Classification Assignment

Problem Statement or Requirement:

A requirement from the Hospital, Management asked us to create a predictive model which will predict the Chronic Kidney Disease (CKD) based on the several parameters. The Client has provided the dataset of the same.

1.) Identify your problem statement

2.) Tell basic info about the dataset (Total number of rows, columns)

3.) Mention the pre-processing method if you’re doing any (like converting string to number – nominal data)

4.) Develop a good model with good evaluation metric. You can use any machine learning algorithm; you can create many models. Finally, you have to come up with final model.

5.) All the research values of each algorithm should be documented. (You can make tabulation or screenshot of the results.)

6.) Mention your final model, justify why u have chosen the same.

1. The hospital wants a reliable predictive model to classify patients into CKD (Chronic Kidney Disease) or Not CKD, using medical attributes. You are tasked with building the best-performing classification model, evaluated using precision, recall, f1-score, and accuracy.

Domain: ML

Type: Supervised Learning

Objective: Binary Classification

In [84]:
```
#importing the Libraies
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [85]:
```
# Reading the Dataset
dataset = pd.read_csv('CKD.csv')
```

2. Tell basic info about the dataset (Total number of rows, columns)

In [86]:
```
print(f"\nRows: {dataset.shape[0]}")
print(f"\nColumns: {dataset.shape[1]}\n\n")

display(dataset)

# Displaying the dataset information
display(dataset.info())
```

```
Rows: 399

Columns: 25
```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
0	2.000000	76.459948	c	3.0	0.0	normal	abnormal	notpresent	notpresent	148.112676	...	38.868902	8408.191126	4.705597	no	no	no	yes	yes	no	yes
1	3.000000	76.459948	c	2.0	0.0	normal	normal	notpresent	notpresent	148.112676	...	34.000000	12300.000000	4.705597	no	no	no	yes	poor	no	yes
2	4.000000	76.459948	a	1.0	0.0	normal	normal	notpresent	notpresent	99.000000	...	34.000000	8408.191126	4.705597	no	no	no	yes	poor	no	yes
3	5.000000	76.459948	d	1.0	0.0	normal	normal	notpresent	notpresent	148.112676	...	38.868902	8408.191126	4.705597	no	no	no	yes	poor	yes	yes
4	5.000000	50.000000	c	0.0	0.0	normal	normal	notpresent	notpresent	148.112676	...	36.000000	12400.000000	4.705597	no	no	no	yes	poor	no	yes
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
394	51.492308	70.000000	a	0.0	0.0	normal	normal	notpresent	notpresent	219.000000	...	37.000000	9800.000000	4.400000	no	no	no	yes	poor	no	yes
395	51.492308	70.000000	c	0.0	2.0	normal	normal	notpresent	notpresent	220.000000	...	27.000000	8408.191126	4.705597	yes	yes	no	yes	poor	yes	yes
396	51.492308	70.000000	c	3.0	0.0	normal	normal	notpresent	notpresent	110.000000	...	26.000000	9200.000000	3.400000	yes	yes	no	poor	poor	no	yes
397	51.492308	90.000000	a	0.0	0.0	normal	normal	notpresent	notpresent	207.000000	...	38.868902	8408.191126	4.705597	yes	yes	no	yes	poor	yes	yes
398	51.492308	80.000000	a	0.0	0.0	normal	normal	notpresent	notpresent	100.000000	...	53.000000	8500.000000	4.900000	no	no	no	yes	poor	no	no

399 rows × 25 columns

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 399 entries, 0 to 398
Data columns (total 25 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   age             399 non-null    float64
 1   bp              399 non-null    float64
 2   sg              399 non-null    object 
 3   al              399 non-null    float64
 4   su              399 non-null    float64
 5   rbc             399 non-null    object 
 6   pc              399 non-null    object 
 7   pcc             399 non-null    object 
 8   ba              399 non-null    object 
 9   bgr             399 non-null    float64
 10  bu              399 non-null    float64
 11  sc              399 non-null    float64
 12  sod             399 non-null    float64
 13  pot             399 non-null    float64
 14  hrmo            399 non-null    float64
 15  pcv             399 non-null    float64
 16  wc              399 non-null    float64
 17  rc              399 non-null    float64
 18  htn             399 non-null    object 
 19  dm              399 non-null    object 
 20  cad             399 non-null    object 
 21  appet           399 non-null    object 
 22  pe              399 non-null    object 
 23  ane             399 non-null    object 
 24  classification  399 non-null    object 
dtypes: float64(13), object(12)
memory usage: 78.1+ KB
None
```

3.) Mention the pre-processing method if you're doing any (like converting string to number – nominal data)

In [87]:

```
# Converting categorical variables to numerical
# as the classification column is categorical - ordinal data, will be converted to numerical values using LabelEncoder
from sklearn.preprocessing import LabelEncoder

# Handle categorical and numerical separately
categorical_cols = dataset.select_dtypes(include='object').columns
numerical_cols = dataset.select_dtypes(exclude='object').columns

# Fill missing values
dataset[categorical_cols] = dataset[categorical_cols].fillna(dataset[categorical_cols].mode().iloc[0])
```

```python
dataset[numerical_cols] = dataset[numerical_cols].fillna(dataset[numerical_cols].mean())

# Encode categorical columns
label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    dataset[col] = le.fit_transform(dataset[col])
    label_encoders[col] = le
```

4. Develop a good model with good evaluation metric. You can use any machine learning algorithm; you can create many models. Finally, you have to come up with final model.

In [88]:
```python
# Splitting the dataset into independent and dependent variables
independent = dataset.drop('classification', axis=1)
dependent = dataset['classification']
```

In [89]:
```python
#split into training set and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(independent, dependent, test_size = 1/3, random_state = 0)
```

In [90]:
```python
from sklearn.preprocessing import StandardScaler
StandardScaler = StandardScaler()
X_train = StandardScaler.fit_transform(X_train)
X_test = StandardScaler.transform(X_test)
```

5. All the research values of each algorithm should be documented. (You can make tabulation or screenshot of the results.)

The report using LogisticRegression :

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        51
           1       1.00      0.98      0.99        82

    accuracy                           0.98       133
```

macro avg 0.98 0.99 0.98 133 weighted avg 0.99 0.98 0.99 133

The report using K-Nearest_Neibour KNN :

```
              precision    recall  f1-score   support

           0       0.93      1.00      0.96        51
           1       1.00      0.95      0.97        82

    accuracy                           0.97       133
```

macro avg 0.96 0.98 0.97 133 weighted avg 0.97 0.97 0.97 133

The report using GaussianNB :

```
              precision    recall  f1-score   support

           0       0.91      1.00      0.95        51
           1       1.00      0.94      0.97        82

    accuracy                           0.96       133
```

macro avg 0.96 0.97 0.96 133 weighted avg 0.97 0.96 0.96 133

The report using MultinomialNB :

```
              precision    recall  f1-score   support

           0       0.39      1.00      0.56        51
           1       1.00      0.04      0.07        82

    accuracy                           0.41       133
```

macro avg 0.70 0.52 0.32 133 weighted avg 0.77 0.41 0.26 133

The report using BernoulliNB :

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        51
           1       1.00      0.98      0.99        82

    accuracy                           0.98       133
```

macro avg 0.98 0.99 0.98 133 weighted avg 0.99 0.98 0.99 133

The report using ComplementNB :

```
              precision    recall  f1-score   support

           0       0.40      1.00      0.57        51
           1       1.00      0.05      0.09        82

    accuracy                           0.41       133
```

macro avg 0.70 0.52 0.33 133 weighted avg 0.77 0.41 0.27 133

The report using CategoricalNB :

```
              precision    recall  f1-score   support

         0.0       0.98      1.00      0.99        51
         1.0       1.00      0.99      0.99        82

    accuracy                           0.99       133
```

macro avg 0.99 0.99 0.99 133 weighted avg 0.99 0.99 0.99 133

The report using SVC :

```
              precision    recall  f1-score   support

           0       0.94      0.96      0.95        51
           1       0.98      0.96      0.97        82

    accuracy                           0.96       133
```

macro avg 0.96 0.96 0.96 133 weighted avg 0.96 0.96 0.96 133

The report using RandomForestClassifier :

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        51
           1       1.00      1.00      1.00        82
```

```
   accuracy                           1.00       133
```

macro avg 1.00 1.00 1.00 133 weighted avg 1.00 1.00 1.00 133

The report using DecisionTreeClassifier :

```
              precision    recall  f1-score   support

           0       0.86      0.98      0.92        51
           1       0.99      0.90      0.94        82

    accuracy                           0.93       133
```

macro avg 0.92 0.94 0.93 133 weighted avg 0.94 0.93 0.93 133

6. Mention your final model, justify why u have chosen the same.

After evaluating multiple classification algorithms, RandomForestClassifier was chosen as the final model due to its perfect classification performance with 100% accuracy, precision, recall, and F1-score on the test dataset. In addition to superior performance, its ability to handle both numerical and categorical data, resistance to overfitting, and interpretability through feature importance make it the most appropriate choice for chronic kidney disease prediction.

Justification:

🔹 1. Superior Performance RandomForestClassifier delivers perfect classification on the test data. While other models like CategoricalNB, BernoulliNB, and LogisticRegression come very close with ~98–99% accuracy, RandomForest achieves a clean 1.00 across all evaluation metrics.

🔹 2. Handles Both Numerical and Categorical Data Random Forests are versatile—they do not require strict feature scaling or encoding schemes, unlike SVM or Naive Bayes models. This is particularly useful in medical datasets like CKD, which often contain mixed data types.

🔹 3. Robust to Noise & Outliers Ensemble methods like RandomForest are less likely to overfit, especially compared to Decision Trees or models that rely on distribution assumptions (e.g., GaussianNB).

🔹 4. Feature Importance Random Forests allow you to interpret feature importance, which is useful in a clinical setting to understand which parameters (e.g., serum creatinine, blood pressure) contribute most to CKD risk.

Final Model: RandomForestClassifier

Performance:

Accuracy: 100%

Precision, Recall, F1-Score: All metrics are 1.00 for both classes.

Support: Balanced performance on both CKD (label 1) and non-CKD (label 0) classes.