICTION
Links
Feels hotter
15:46
08-05-2023


127.0.0.1:5000
CKD Predictor
127.0.0.1:5000/Prediction?
Chronic Kidney Disease
A Machine Learning Web App, Built with Flask
Enter your blood_urea
Enter your blood glucose random
Select anemia or not
Select coronary artery disease or not
Select pus_cell or not
Select red_blood_cell level
Select diabetesmellitus or not
Select pedal_edema or not
Predict
Links
33°C Rain showers
15:45
08-05-2023

Input - Now, the user will give inputs to get the predicted result after clicking onto the submit button.



Out will be excuted.



127.0.0.1:5000 | Chronic Kidney Disease

127.0.0.1:5000/predict

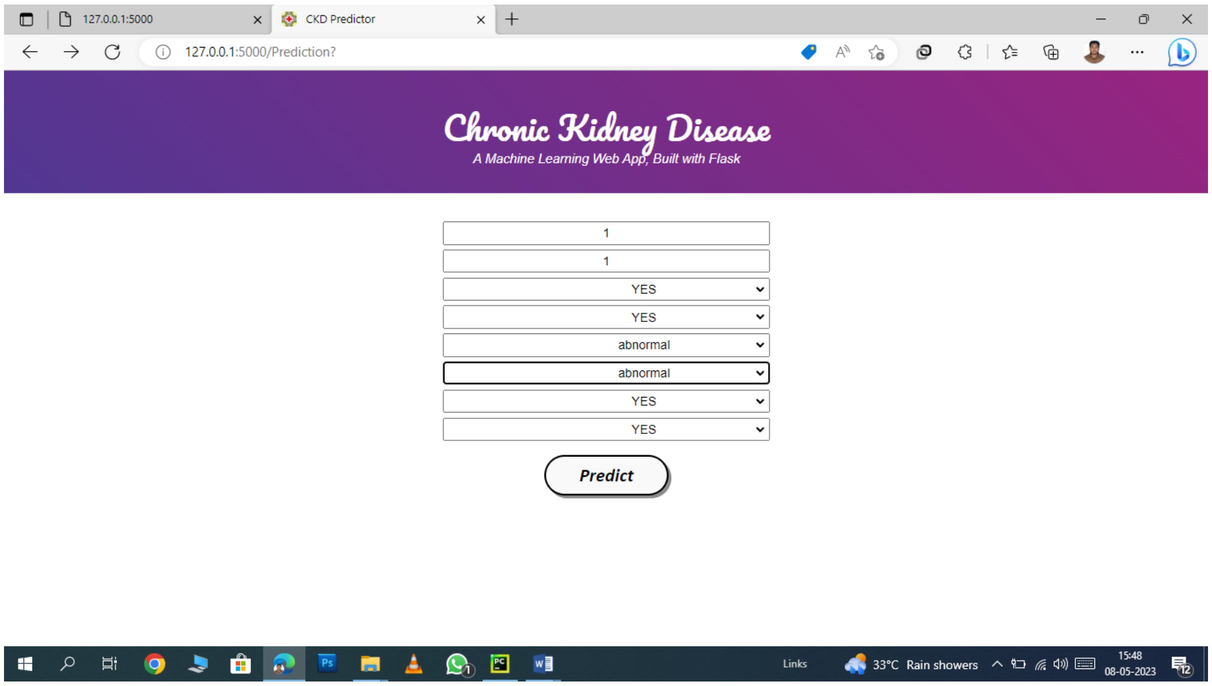
**Chronic Kidney Disease**

*A Machine Learning Web App, Built with Flask*

**Prediction: Oops! You have Chronic Kidney Disease.**

Links | Feels hotter | 15:47 08-05-2023

Another Input Also Given:



127.0.0.1:5000 | CKD Predictor

127.0.0.1:5000/Prediction?

**Chronic Kidney Disease**

*A Machine Learning Web App, Built with Flask*

1

1

YES

YES

abnormal

abnormal

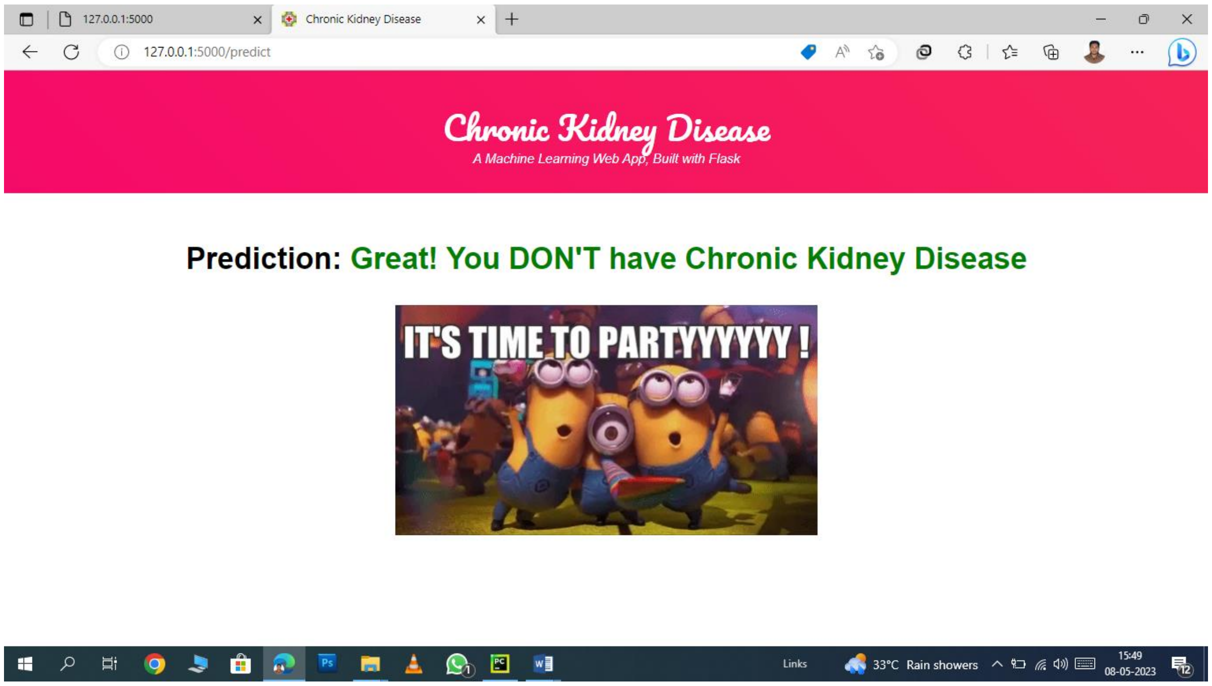
YES

YES

Predict

Links | 33°C Rain showers | 15:48 08-05-2023

Output There,,



127.0.0.1:5000
Chronic Kidney Disease
127.0.0.1:5000/predict

Chronic Kidney Disease
A Machine Learning Web App, Built with Flask

Prediction: Great! You DON'T have Chronic Kidney Disease