



Early Prediction for Chronic Kidney Disease Detection

Project Hand-out, Faculty Development Program –

College: Anand International College of Engineering, Jaipur

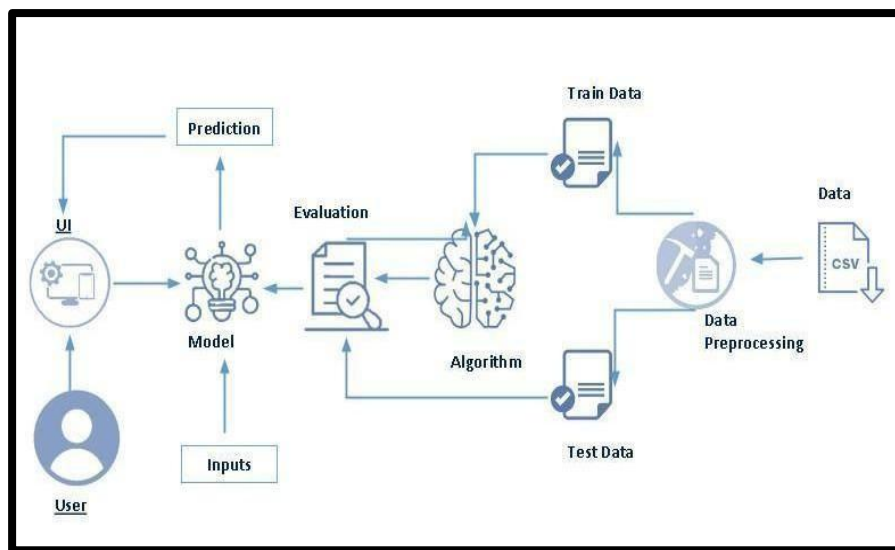
Track: Artificial Intelligence and Machine Learning
Team Details:

Name	College id
1. Priyanshu Kumar(leader)	23CS071
2. Ritwik Kumar	23CS083
3. Shailja Agrawal	23CS087
4. Pari Swami	23CS068

Early Prediction for Chronic Kidney Disease Detection: Progressive Approach to Health Management

Chronic Kidney Disease (CKD) is a major medical problem and can be cured if treated 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.

Technical Architecture:



Project Flow

- User interacts with the UI to enter the input.
- Entered input is analysed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI

To accomplish this, we have to complete all the activities listed below,

- Define Problem / Problem Understanding
 - Specify the business problem
 - Business requirements
 - Literature Survey
 - Social or Business Impact.
- Data Collection & Preparation
 - Collect the dataset
 - Data Preparation
- Exploratory Data Analysis
 - Descriptive statistical
 - Visual Analysis
- Model Building
 - Training the model in multiple algorithms
 - Testing the model
- Performance Testing & Evaluate the results
 - Testing model with multiple evaluation metrics
 - Evaluate the results
- Model Deployment
 - Save the best model
 - Integrate with Web Framework
- Project Demonstration & Documentation
 - Record explanation Video for project end to end solution
 - Project Documentation-Step by step project development procedure

Prior Knowledge:

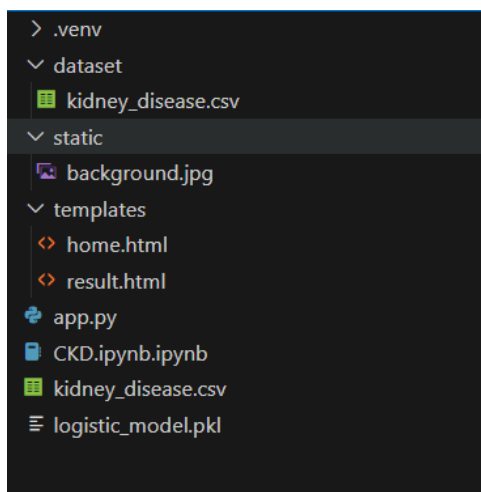
You must have prior knowledge of the following topics to complete this

project:

- ML Concepts
 - Supervised learning
 - Unsupervised learning
- Algorithms
 - Decision Tree
 - Random Forest
 - K-Nearest Neighbors (KNN)
 - XGBoost
- Evaluation Metrics
 - Accuracy, Precision, Recall, F1-Score, ROC-AUC, etc.
- Flask Basics
 - Understanding routing, rendering templates, and handling forms

Project Structure:

Create the Project folder which contains files as shown below



- We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.
 - Logistic_regression_model.pkl that is our best and saved model. Further we will use this model for flask integration.
 - Data Folder contains the Dataset used
- The Notebook file contains procedure for building the model.

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the Business Problem

Chronic Kidney Disease (CKD) is a progressive condition that leads to a gradual loss of kidney function over time. If not detected early, it can result in severe complications, including end-stage renal disease (ESRD), requiring dialysis or a kidney transplant. The goal of this project is to build a machine learning model that can assist in the early detection of CKD using patient medical data. This predictive tool aims to support healthcare professionals in identifying at-risk individuals early, enabling timely intervention and improved patient care.

Activity 2: Business Requirements

The business requirements for developing a machine learning model to predict chronic kidney disease are as follows:

- The model should accurately classify whether a patient is likely to have CKD based on input medical attributes.
- The system must minimize false positives (predicting CKD when the patient doesn't have it) and false negatives (failing to predict CKD in a patient who has it).
- The model should be interpretable, offering insight into the factors that led to the prediction, ensuring transparency and compliance with healthcare standards.
- The solution must be lightweight and suitable for integration with a web-based application (e.g., using Flask) for real-world deployment by medical professionals.

Activity 3: Literature Survey

Chronic Kidney Disease (CKD) poses a serious health risk worldwide, affecting approximately 14% of the global population. The condition progresses slowly and is often asymptomatic in its early stages, making early detection vital. If left untreated, CKD can advance to ESRD, requiring life-sustaining treatments such as dialysis or transplantation.

Recent research in medical data science has shown the effectiveness of machine learning in predicting CKD. These models leverage structured clinical data—such as blood pressure, blood glucose, and serum creatinine levels—to detect patterns indicative of kidney dysfunction. Literature also highlights the importance of model interpretability in clinical applications, making explainable AI approaches critical in healthcare deployments.

Activity 4: Social or Business Impact

Social Impact:

Early detection of CKD through predictive modeling can significantly improve patient outcomes by allowing timely intervention. This can lead to better disease management through lifestyle changes, medication, and regular monitoring—ultimately preventing or delaying the onset of ESRD. Early intervention enhances the quality of life and reduces the emotional and financial stress on patients and their

families.

Business Impact:

From a healthcare management perspective, early prediction can reduce the need for expensive treatments like dialysis and kidney transplantation. It also decreases the frequency of hospital admissions and emergency visits, thereby lowering the overall burden on healthcare systems. Implementing predictive models in clinical settings can lead to cost-effective, data-driven decision-making, enhancing operational efficiency and patient satisfaction.

Milestone 2: Data Collection & Preparation

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

Activity 1: Collect the dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link: <https://www.kaggle.com/datasets/mansoordaku/ckdisease>

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Activity 1.1: Importing the libraries

Import the necessary libraries as shown in the image. (optional) Here we have used visualization style as FiveThirtyEight.

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5
6 from sklearn.model_selection import train_test_split
7 from sklearn.preprocessing import LabelEncoder
8 !pip install scikeras
9 import pickle
10 from sklearn.linear_model import LogisticRegression
```

Activity 1.2: 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 we have to give the directory of the csv file.

```
1 data=pd.read_csv("/content/kidney_disease.csv")
2 data.head()
```

	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 x 26 columns

Data has 25 features which may predict a patient with chronic kidney ..

Data has 25 features which may predict a patient with chronic kidney disease..

<https://www.kaggle.com/datasets/mansoordaku/ckdisease>

Activity 2: 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.

Activity 2.1: Rename the columns

```
1 data.columns

Index(['id', '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')

1 data.columns=['id','age', 'blood_pressure', 'specific_gravity', 'albumin',
2               'sugar','red_blood_cells', 'pus_cell', 'pus_cell_clumps', 'bacteria',
3               'blood glucose random', 'blood_urea', 'serum_creatinine', 'sodium','potassium',
4               ' hemoglobin', 'packed_cell_volume', 'white_blood_cell_count', 'red_blood_cell_count',
5               'hypertension', 'diabetesmellitus', 'coronary_artery_disease','appetite',
6               'pedal_edema', 'anemia', 'class']
7 data.columns

Index(['id', 'age', 'blood_pressure', 'specific_gravity', 'albumin', 'sugar',
      'red_blood_cells', 'pus_cell', 'pus_cell_clumps', 'bacteria',
      'blood glucose random', 'blood_urea', 'serum_creatinine', 'sodium',
      'potassium', ' hemoglobin', 'packed_cell_volume',
      'white_blood_cell_count', 'red_blood_cell_count', 'hypertension',
      'diabetesmellitus', 'coronary_artery_disease', 'appetite',
      'pedal_edema', 'anemia', 'class'],
      dtype='object')
```


Activity 2.2: 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()` [we used data as df which is data frame] function is used.

```
1 data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 26 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   id                                     400 non-null    int64
1   age                                   391 non-null    float64
2   blood_pressure                       388 non-null    float64
3   specific_gravity                    353 non-null    float64
4   albumin                             354 non-null    float64
5   sugar                               351 non-null    float64
6   red_blood_cells                     248 non-null    object
7   pus_cell                            335 non-null    object
8   pus_cell_clumps                     396 non-null    object
9   bacteria                            396 non-null    object
10  blood_glucose_random                 356 non-null    float64
11  blood_urea                           381 non-null    float64
12  serum_creatinine                     383 non-null    float64
13  sodium                              313 non-null    float64
14  potassium                            312 non-null    float64
15  hemoglobin                           348 non-null    float64
16  packed_cell_volume                   330 non-null    object
17  white_blood_cell_count               295 non-null    object
18  red_blood_cell_count                 270 non-null    object
19  hypertension                         398 non-null    object
20  diabetesmellitus                     398 non-null    object
21  coronary_artery_disease              398 non-null    object
22  appetite                             399 non-null    object
23  pedal_edema                          399 non-null    object
24  anemia                               399 non-null    object
25  class                                400 non-null    object
dtypes: float64(11), int64(1), object(14)
memory usage: 81.4+ KB
```

```
] 1 data.isnull().any()
```

	0
id	False
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

```

1 data['packed_cell_volume'] = pd.to_numeric(data['packed_cell_volume'], errors='coerce')
2 data['white_blood_cell_count'] = pd.to_numeric(data['white_blood_cell_count'], errors='coerce')
3 data['red_blood_cell_count'] = pd.to_numeric(data['red_blood_cell_count'], errors='coerce')
4
5 data['blood_glucose_random'] = data['blood glucose random'].fillna(data['blood glucose random'].mean())
6 data['blood_pressure'] = data['blood pressure'].fillna(data['blood pressure'].mean())
7 data['blood_urea'] = data['blood urea'].fillna(data['blood urea'].mean())
8 data['hemoglobin'] = data['hemoglobin'].fillna(data['hemoglobin'].mean())
9 data['packed_cell_volume'] = data['packed_cell_volume'].fillna(data['packed_cell_volume'].mean())
10 data['potassium'] = data['potassium'].fillna(data['potassium'].mean())
11 data['red_blood_cell_count'] = data['red_blood_cell_count'].fillna(data['red_blood_cell_count'].mean())
12 data['serum_creatinine'] = data['serum_creatinine'].fillna(data['serum_creatinine'].mean())
13 data['sodium'] = data['sodium'].fillna(data['sodium'].mean())
14 data['white_blood_cell_count'] = data['white blood cell count'].fillna(data['white_blood_cell_count'].mean())
15 data['age'] = data['age'].fillna(data['age'].mode()[0])
16 data['hypertension'] = data['hypertension'].fillna(data['hypertension'].mode()[0])
17 data['pus_cell_clumps'] = data['pus_cell clumps'].fillna(data['pus_cell clumps'].mode()[0])
18 data['appetite'] = data['appetite'].fillna(data['appetite'].mode()[0])
19 data['albumin'] = data['albumin'].fillna(data['albumin'].mode()[0])
20 data['pus_cell'] = data['pus cell'].fillna(data['pus cell'].mode()[0])
21 data['red_blood_cells'] = data['red blood cells'].fillna(data['red blood cells'].mode()[0])
22 data['coronary_artery_disease'] = data['coronary artery disease'].fillna(data['coronary_artery_disease'].mode()[0])
23 data['bacteria'] = data['bacteria'].fillna(data['bacteria'].mode()[0])
24 data['anemia'] = data['anemia'].fillna(data['anemia'].mode()[0])
25 data['sugar'] = data['sugar'].fillna(data['sugar'].mode()[0])
26 data['diabetesmellitus'] = data['diabetesmellitus'].fillna(data['diabetesmellitus'].mode()[0])
27 data['pedal_edema'] = data['pedal edema'].fillna(data['pedal edema'].mode()[0])
28 data['specific_gravity'] = data['specific gravity'].fillna(data['specific gravity'].mode()[0])

```

Let's now check the count of null values after filling all null values using `isnull.sum()`

Activity 2.3: 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**.

```

1 catcols = set(data.dtypes[data.dtypes == 'O'].index.values)
2 print(catcols)

{'coronary_artery_disease', 'pus_cell', 'red_blood_cells', 'appetite', 'anemia', 'hypertension', 'class', 'bacteria', 'diabetesmellitus', 'pus_co

```

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

```

1 from collections import Counter as c
2
3 for i in catcols:
4     print(f"Column: {i}")
5     print(c(data[i]))
6     print("=="*120 + "\n")

Column: coronary_artery_disease
Counter({'no': 364, 'yes': 34, '\tno': 2})
=====

Column: pus_cell
Counter({'normal': 324, 'abnormal': 76})
=====

Column: red_blood_cells
Counter({'normal': 353, 'abnormal': 47})
=====

Column: appetite
Counter({'good': 318, 'poor': 82})
=====

Column: anemia
Counter({'no': 340, 'yes': 60})
=====

Column: hypertension
Counter({'no': 253, 'yes': 147})
=====

Column: class

```

```

Column: class
Counter({'ckd': 248, 'notckd': 150, 'ckd\t': 2})
=====

Column: bacteria
Counter({'notpresent': 378, 'present': 22})
=====

Column: diabetesmellitus
Counter({'no': 260, 'yes': 134, '\tno': 3, '\tyes': 2, ' yes': 1})
=====

Column: pus_cell_clumps
Counter({'notpresent': 358, 'present': 42})
=====

Column: pedal_edema
Counter({'no': 324, 'yes': 76})
=====

```

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.

```

1 catcols = set(data.dtypes[data.dtypes == 'O'].index.values)
2 print(catcols)

{'coronary_artery_disease', 'pus_cell', 'red_blood_cells', 'appetite', 'anemia', 'hypertension', 'class', 'bacteria', 'diabetesmellitus', 'pus_cell_clumps', 'pedal_edema'}

1 from collections import Counter as c
2
3 for i in catcols:
4     print(f"Column: {i}")
5     print(c(data[i]))
6     print("*****")

Column: coronary_artery_disease
Counter({'no': 364, 'yes': 34, '\tno': 2})
*****

Column: pus_cell
Counter({'normal': 324, 'abnormal': 76})
*****

Column: red_blood_cells
Counter({'normal': 353, 'abnormal': 47})
*****

Column: appetite
Counter({'good': 318, 'poor': 82})
*****

Column: anemia
Counter({'no': 340, 'yes': 60})
*****

Column: hypertension
Counter({'no': 253, 'yes': 147})
*****

Column: class
Counter({'ckd': 248, 'notckd': 150, 'ckd\t': 2})
*****

Column: bacteria
Counter({'notpresent': 378, 'present': 22})
*****

Column: diabetesmellitus
Counter({'no': 260, 'yes': 134, '\tno': 3, '\tyes': 2, ' yes': 1})
*****

Column: pus_cell_clumps
Counter({'notpresent': 358, 'present': 42})
*****

```

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.

Activity 2.3.1: 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. 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.

```
1 catcols=['bacteria', 'anemia', 'hypertension', 'coronary_artery_disease', 'class', 'appetite', 'pus_cell_clumps', 'pus_cell', 'diabetesmellitus', 'red_blood_cells', 'pedal_edema']

1 from sklearn.preprocessing import LabelEncoder #importing the LabelEncoding from sklearn
2 for i in catcols: #looping through all the categorical columns
3     print("LABEL ENCODING OF:",i)
4     LEi = LabelEncoder() # creating an object of LabelEncoder
5     print(c(data[i])) #getting the classes values before transformation
6     data[i]= LEi.fit_transform(data[i])# transforming our text classes to numerical values
7     print(c(data[i])) #getting the classes values after transformation
8     print("****100")

LABEL ENCODING OF: bacteria
Counter({'notpresent': 378, 'present': 22})
Counter({0: 378, 1: 22})
*****
LABEL ENCODING OF: anemia
Counter({'no': 340, 'yes': 60})
Counter({0: 340, 1: 60})
*****
LABEL ENCODING OF: hypertension
Counter({'no': 253, 'yes': 147})
Counter({0: 253, 1: 147})
*****
LABEL ENCODING OF: coronary_artery_disease
Counter({'no': 364, 'yes': 34, 'tno': 2})
Counter({1: 364, 2: 34, 0: 2})
*****
LABEL ENCODING OF: class
Counter({'ckd': 248, 'notckd': 150, 'ckd\t': 2})
Counter({0: 248, 2: 150, 1: 2})
*****
LABEL ENCODING OF: appetite
Counter({'good': 318, 'poor': 82})
Counter({0: 318, 1: 82})
*****
LABEL ENCODING OF: pus_cell_clumps
Counter({'notpresent': 358, 'present': 42})
Counter({0: 358, 1: 42})
*****
LABEL ENCODING OF: pus_cell
Counter({'normal': 324, 'abnormal': 76})
Counter({1: 324, 0: 76})
*****
LABEL ENCODING OF: diabetesmellitus
Counter({'no': 260, 'yes': 134, 'tno': 3, 'tyes': 2, ' yes': 1})
Counter({3: 260, 4: 134, 0: 3, 1: 2, 2: 1})
*****
LABEL ENCODING OF: red_blood_cells
Counter({'normal': 353, 'abnormal': 47})
Counter({1: 353, 0: 47})
*****
LABEL ENCODING OF: pedal_edema
Counter({'no': 324, 'yes': 76})
```

In the above code we are looping through all the selected text class categorical columns and performing label encoding.

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

Activity 2.4: Handling Numerical columns

```
1 catcols = set(data.dtypes[data.dtypes == 'O'].index.values)
2 print(catcols)

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

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

```
1 from collections import Counter as c
2
3 for i in catcols:
4     print(f"Column: {i}")
5     print(c(data[i]))
6     print(" "*120 + "\n")

Column: coronary_artery_disease
Counter({'no': 364, 'yes': 34, '\tno': 2})
*****
Column: pus_cell
```

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.

```
1 keys_to_remove = ['specific_gravity', 'albumin', 'sugar']
2 for key in keys_to_remove:
3     if key in contcols:
4         contcols.remove(key)
5 print(contcols)

'class', 'hemoglobin', 'blood_pressure', 'pus_cell_clumps', 'diabetesmellitus', 'pus_cell', 'appetite', 'blood_glucose_random', 'pedal_edema', 'coronary_artery_disease', 'red_blood_cells'
```

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

```
1 contcols.add('red_blood_cell_count')
2 contcols.add('white_blood_cell_count')
3 contcols.add('packed_cell_volume')
4 print(contcols)

'class', ' hemoglobin', 'blood_pressure', 'pus_cell_c

1 catcols = set(catcols)
2 catcols.add('specific_gravity')
3 catcols.add('albumin')
4 catcols.add('sugar')
5 print(catcols)

'coronary_artery_disease', 'pus_cell', 'albumin', 'red
```

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

```
1 data['coronary_artery_disease'] = data['coronary_artery_disease'].replace({1: 'no', 2: 'yes', 0: np.nan})
2 print(c(data['coronary_artery_disease']))

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

1 data['diabetesmellitus'] = data['diabetesmellitus'].replace({3: 'no', 4: 'yes', 0: np.nan, 1: 'no', 2: 'yes'})
2 print(c(data['diabetesmellitus']))

Counter({'no': 262, 'yes': 135, nan: 3})
```


Milestone 3: Exploratory Data Analysis

Activity 1: 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.

```
1 data.describe()
```

	id	age	blood_pressure	specific_gravity	albumin	sugar	red_blood_cells	pus_cell	pus_cell_clumps	bacteria	...	potassium	hemoglobin
count	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	...	400.000000	400.000000
mean	199.500000	51.675000	76.469072	1.017712	0.900000	0.395000	0.882500	0.810000	0.105000	0.055000	...	4.627244	12.526437
std	115.614301	17.022008	13.476298	0.005434	1.313131	1.040038	0.322418	0.392792	0.306937	0.228266	...	2.819783	2.716171
min	0.000000	2.000000	50.000000	1.005000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	2.500000	3.100000
25%	99.750000	42.000000	70.000000	1.015000	0.000000	0.000000	1.000000	1.000000	0.000000	0.000000	...	4.000000	10.875000
50%	199.500000	55.000000	78.234536	1.020000	0.000000	0.000000	1.000000	1.000000	0.000000	0.000000	...	4.627244	12.526437
75%	299.250000	64.000000	80.000000	1.020000	2.000000	0.000000	1.000000	1.000000	0.000000	0.000000	...	4.800000	14.625000
max	399.000000	90.000000	180.000000	1.025000	5.000000	5.000000	1.000000	1.000000	1.000000	1.000000	...	47.000000	17.800000

8 rows x 24 columns

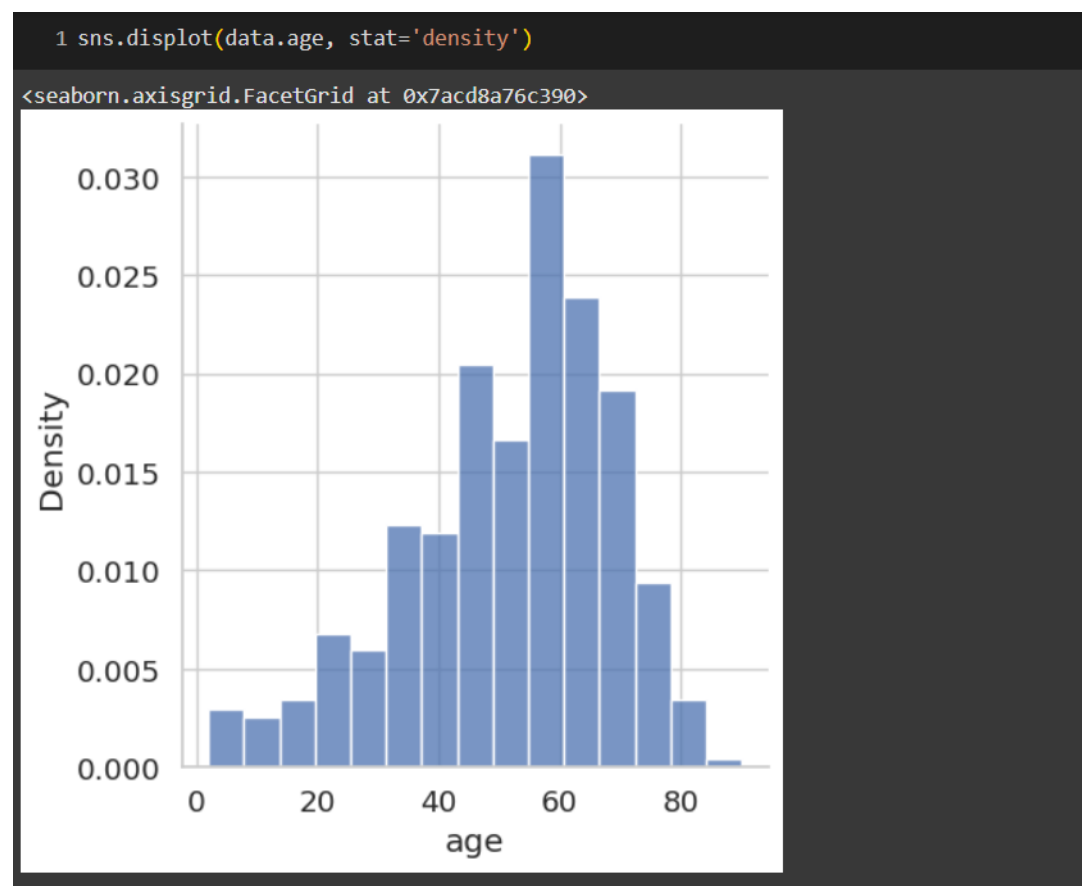
Activity 2: Visual analysis

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.

Activity 2.1: 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.

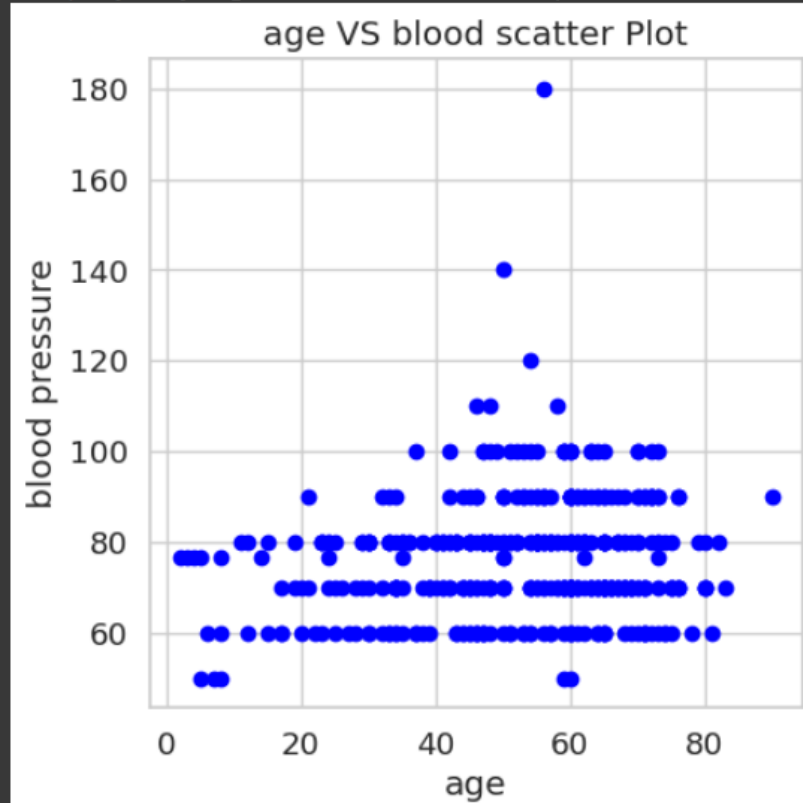
The Seaborn package provides a wonderful function distplot. With the help of distplot, we can find the distribution of the feature.



Activity 2.2: Bivariate analysis

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

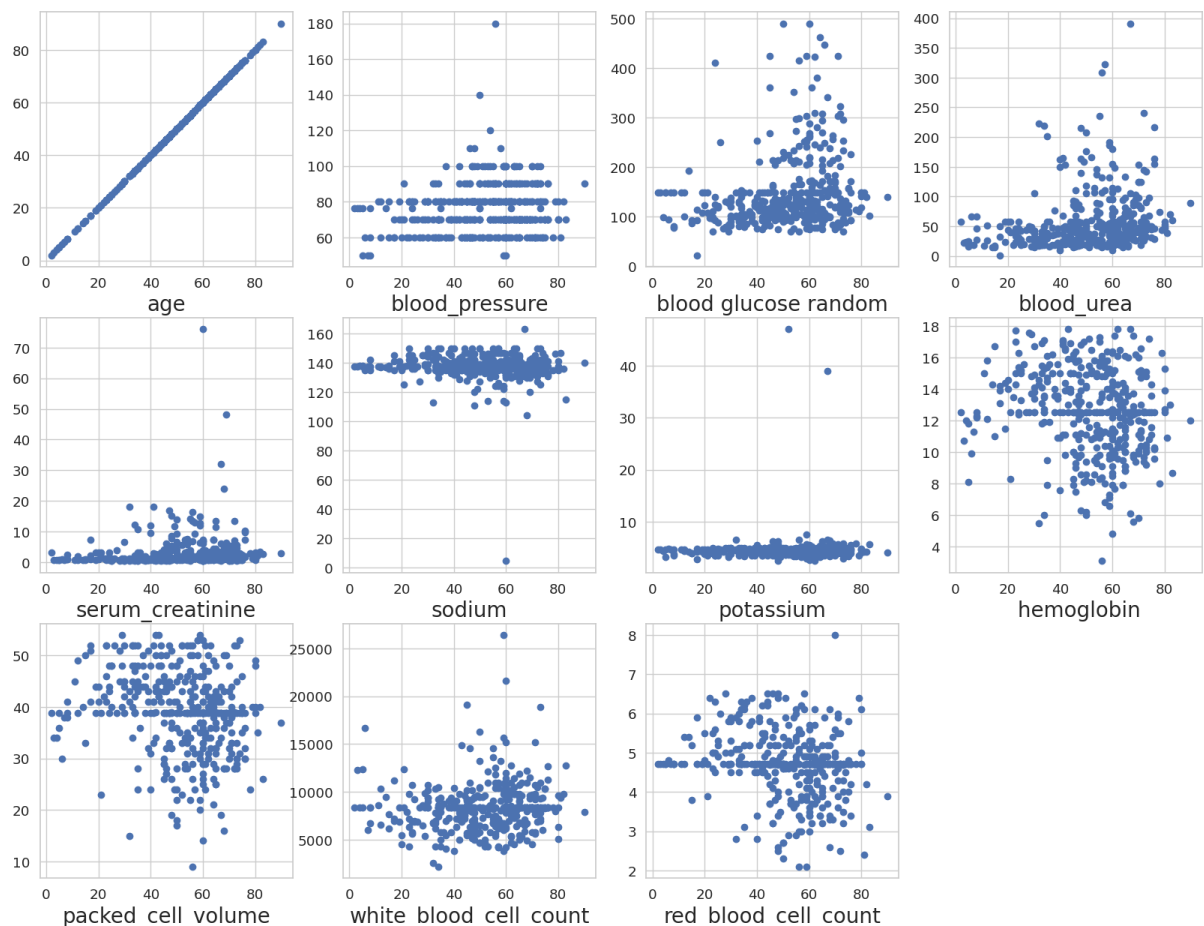
Text(0.5, 1.0, 'age VS blood scatter Plot')



Activity 2.3: Multivariate analysis

Age vs all continuous columns

```
1 # Re-define contcols to only include the actual continuous columns
2 contcols = ['age', 'blood_pressure', 'blood_glucose_random', 'blood_urea', 'serum_creatinine', 'sodium',
3             'potassium', 'hemoglobin', 'packed_cell_volume', 'white_blood_cell_count', 'red_blood_cell_count']
4
5 plt.figure(figsize=(20,15), facecolor='white')
6 plotnumber = 1
7 for column in contcols:
8     if plotnumber <= 11 :
9         ax = plt.subplot(3,4, plotnumber)
10        plt.scatter(data['age'], data[column])
11        plt.xlabel(column, fontsize=20)
12        plotnumber += 1
13 plt.show()
```



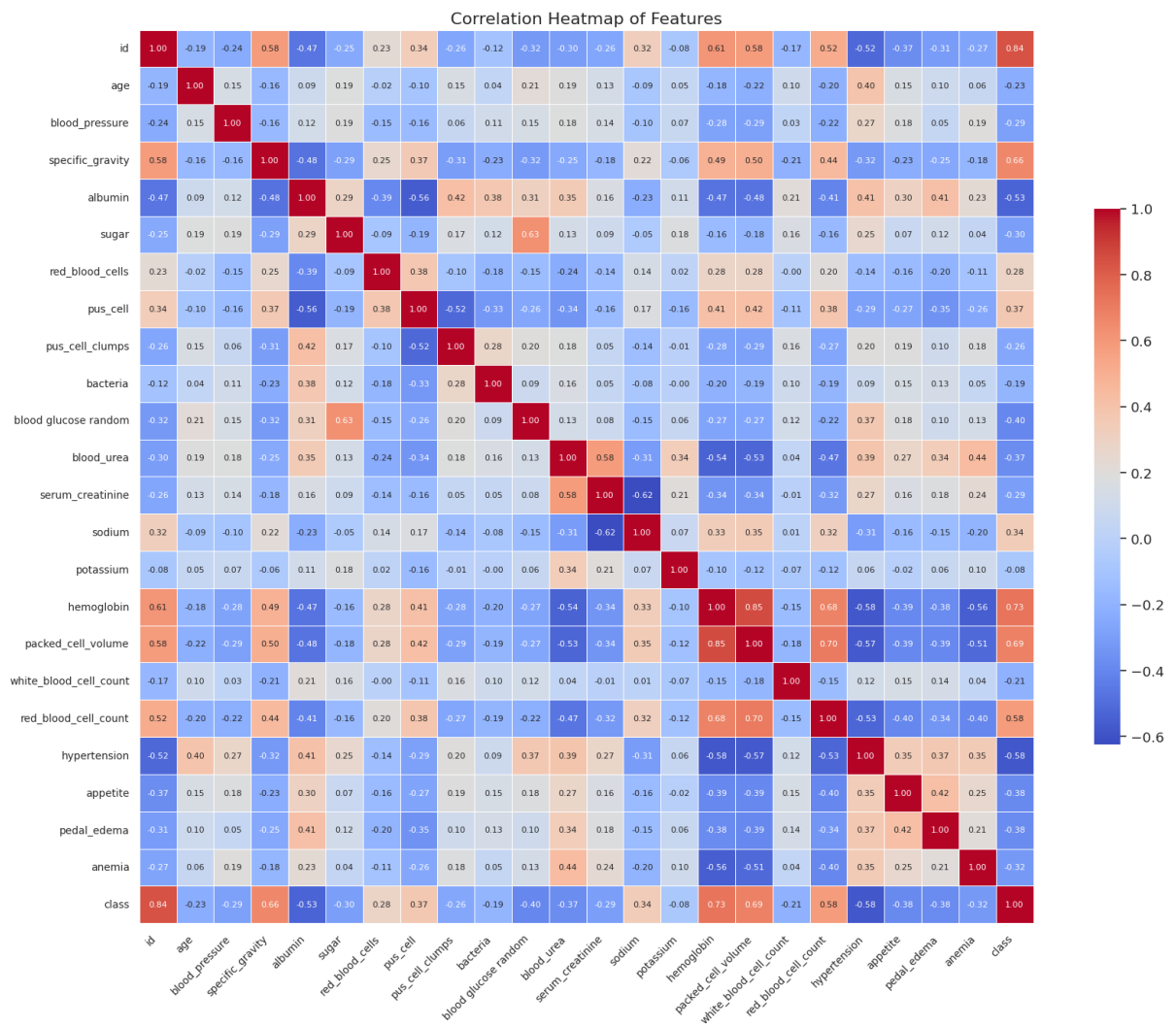
As you can observe with the scatter plot many of features are correlated with age.

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.

```
1 #correlation between independent columns
2 #heatmap
3 plt.figure(figsize=(18, 14))
4 corr_matrix = data.corr(numeric_only=True)
5
6 sns.heatmap(
7     corr_matrix,
8     annot=True,
9     fmt=".2f",
10    cmap="coolwarm",
11    linewidths=0.5,
12    linecolor="white",
13    square=True,
14    cbar_kws={"shrink": 0.6},
15    annot_kws={"size": 8} # smaller annotation text
16 )
17
18 plt.xticks(rotation=45, ha='right', fontsize=10)
19 plt.yticks(rotation=0, fontsize=10)
20 plt.title("Correlation Heatmap of Features", fontsize=16)
21 plt.tight_layout()
22 plt.show()
```

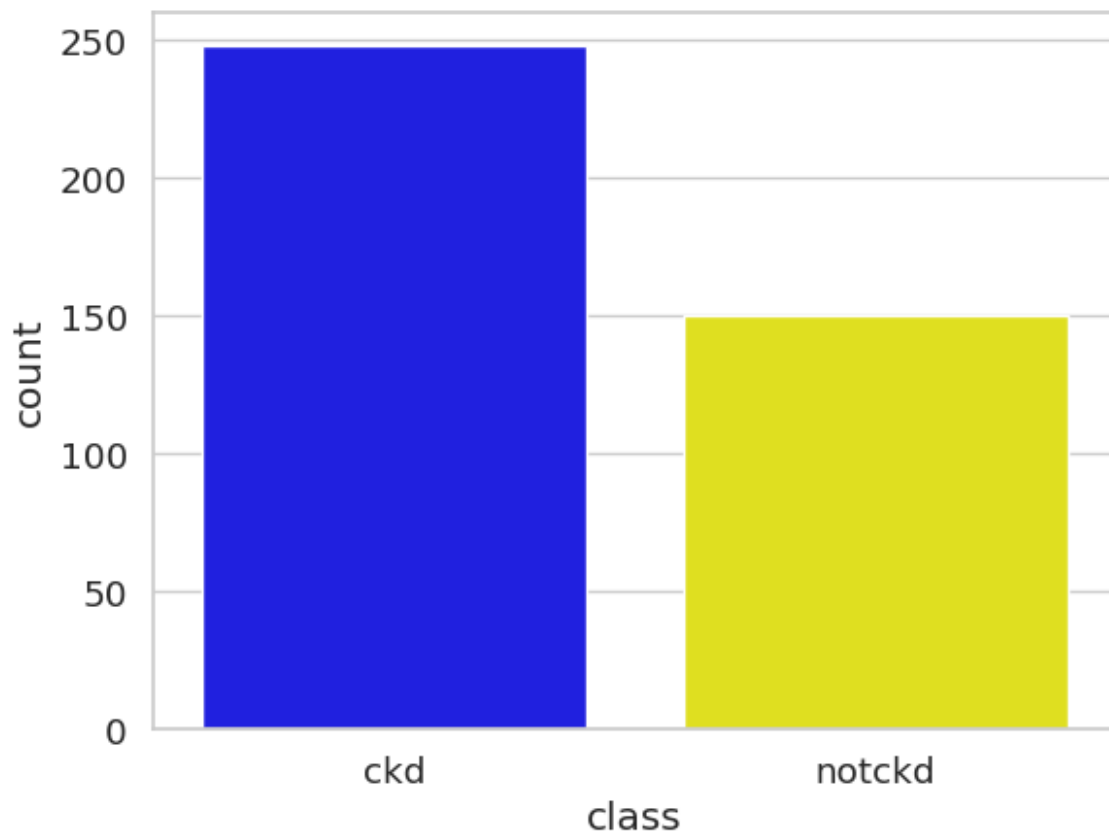


If you observe the heatmap, lighter the color the correlation between that two variables will be high.

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

```
1 plot_data = data.copy()
2 plot_data['class'] = plot_data['class'].replace({0: 'ckd', 2: 'notckd'})
3
4 # Filter out rows where 'class' is 1
5 plot_data = plot_data[plot_data['class'] != 1]
6
7 sns.countplot(x='class', data=plot_data, palette={'ckd': 'blue', 'notckd': 'yellow'})
```



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.

```
# performing feature scaling operation using standard scaler 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.

```
1 selcols=['red_blood_cells', 'pus_cell','blood glucose random', 'blood_urea',  
2         'pedal_edema','anemia', 'diabetesmellitus', 'coronary_artery_disease']  
3 x=pd.DataFrame(data,columns=selcols)  
4 y=pd.DataFrame(data,columns=['class'])  
5 print(x.shape)  
6 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.

```
1 from sklearn.model_selection import train_test_split  
2 x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2, random_state=42)#train test split
```

Milestone 4: Model Building

Activity 1: 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.

Activity 1.1: 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

```
1 from tensorflow.keras.models import Sequential
2 from tensorflow.keras.layers import Dense
3
4 ann_model = Sequential([
5     Dense(16, activation='relu', input_shape=(x_train.shape[1],)),
6     Dense(8, activation='relu'),
7     Dense(1, activation='sigmoid')
8 ])
9
10 ann_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
11 ann_model.fit(x_train, y_train, epochs=50, batch_size=16, verbose=0)
12 ann_loss, ann_acc = ann_model.evaluate(x_test, y_test, verbose=0)
13 print("ANN Accuracy:", ann_acc)
```

/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass super().__init__(activity_regularizer=activity_regularizer, **kwargs)
ANN Accuracy: 0.36250001192092896

Activity 1.2: 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.

```
1 from sklearn.ensemble import RandomForestClassifier
2 from sklearn.metrics import accuracy_score # Import accuracy_score
3
4 rf_model = RandomForestClassifier(n_estimators=100)
5 rf_model.fit(x_train, y_train)
6 rf_pred = rf_model.predict(x_test)
7 rf_acc = accuracy_score(y_test, rf_pred)
8 print("Random Forest Accuracy:", rf_acc)

/usr/local/lib/python3.11/dist-packages/sklearn/base.py:1389: DataConversionWarning:
  return fit_method(estimator, *args, **kwargs)
Random Forest Accuracy: 0.975
```

Activity 1.3: 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 confusion matrix and classification report is done.

```
1 from sklearn.tree import DecisionTreeClassifier
2 from sklearn.metrics import accuracy_score
3
4 dt_model = DecisionTreeClassifier()
5 dt_model.fit(x_train, y_train)
6 dt_pred = dt_model.predict(x_test)
7 dt_acc = accuracy_score(y_test, dt_pred)
8 print("Decision Tree Accuracy:", dt_acc)

Decision Tree Accuracy: 0.925
```


Activity 1.4: Logistic Regression

```
1 from sklearn.linear_model import LogisticRegression
2 from sklearn.metrics import accuracy_score
3 from sklearn.preprocessing import LabelEncoder # Import LabelEncoder
4
5 # Instantiate LabelEncoder
6 le = LabelEncoder()
7
8 # Apply Label Encoding to the relevant columns in x_train and x_test
9 for col in ['diabetesmellitus', 'coronary_artery_disease']:
10     # Ensure that the column exists before attempting to encode
11     if col in x_train.columns:
12         # Handle potential NaN values by converting to string before encoding
13         x_train[col] = le.fit_transform(x_train[col].astype(str))
14         x_test[col] = le.transform(x_test[col].astype(str))
15
16
17 log_model = LogisticRegression(max_iter=1000)
18 log_model.fit(x_train, y_train)
19 log_pred = log_model.predict(x_test)
20 log_acc = accuracy_score(y_test, log_pred)
21 print("Logistic Regression Accuracy:", log_acc)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataConversionWarning:
  y = column_or_1d(y, warn=True)
Logistic Regression Accuracy: 0.975
```

Activity 2: Testing the model

```
# 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])
```

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.

[illegible]

```
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)
```

98]

```
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.')
```

00]

.. Prediction: Low chance of CKD.

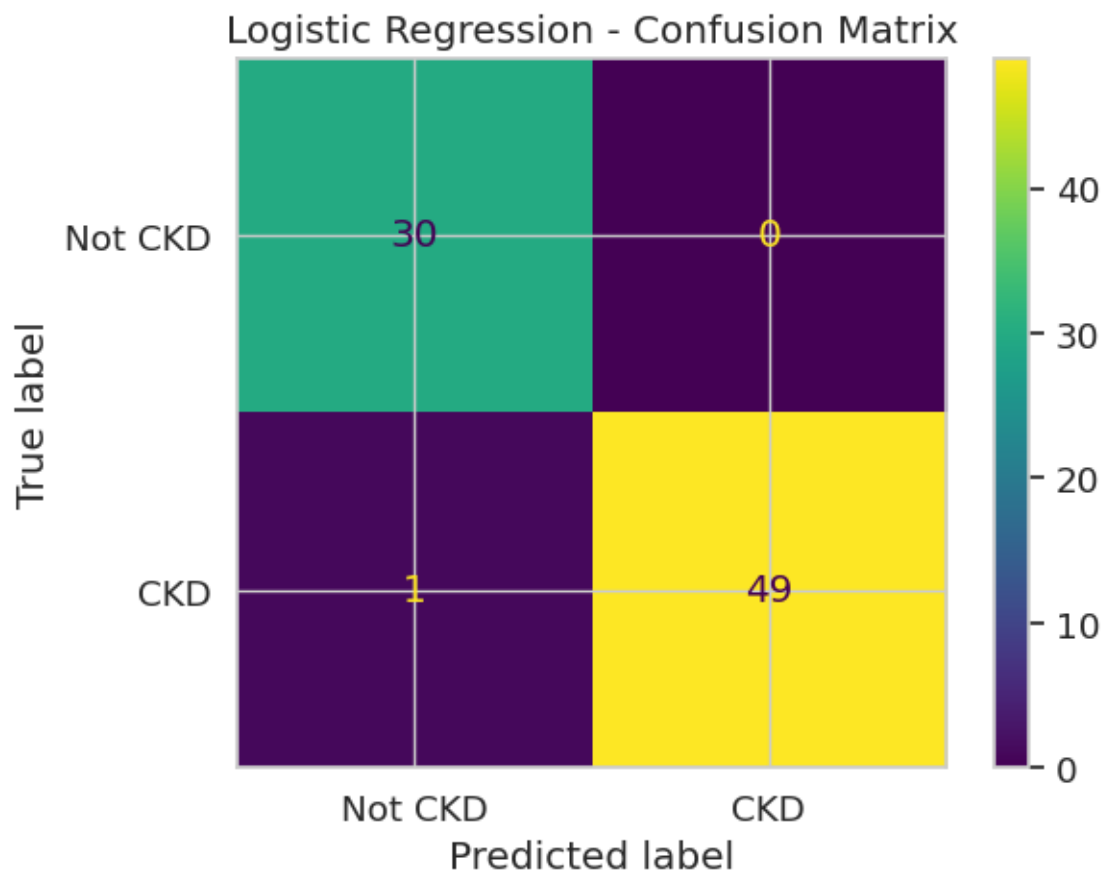
Milestone 5: Performance Testing & Evaluate the results

Activity 1: 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.

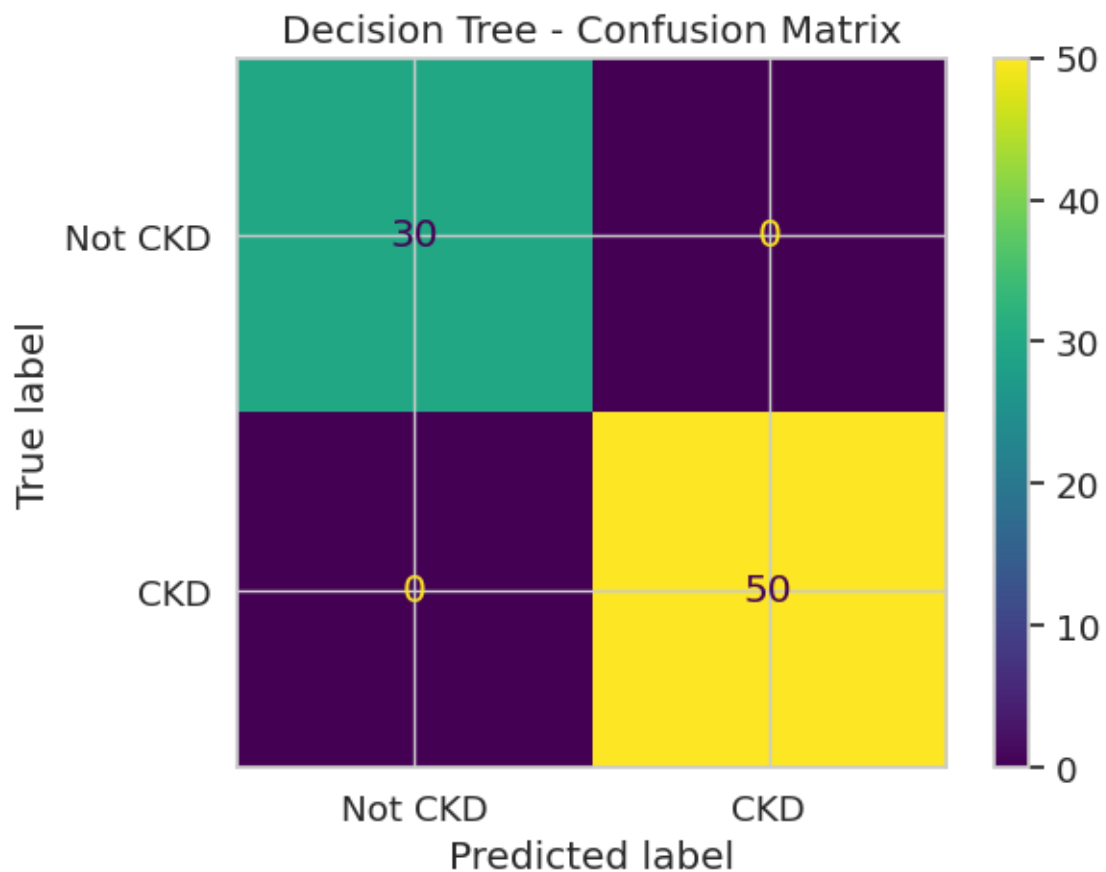
```
1 from sklearn import model_selection
2 import pandas as pd
3 import numpy as np # Import numpy
4
5 models = [
6     ('LogReg', log_model),
7     ('DecisionTree', dt_model),
8     ('RandomForest', rf_model)
9 ]
10
11 results = {}
12 scoring = ['accuracy', 'precision_weighted', 'recall_weighted', 'f1_weighted'] # Exclude roc_auc for now
13
14 # Impute missing values in relevant columns using the mode from the training data
15 for col in x_train.columns:
16     if x_train[col].isnull().any(): # Check if the column has any missing values
17         mode_train = x_train[col].mode()[0] # Calculate mode from training data
18         x_train[col] = x_train[col].fillna(mode_train) # Fill train set
19         x_test[col] = x_test[col].fillna(mode_train) # Fill test set using mode from train set
20
21
22 for name, model in models:
23     kfold = model_selection.KFold(n_splits=5, shuffle=True, random_state=90210)
24     cv_results = model_selection.cross_validate(model, x_train, y_train, cv=kfold, scoring=scoring)
25     results[name] = cv_results
26     print(f"{name} cross-validation results:")
27     for metric in scoring:
28         print(f"    {metric}: {np.mean(cv_results[f'test_{metric}']):.4f} (+/- {np.std(cv_results[f'test_{metric}']):.4f})")
29     print("-" * 20)
```

```
1 from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
2 import matplotlib.pyplot as plt
3 from sklearn.linear_model import LogisticRegression # Import LogisticRegression
4 from sklearn.preprocessing import LabelEncoder # Import LabelEncoder
5
6
7 # Instantiate LabelEncoder
8 le = LabelEncoder()
9
10 # Apply Label Encoding to the relevant columns in x_train and x_test
11 for col in ['diabetesmellitus', 'coronary_artery_disease']:
12     # Ensure that the column exists before attempting to encode
13     if col in x_train.columns:
14         # Handle potential NaN values by converting to string before encoding
15         x_train[col] = le.fit_transform(x_train[col].astype(str))
16         x_test[col] = le.transform(x_test[col].astype(str))
17
18
19 # Define and fit the Logistic Regression model on the binary target
20 log_model = LogisticRegression(max_iter=1000)
21 log_model.fit(x_train, y_train)
22
23 # Get predictions on the binary test set
24 log_pred = log_model.predict(x_test)
25
26 # Calculate and display the confusion matrix
27 log_cm = confusion_matrix(y_test, log_pred)
28 ConfusionMatrixDisplay(log_cm, display_labels=['Not CKD', 'CKD']).plot() # Updated display_labels for binary classification
29 plt.title("Logistic Regression - Confusion Matrix")
30 plt.show()
```



```
1 dfs = []
2 this_df = pd.DataFrame(cv_results)
3 this_df['model'] = name
4 dfs.append(this_df)

1 from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
2 import matplotlib.pyplot as plt
3 from sklearn.tree import DecisionTreeClassifier # Import DecisionTreeClassifier
4
5 # Ensure x_train and x_test are DataFrames before imputation (assuming imputation was done previously in NP5LW0Q0hc7t)
6 # If imputation is needed here, add the imputation code block
7
8 # Define and fit the Decision Tree model on the binary target
9 dt_model = DecisionTreeClassifier()
10 dt_model.fit(x_train, y_train)
11
12 # Get predictions on the binary test set
13 dt_pred = dt_model.predict(x_test)
14
15 # Calculate and display the confusion matrix
16 dt_cm = confusion_matrix(y_test, dt_pred)
17 ConfusionMatrixDisplay(dt_cm, display_labels=['Not CKD', 'CKD']).plot() # Updated display_labels for binary classification
18 plt.title("Decision Tree - Confusion Matrix")
19 plt.show()
```



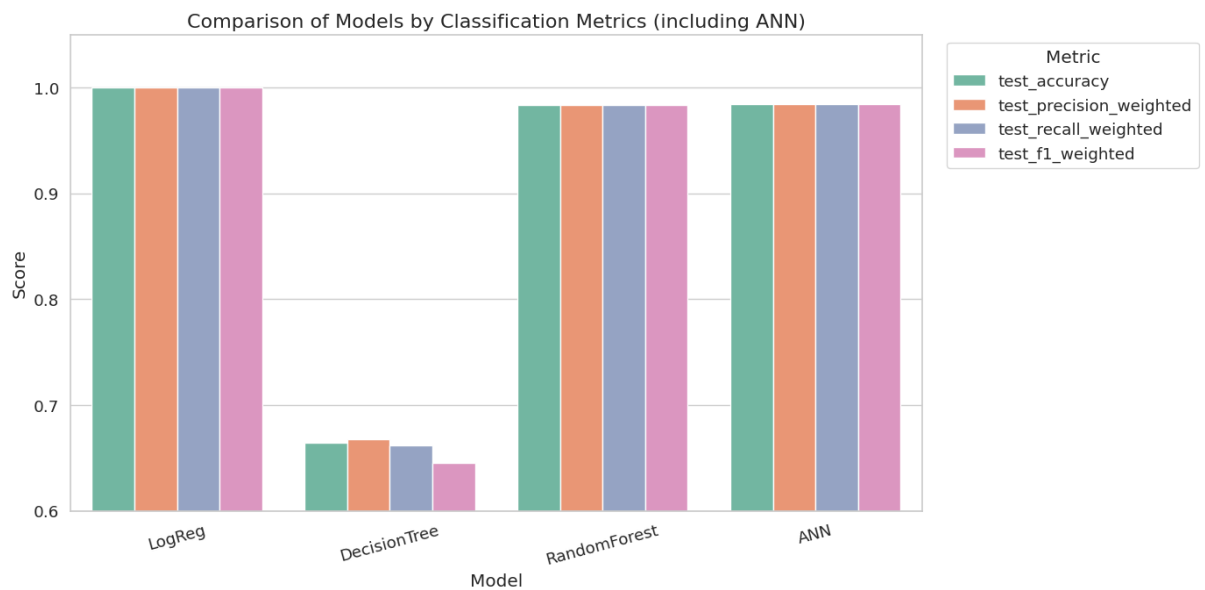
```
1 this_df = pd.DataFrame(cv_results)
2 this_df['model'] = name # IMPORTANT!
3 dfs.append(this_df)

1 from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
2 import matplotlib.pyplot as plt
3 from sklearn.ensemble import RandomForestClassifier # Import RandomForestClassifier
4
5 # Ensure X_train and X_test are DataFrames before imputation
6
7 # Define and fit the Random Forest model on the binary target
8 rf_model = RandomForestClassifier(n_estimators=100)
9 rf_model.fit(x_train, y_train)
10
11 # Get predictions on the binary test set
12 rf_pred = rf_model.predict(x_test)
13
14 # Calculate and display the confusion matrix
15 rf_cm = confusion_matrix(y_test, rf_pred)
16 ConfusionMatrixDisplay(rf_cm, display_labels=['Not CKD', 'CKD']).plot() # Updated display_labels for binary classification
17 plt.title("Random Forest - Confusion Matrix")
18 plt.show()
```

All above models are performing well for this dataset.

Activity 2: Evaluate the results

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4
5 comparison_data = pd.DataFrame({
6     'Model': ['LogReg', 'LogReg', 'LogReg', 'LogReg',
7             'DecisionTree', 'DecisionTree', 'DecisionTree', 'DecisionTree',
8             'RandomForest', 'RandomForest', 'RandomForest', 'RandomForest',
9             'ANN', 'ANN', 'ANN', 'ANN'],
10    'Metric': ['test_accuracy', 'test_precision_weighted', 'test_recall_weighted', 'test_f1_weighted'] * 4,
11    'Score': [1.0, 1.0, 1.0, 1.0,
12             0.664, 0.668, 0.662, 0.645,
13             0.983, 0.983, 0.983, 0.983,
14             0.984, 0.984, 0.984, 0.984]
15 })
16
17 plt.figure(figsize=(14, 7))
18 sns.set(style="whitegrid", font_scale=1.2)
19 sns.barplot(data=comparison_data, x='Model', y='Score', hue='Metric', palette='Set2')
20
21 plt.title('Comparison of Models by Classification Metrics (including ANN)', fontsize=16)
22 plt.ylim(0.6, 1.05)
23 plt.xticks(rotation=15)
24 plt.xlabel('Model')
25 plt.ylabel('Score')
26 plt.legend(title='Metric', bbox_to_anchor=(1.02, 1), loc='upper left')
27 plt.tight_layout()
28 plt.show()
```



Milestone6:

Model Deployment

Activity 1: 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.

```
# Save the model
with open('logistic_model.pkl', 'wb') as file:
    pickle.dump(logreg, file)
```

Activity 2: 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

Activity 2.1: Building Html Pages:

For this project create four HTML files namely

- home.html
- result.html

and save them in the templates folder.

Activity 2.2: 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

Load the model:

```
# Load the model
with open("logistic_model.pkl", "rb") as file:
    model = pickle.load(file)
```

Render HTML page:

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

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 be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.

Retrieves the value from UI:

```
@app.route('/predict', methods=['POST'])
def predict():
    try:
        rbc = 1 if request.form['red_blood_cells'] == 'abnormal' else 0
        pc = 1 if request.form['pus_cell'] == 'abnormal' else 0
        bgr = float(request.form['blood_glucose_random'])
        bu = float(request.form['blood_urea'])
        pe = 1 if request.form['pedal_edema'] == 'yes' else 0
        ane = 1 if request.form['anemia'] == 'yes' else 0
        dm = 1 if request.form['diabetesmellitus'] == 'yes' else 0
        cad = 1 if request.form['coronary_artery_disease'] == 'yes' else 0

        features = np.array([[rbc, pc, bgr, bu, pe, ane, dm, cad]])
        prediction = model.predict(features)[0]

        result = "CKD Detected" if prediction == 0 else "No CKD Detected"
        return render_template('result.html', prediction_text=result)

    except Exception as e:
        return f"Error: {str(e)}"

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

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.

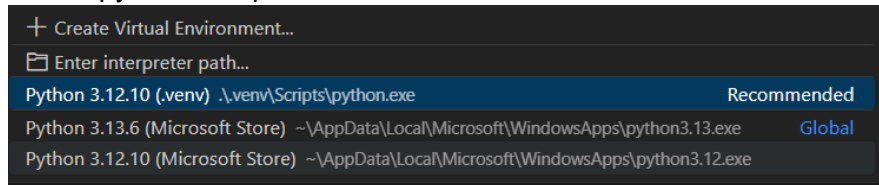
```
result = "CKD Detected" if prediction == 0 else "No CKD Detected"
return render_template('result.html', prediction_text=result)
```

Main Function:

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

Activity 2.3: Run the web application

- Open VSCODE from the start menu
- Navigate to the folder where your python script is. Then press ctrl+shift+p.
- Select python interpreter




- Activate the environment from the terminal by typing : `.venv\Scripts\activate`
- Now type “python app.py” command in the terminal.
- 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.

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

A screenshot of a web application titled 'Chronic Kidney Disease' with the subtitle 'A Machine Learning Web App, Built with Flask'. The interface features a purple header and a white form overlay. The form is titled 'Enter Patient Details' and contains several input fields: 'Red Blood Cells' (dropdown menu with 'Normal' selected), 'Pus Cell' (dropdown menu with 'Normal' selected), 'Blood Glucose Random' (text input), 'Blood Urea' (text input), 'Pedal Edema' (dropdown menu with 'No' selected), 'Anemia' (dropdown menu with 'No' selected), 'Diabetes Mellitus' (dropdown menu with 'No' selected), and 'Coronary Artery Disease' (dropdown menu with 'No' selected). A red 'Predict' button is located at the bottom of the form. The background of the web page shows a person's hands typing on a laptop keyboard with a stethoscope resting on the desk.

Prediction Result

No CKD Detected

 Predict Again

Prediction Result

CKD Detected

 Predict Again

In this project, we developed a machine learning-based web application to predict Chronic Kidney Disease (CKD) using patient health data. By leveraging a Logistic Regression model trained on 24 key clinical features—including blood pressure, specific gravity, red and white blood cell counts, and various biochemical indicators—we achieved a reliable method to assist in early detection of CKD.

The model was integrated into a Flask-based web interface that allows users to input health parameters and receive instant predictions regarding CKD risk. Proper feature alignment between the model and input form was ensured to maintain prediction accuracy and prevent input mismatch errors.

This solution can support healthcare professionals by providing a quick and accessible tool for preliminary diagnosis, ultimately contributing to earlier intervention and better patient outcomes. Future enhancements may include:

- Testing more advanced models like Random Forest and Logistic regression for higher accuracy,
- Adding data visualization,
- Implementing explainable AI techniques for transparent decision-making,
- And deploying the application on cloud platforms for broader accessibility.

Overall, this project demonstrates the powerful role machine learning can play in predictive healthcare and decision support systems.