Chronic kidney Disease Analysis

Chronic Kidney Disease (CKD) is a major medical problem and can be cured if treated it in the early stages. Usually, people are not aware that medical tests, we take for different purposes could contain valuable information concerning kidney diseases. Consequently, attributes of various medical tests are investigated to distinguish which attributes may contain helpful information about the disease.The information says that it helps us to measure the severity of the problem, the predicted survival of the patient after the illness, the pattern of the disease and work for curing the disease

Prerequisites:

1. To develop this project, we need to install following software/packages:

Anaconda Navigator:

Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning related applications. It can
be installed on Windows, Linux, and macOS. Conda is an open-source, cross-platform, package management system. Anaconda comes with so very nice tools like JupyterLab, Jupyter Notebook,
QtConsole, Spyder, Glueviz, Orange, Rstudio, Visual Studio Code. For this project, we will be
using Jupyter notebook and Visual studio code.

->To accomplish this, we must complete all the activities and tasks listed below

1. Data Collection

Download dataset/create dataset:

2.Data pre-processing

Importing required libraries:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix
```

Pandas: It is a python library mainly used for data manipulation.
NumPy: This python library is used for numerical analysis.
Matplotlib and Seaborn: Both are the data visualization library used for plotting graph which will help us for understanding the data.

Accuracy Score: used in classification type problem and for finding accuracy it is used.

Train_test_split:used for splitting data arrays into training data and for testing data.

Sklearn is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering, and dimensionality reduction via a consistence interface in Python.

Importing the dataset:

```
data=pd.read_csv(r"D:\project\chronickidney_disease.csv")
```

```
data.head(10)
```

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	sod	pot	hemo	htn	dm	cad	appet	pe	ane	classification
0	0	48.0	80.000000	1.020	1.0	0.0	1	1	0	0	...	137.528754	4.627244	15.4	1	4	1	0	0	0	0
1	1	7.0	50.000000	1.020	4.0	0.0	1	1	0	0	...	137.528754	4.627244	11.3	0	3	1	0	0	0	0
2	2	62.0	80.000000	1.010	2.0	3.0	1	1	0	0	...	137.528754	4.627244	9.6	0	4	1	1	0	1	0
3	3	48.0	70.000000	1.005	4.0	0.0	1	0	1	0	...	111.000000	2.500000	11.2	1	3	1	1	1	1	0
4	4	51.0	80.000000	1.010	2.0	0.0	1	1	0	0	...	137.528754	4.627244	11.6	0	3	1	0	0	0	0
5	5	60.0	90.000000	1.015	3.0	0.0	1	1	0	0	...	142.000000	3.200000	12.2	1	4	1	0	1	0	0
6	6	68.0	70.000000	1.010	0.0	0.0	1	1	0	0	...	104.000000	4.000000	12.4	0	3	1	0	0	0	0
7	7	24.0	76.469072	1.015	2.0	4.0	1	0	0	0	...	137.528754	4.627244	12.4	0	4	1	0	1	0	0
8	8	52.0	100.000000	1.015	3.0	0.0	1	0	1	0	...	137.528754	4.627244	10.8	1	4	1	0	0	1	0
9	9	53.0	90.000000	1.020	2.0	0.0	0	0	1	0	...	114.000000	3.700000	9.5	1	4	1	1	0	1	0

10 rows × 23 columns

- You might have your data in .csv files, .excel files or .tsv files or something else. But the goal is the same in all cases. If you want to analyse that data using pandas, the first step will be to read it into a data structure that's compatible with

pandas.

- Let's load a .csv data file into pandas. There is a function for it, called **read_csv().**We will need to locate the directory of the CSV file at first (it's more efficient to keep the dataset in the same directory as your program).

Data visualization:

- To check first five rows of dataset, we have a function call **head( ).**

```
data.head(5)
```

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	sod	pot	hemo	htn	dm	cad	appet	pe	ane	classification
0	0	48.0	80.0	1.020	1.0	0.0	1	1	0	0	...	137.528754	4.627244	15.4	1	4	1	0	0	0	0
1	1	7.0	50.0	1.020	4.0	0.0	1	1	0	0	...	137.528754	4.627244	11.3	0	3	1	0	0	0	0
2	2	62.0	80.0	1.010	2.0	3.0	1	1	0	0	...	137.528754	4.627244	9.6	0	4	1	1	0	1	0
3	3	48.0	70.0	1.005	4.0	0.0	1	0	1	0	...	111.000000	2.500000	11.2	1	3	1	1	1	1	0
4	4	51.0	80.0	1.010	2.0	0.0	1	1	0	0	...	137.528754	4.627244	11.6	0	3	1	0	0	0	0

5 rows × 23 columns

- To check last five rows of dataset, we have a function call **tail().**

```
data.tail(5)
```

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	sod	pot	hemo	htn	dm	cad	appet	pe	ane	classification
395	395	55.0	80.0	1.020	0.0	0.0	1	1	0	0	...	150.0	4.9	15.7	0	3	1	0	0	0	2
396	396	42.0	70.0	1.025	0.0	0.0	1	1	0	0	...	141.0	3.5	16.5	0	3	1	0	0	0	2
397	397	12.0	80.0	1.020	0.0	0.0	1	1	0	0	...	137.0	4.4	15.8	0	3	1	0	0	0	2
398	398	17.0	60.0	1.025	0.0	0.0	1	1	0	0	...	135.0	4.9	14.2	0	3	1	0	0	0	2
399	399	58.0	80.0	1.025	0.0	0.0	1	1	0	0	...	141.0	3.5	15.8	0	3	1	0	0	0	2

5 rows × 23 columns

- **data.columns** will return you all the column names which are present in your data.

```
data.columns
```

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

Taking care of Missing Data:

Sometimes you may find some data are missing in the dataset. We need to be equipped to handle the problem when we come across them. Obviously, you could remove the entire line of data but what if you are unknowingly removing crucial information? Of course we would not want to do that. One of the most common ideas to handle the problem is to take a mean of all the values for continuous and for categorical we make use of mode values and replace the missing data.

1. We will be using **isnull().any()** method to see which column has missing values.

```
data.isnull().any()
```

```
id                False
age               False
bp                False
sg                False
al                False
su                False
rbc               False
pc                False
pcc               False
ba                False
bgr               False
bu                False
sc                False
sod               False
pot               False
hemo              False
htn               False
dm                False
cad               False
appet             False
pe                False
ane               False
classification    False
dtype: bool
```

Label encoding

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 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. In this technique, each label is assigned a unique integer based on alphabetical ordering.

As we have to convert only the text class category columns, we first select it then we will implement Label Encoding to it.

```
data['rbc'].unique()
```

```
array(['normal', 'abnormal'], dtype=object)
```

```
data['htn'].unique()
```

```
array(['yes', 'no'], dtype=object)
```

```
data['appet'].unique()
```

```
array(['good', 'poor'], dtype=object)
```

```
data['ba'].unique()
```

```
array(['notpresent', 'present'], dtype=object)
```

```
data['rbc']=le.fit_transform(data['rbc'])
data['pc']=le.fit_transform(data['pc'])
data['pcc']=le.fit_transform(data['pcc'])
data['ba']=le.fit_transform(data['ba'])
data['htn']=le.fit_transform(data['htn'])
data['dm']=le.fit_transform(data['dm'])
data['cad']=le.fit_transform(data['cad'])
data['appet']=le.fit_transform(data['appet'])
data['pe']=le.fit_transform(data['pe'])
data['ane']=le.fit_transform(data['ane'])
data['classification']=le.fit_transform(data['classification'])
```

```
data.head(5)
```

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	sod	pot	hemo	htn	dm	cad	appet	pe	ane	classification
0	0	48.0	80.0	1.020	1.0	0.0	1	1	0	0	...	137.528754	4.627244	15.4	1	4	1	0	0	0	0
1	1	7.0	50.0	1.020	4.0	0.0	1	1	0	0	...	137.528754	4.627244	11.3	0	3	1	0	0	0	0
2	2	62.0	80.0	1.010	2.0	3.0	1	1	0	0	...	137.528754	4.627244	9.6	0	4	1	1	0	1	0
3	3	48.0	70.0	1.005	4.0	0.0	1	0	1	0	...	111.000000	2.500000	11.2	1	3	1	1	1	1	0
4	4	51.0	80.0	1.010	2.0	0.0	1	1	0	0	...	137.528754	4.627244	11.6	0	3	1	0	0	0	0

5 rows × 23 columns

Feature scaling:

spliting data into train and test

The train-test split is a technique for evaluating the performance of a machine learning

algorithm.

Train Dataset: Used to fit the machine learning model.

Test Dataset: Used to evaluate the fit machine learning model.

In general you can allocate 80% of the dataset to training set and the remaining 20% to test

set.

We will create 4 sets— X_train (training part of the matrix of features), X_test (test part of the matrix of features), Y_train (training part of the dependent variables associated with the X train sets, and therefore also the same indices), Y_test (test part of the dependent variables associated with the X test sets, and therefore also the same indices.

There are a few other parameters that we need to understand before we use the class:

1. **test_size** — this parameter decides the size of the data that has to be split as the test dataset. This is given as a fraction. For example, if you pass 0.5 as the value, the dataset will be split 50% as the test dataset
2. **train_size** — you have to specify this parameter only if you're not specifying the test_size. This is the same as test_size, but instead you tell the class what percent of the dataset you want to split as the training set.
3. **random_state** — here you pass an integer, which will act as the seed for the random number generator during the split. Or, you can also pass an instance of the Random_state class, which will become the number generator. If you don't

pass anything, the Random_state instance used by np.random will be used instead.

4. Now split our dataset into train set and test using train_test_split class from scikit learn library.

```
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=0)
```

```
x_train.shape
```

```
(320, 22)
```

```
y_train.shape
```

```
(320, 1)
```

Model Building

Training and testing the model:

There are several Machine learning algorithms to be used depending on the data you are going to process such as images, sound, text, and numerical values. The algorithms that you can choose according to the objective that you might have it may be Classification algorithms or Regression algorithms

Example:

1.Random Forest Classifier

2.KneighborsClassifier

Now we apply Random Forest Classifier to our dataset

Random Forest: It is a supervised learning algorithm.It is an ensemble of decision trees usually trained with bagging method.The bagging method is combination of learning models that increases the overall result.

 # Using Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state=0)
rfc.fit(x_train,y_train)
```

```
<ipython-input-66-b184fc2770cb>:3: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please cha
nge the shape of y to (n_samples,), for example using ravel().
  rfc.fit(x_train,y_train)
```

```
RandomForestClassifier(random_state=0)
```

```
y_pred_1 = rfc.predict(x_test)
```

```
y_pred_1
```

```
array([0, 2, 2, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 2, 0, 0, 2, 0, 0, 0, 0, 2,
       0, 2, 0, 2, 0, 0, 2, 0, 0, 2, 2, 0, 0, 0, 0, 0, 0, 2, 0, 2, 0, 0,
       2, 0, 2, 2, 0, 0, 0, 0, 2, 0, 2, 2, 0, 0, 0, 0, 0, 2, 0, 2, 0, 2,
       0, 0, 0, 2, 0, 2, 0, 0, 0, 0, 2, 2, 2, 2])
```

```
from sklearn.metrics import accuracy_score,confusion_matrix
cm=accuracy_score(y_test,y_pred_1)
```

```
cm
```

```
0.9875
```

Now we will apply SVM to our dataset

SVM:SVM's are a set of supervised learning methods used for classification, regression and outliers detection.The advantages of support vector machines are: Effective in high dimensional spaces.Still effective in cases where number of dimensions is greater than the number of samples.

Using SVM

```
from sklearn.svm import SVC
svm=SVC()
svm.fit(x_train,y_train)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:73: DataConversionWarning: A column-vector y was passed
when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
SVC()
```

```
ypred_svm = svm.predict(x_test)
```

```
ypred_svm
```

```
array([0, 2, 2, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 2, 0, 0, 2, 0, 0, 0, 0, 2,
       0, 2, 0, 2, 0, 0, 2, 0, 0, 2, 2, 0, 0, 0, 0, 0, 0, 2, 0, 2, 0, 0,
       2, 0, 2, 2, 0, 2, 2, 0, 2, 0, 2, 2, 0, 0, 0, 0, 0, 2, 0, 2, 0, 2,
       0, 0, 0, 2, 0, 2, 0, 0, 0, 0, 2, 2, 2, 2])
```

```
acc_svm=accuracy_score(y_test,ypred_svm)
```

```
acc_svm
```

```
0.9625
```

Application Building

Creating a HTML File, flask application.

1. Build python code
    a. Importing Libraries
    b. Routing to the html Page
    c. Showcasing prediction on UI
    d. Run The app in local browser

Task-1:Importing libraries

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 Pickle library to load the model file.

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

app=Flask(__name__)
rfc=pickle.load(open('kidney.pkl','rb'))
```

Task 2: Routing to the html Page

Here, declared constructor is used to route to the HTML page created earlier.

In the above example, '/' URL is bound with home.html function. Hence, when the home page of the web server is opened in browser, the html page is rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method. Here, "kidney.html" is rendered when home button is clicked on the UI.

```
@app.route('/predict',methods=['post'])
def predict():
    Id=int(request.form['id'])
    Age=int(request.form['age'])
    Bp=int(request.form['bp'])
    Sg=int(request.form['sg'])
    Al=int(request.form['al'])
    Su=int(request.form['su'])
    Rbc=int(request.form['rbc'])
    Pc=int(request.form['pc'])
    Pcc=int(request.form['pcc'])
    Ba=int(request.form['ba'])
    Bgr=int(request.form['bgr'])
    Bu=int(request.form['bu'])
    Sc=int(request.form['sc'])
    Sod=int(request.form['sod'])
    Pot=int(request.form['pot'])
    Hemo=int(request.form['hemo'])
    Htn=int(request.form['htn'])
    Dm=int(request.form['dm'])
    Cad=int(request.form['cad'])
    Appet=int(request.form['appet'])
    Pe=int(request.form['pe'])
    Ane=int(request.form['ane'])


    a=np.array([[Id,Age,Bp,Sg,Al,Su,Rbc,Pc,Pcc,Ba,Bgr,Bu,Sc,Sod,Pot,Hemo,Htn,Dm,C
    print(a)

    pred=rfc.predict(a)
```

```
if(pred == 0):
    output="Chronic Kidney Disease"
    print("Chronic Kidney Disease")
else:
    output="Not Chronic Kidney Disease"
    print("Not Chronic Kidney Disease")


return render_template('kidney.html', prediction_text= output)
```

Task3: Main Function

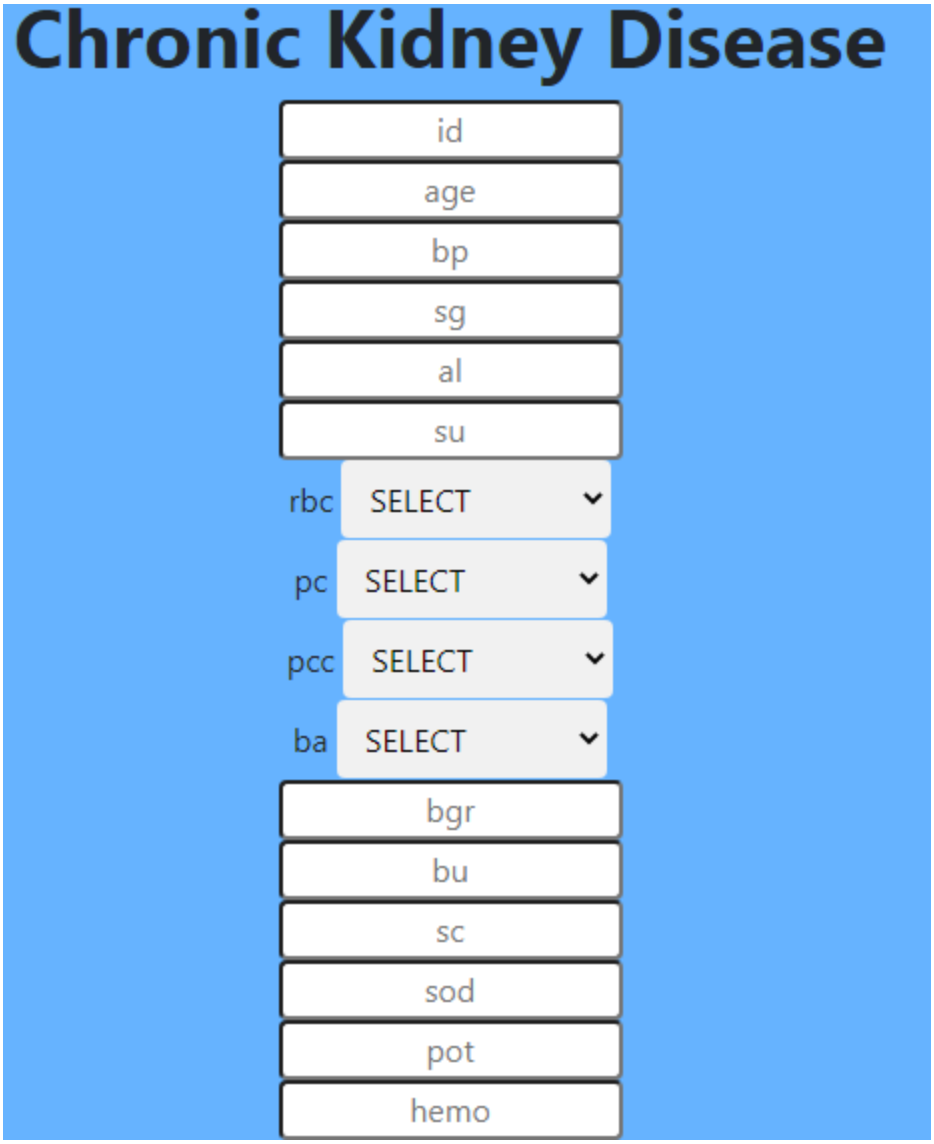
This is used to run the application in a local host

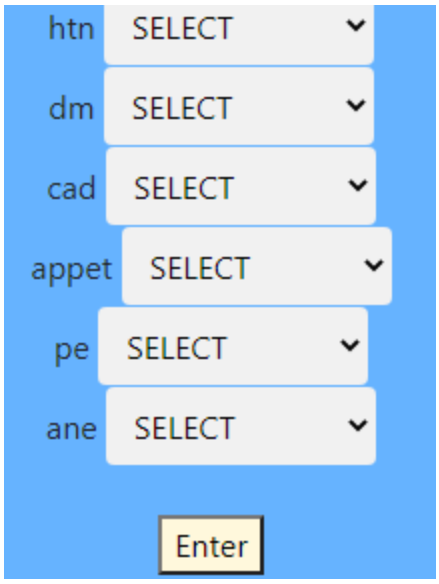
```
if __name__ == '__main__':
    app.run()
```

Activity 3:Run the application

```
In [4]: runfile('C:/Users/Manikanta/Desktop/ml/kidney.py', wdir='C:/Users/Manikanta/
Desktop/ml')
 * Serving Flask app "kidney" (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)
```

Output Screen:




htn
SELECT
dm
SELECT
cad
SELECT
appet
SELECT
pe
SELECT
ane
SELECT
Enter