

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

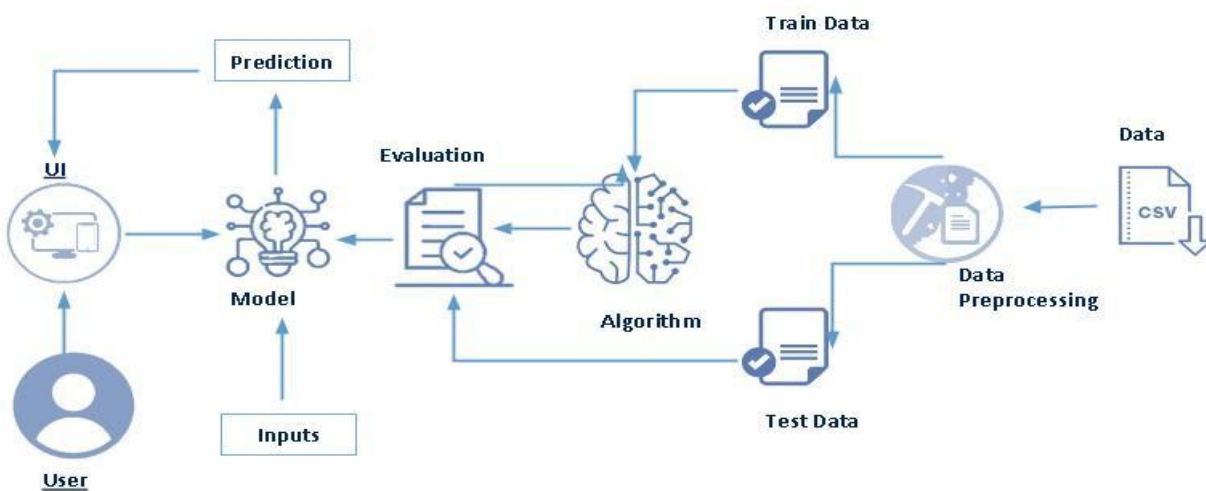
## Early Prediction for Chronic Kidney Disease Detection Using Machine Learning

Chronic Kidney Disease (CKD) is a long-term condition where the kidneys lose their ability to function effectively over time. Early stages of CKD often show no symptoms, making timely diagnosis difficult. If left undetected, CKD can progress to kidney failure, requiring dialysis or transplantation, and may lead to severe complications or even death.

In today's healthcare systems, early detection of CKD plays a vital role in preventing critical outcomes, reducing treatment costs, and improving patient survival rates. However, manual diagnosis based on medical tests can be time-consuming and prone to human error, especially when evaluating complex combinations of clinical parameters.

To address this issue, our project aims to develop a machine learning-based web application that can accurately predict the presence of CKD using 24 key medical input parameters. The goal is to support doctors, hospitals, and patients with a fast, easy-to-use, and reliable system that helps in the early identification of CKD risk. This proactive approach enables early intervention and personalized care strategies, potentially saving lives and reducing healthcare burdens.

### Technical Architecture:



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

## Project Flow: CKD Prediction Web App

- The user interacts with the web interface to enter 24 medical input parameters such as age, blood pressure, sugar level, and creatinine levels.
- These inputs are sent to a trained machine learning model (SVM), integrated within a Flask web framework.
- The model analyzes the inputs and instantly predicts whether the patient is likely to have Chronic Kidney Disease or not.
- The prediction result is displayed on the screen with a clear message for the user.

## Milestone 1: Define Problem / Problem Understanding

### Activity 1: Specify the Business Problem

Chronic Kidney Disease (CKD) is a growing health concern globally. It occurs when the kidneys gradually lose function over time, often without visible symptoms in the early stages. If not diagnosed early, CKD can lead to kidney failure, requiring dialysis or transplantation, which can be both life-threatening and financially burdensome.

Manual diagnosis requires interpreting a combination of medical parameters, which can be time-consuming and error-prone. Hence, there is a need for a machine learning-based solution that can predict the risk of CKD using clinical inputs. This system can assist doctors in identifying high-risk patients at an early stage.

### Activity 2: Business Requirements

A CKD prediction system must meet several business and clinical requirements to be effective and scalable:

- **Accuracy and Reliability:** The system must produce accurate predictions using up-to-date, medically relevant patient data to ensure trustworthy results.
- **Early Diagnosis Support:** It should detect early signs of CKD to reduce the burden on healthcare systems and prevent further complications.
- **Flexibility and Scalability:** The system should be flexible to integrate with hospital record systems and scalable for large datasets or multiple clinics.
- **Compliance and Privacy:** The system must comply with patient data protection policies and healthcare data regulations (e.g., HIPAA).

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

- **User-Friendly Interface:** The interface should be simple and intuitive for both healthcare professionals and patients, even those with limited technical knowledge.
- **Low Latency:** The model should provide predictions quickly to be useful in real-time scenarios like outpatient checkups.

## Activity 3: Literature Survey

A literature review was conducted to understand the existing work in the field of CKD prediction using machine learning. Previous studies have shown that classifiers such as Support Vector Machines (SVM), Logistic Regression, Decision Trees, and Random Forests have been used with high success rates in predicting medical outcomes.

Research shows that SVMs are effective in binary classification tasks like CKD detection. Most existing systems are not web-deployed or user-friendly, indicating a gap that this project aims to fill. Various research papers also emphasized the importance of preprocessing (handling missing values, normalization) for medical datasets.

## Activity 4: Social or Business Impact

- **Social Impact:**  
Early prediction of CKD can greatly enhance patient care by enabling early intervention, reducing health complications, and potentially saving lives. It can also empower rural or underserved areas with limited access to nephrologists by acting as a support tool for basic screening.
- **Business Impact:**  
Hospitals and diagnostic labs can integrate such a tool to automate basic screening, reduce manual effort, and speed up preliminary diagnosis. It also opens opportunities for startups and health-tech platforms to develop AI-powered healthcare solutions that reduce costs and improve outcomes.

## Milestone 2: Data Collection & Preparation

Machine learning relies heavily on data — it is the core that enables the model to learn patterns and make predictions. In this milestone, we collected and prepared a dataset related to Chronic Kidney Disease (CKD).

### Activity 1: Collect the Dataset

There are several open sources for datasets such as Kaggle, UCI ML Repository, etc.

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

In this project, we used a `.csv` dataset collected from Kaggle.

- **Dataset Name:** Chronic Kidney Disease Dataset
- **Source:** <https://www.kaggle.com/datasets/mansoordaku/ckdisease>
- **Format:** CSV
- **Total Records:** ~400 patient records
- **Features:** 24 medical input parameters (like age, bp, sc, hemo, al, etc.) and 1 classification label

## 📌 Activity 1.1: Importing the Libraries

```
# Basic Libraries
import pandas as pd          # For data manipulation
import numpy as np          # For numerical operations
import matplotlib.pyplot as plt # For plotting graphs
import seaborn as sns        # For advanced data visualizations

# Preprocessing & Utilities
import warnings              # To ignore warnings
warnings.filterwarnings("ignore")

from sklearn.model_selection import train_test_split # For splitting data
from sklearn.preprocessing import LabelEncoder      # Encoding categorical features
from sklearn.preprocessing import StandardScaler     # Normalization

# Machine Learning Models
from sklearn.svm import SVC          # Support Vector Machine
from sklearn.linear_model import LogisticRegression # Logistic Regression
from sklearn.ensemble import RandomForestClassifier # Random Forest Classifier
from sklearn.neighbors import KNeighborsClassifier # KNN Classifier
from sklearn.tree import DecisionTreeClassifier   # Decision Tree Classifier

# Evaluation Metrics
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Model Saving
import joblib                        # To save and load trained models
```

## 📌 Activity 1.2: Read the Dataset

Our dataset was in `.csv` format, which is commonly used for structured data. To read and load the dataset into a `DataFrame`, we used the `pandas` library in Python.

In `pandas`, the function `read_csv()` is used to load `.csv` files. We passed the path of the file as a parameter to this function.

For checking the null values, `df.isna().any()` function is used. To sum those null values we use `.sum()` function. From the below image, we found that there are no null values present in our dataset. So we can skip handling the missing values step

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

```
In [2]: import pandas as pd
import numpy as np

df=pd.read_csv(r"C:\Users\yvenk\Downloads\chronickidneydisease.csv")
df
```

Out[2]:

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	cla
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	
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	
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	
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	
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	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
395	395	55.0	80.0	1.020	0.0	0.0	normal	normal	notpresent	notpresent	...	47	6700	4.9	no	no	no	good	no	no	
396	396	42.0	70.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	...	54	7800	6.2	no	no	no	good	no	no	
397	397	12.0	80.0	1.020	0.0	0.0	normal	normal	notpresent	notpresent	...	49	6600	5.4	no	no	no	good	no	no	
398	398	17.0	60.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	...	51	7200	5.9	no	no	no	good	no	no	
399	399	58.0	80.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	...	53	6800	6.1	no	no	no	good	no	no	

400 rows × 26 columns

## Activity 2: Data Preparation

As we have now explored the dataset structure, we moved to **preprocessing**. The downloaded CKD dataset was not directly suitable for training a machine learning model due to:

- Missing values represented by -?
- Categorical variables
- Unscaled numeric features

Therefore, data preparation was necessary to clean and convert it into a machine-readable format. The steps involved are:

- Handling missing values
- Encoding categorical variables
- Scaling numerical features
- Optional: Handling outliers

### ☒ Activity 2.1: Handling Missing Values

For checking the count of null values, `df.isnull().sum()` function is used. From the below image, we found that there are many null values present in our dataset. So we will handle the missing values.

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

```
<bound method DataFrame.info of
0      0  48.0  80.0  1.020  1.0  0.0  NaN  normal  notpresent
1      1   7.0  50.0  1.020  4.0  0.0  NaN  normal  notpresent
2      2  62.0  80.0  1.010  2.0  3.0  normal  normal  notpresent
3      3  48.0  70.0  1.005  4.0  0.0  normal  abnormal  present
4      4  51.0  80.0  1.010  2.0  0.0  normal  normal  notpresent
..    ...   ...   ...   ...   ...   ...   ...   ...
395  395  55.0  80.0  1.020  0.0  0.0  normal  normal  notpresent
396  396  42.0  70.0  1.025  0.0  0.0  normal  normal  notpresent
397  397  12.0  80.0  1.020  0.0  0.0  normal  normal  notpresent
398  398  17.0  60.0  1.025  0.0  0.0  normal  normal  notpresent
399  399  58.0  80.0  1.025  0.0  0.0  normal  normal  notpresent

      ba  ...  pcv  wc  rc  htn  dm  cad  appet  pe  ane  \
0  notpresent  ...  44  7800  5.2  yes  yes  no  good  no  no
1  notpresent  ...  38  6000  NaN  no  no  no  good  no  no
2  notpresent  ...  31  7500  NaN  no  yes  no  poor  no  yes
3  notpresent  ...  32  6700  3.9  yes  no  no  poor  yes  yes
4  notpresent  ...  35  7300  4.6  no  no  no  good  no  no
..    ...   ...   ...   ...   ...   ...   ...   ...
395  notpresent  ...  47  6700  4.9  no  no  no  good  no  no
396  notpresent  ...  54  7800  6.2  no  no  no  good  no  no
397  notpresent  ...  49  6600  5.4  no  no  no  good  no  no
398  notpresent  ...  51  7200  5.9  no  no  no  good  no  no
399  notpresent  ...  53  6800  6.1  no  no  no  good  no  no

classification
0      ckd
1      ckd
2      ckd
3      ckd
4      ckd
..    ...
395  notckd
396  notckd
397  notckd
398  notckd
399  notckd

[400 rows x 26 columns]>
```

```
In [4]: df.isnull().sum()
```

```
Out[4]: id          0
age           9
bp           12
sg           47
al           46
su           49
rbc          152
pc           65
pcc           4
ba            4
bgr          44
bu           19
sc           17
sod           87
pot           88
hemo         52
pcv           70
wc          105
rc           130
htn           2
dm            2
cad           2
appet         1
pe            1
ane           1
classification    0
dtype: int64
```

## 🔗 Activity 2.2: Handling Outliers & Missing Data Intelligently

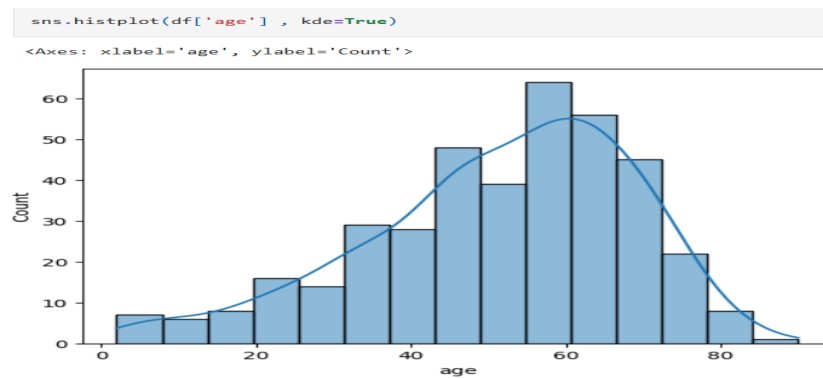
# Early Prediction for Chronic Kidney Disease Detection: A Progressive Approach to Health Management

Instead of classic outlier handling, our CKD dataset had a significant number of **missing values**, which needed to be handled carefully before training the machine learning model.

We followed a column-wise strategy based on **data type** and **distribution**:

## For Numerical Columns:

We first checked for skewness and outliers using:

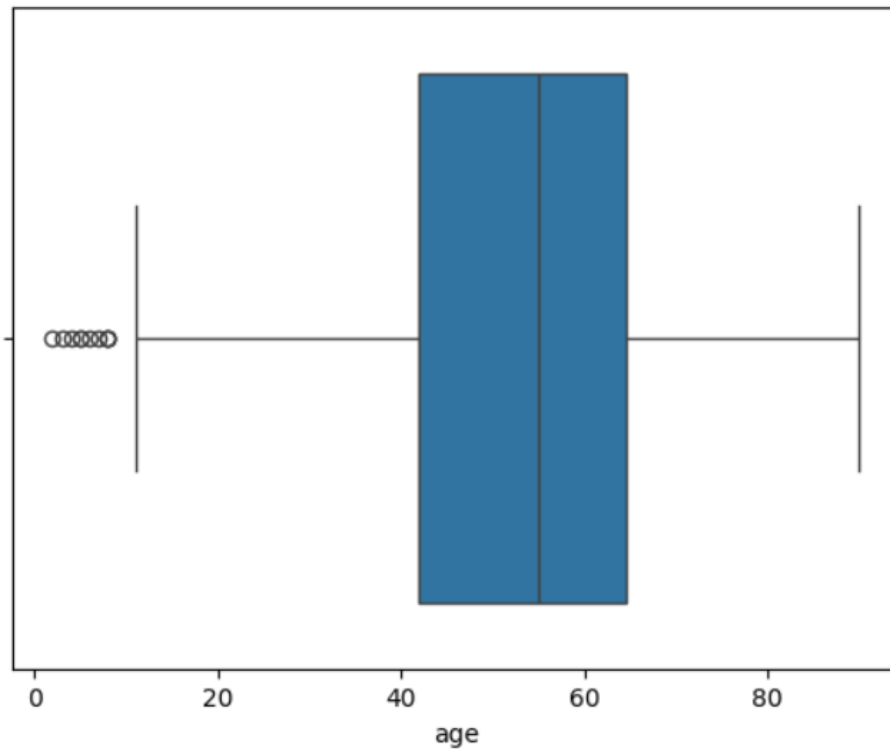


If the distribution was:

- **Normally distributed** → we used **mean** to fill missing values
- **Skewed (left or right)** → we used **median** to avoid distortion caused by outliers

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

Out[6]: <Axes: xlabel='age'>



```
df['age'] = df['age'].fillna(df['age'].median())
```

## ☒ For Categorical Columns:

For categorical features like `rbc`, `pc`, `appet`, etc., we checked the frequency distribution using:

Instead of directly using mode or random fills, we followed a **group-based imputation approach** to increase accuracy.**1. Strip whitespaces**

We first cleaned the data by removing unnecessary spaces from categorical entries to avoid duplication like " normal" vs "normal":

To ensure consistent category names before counting frequencies or grouping.

## ***2. We checked category counts using:***



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

```
df['rbc'].value_counts()

rbc
normal      201
abnormal     47
Name: count, dtype: int64

df['rbc'] = df['rbc'].str.strip()

df['rbc'] = df['rbc'].map({'normal': 0, 'abnormal': 1})
#Binary encoding
```

## 3. Group-wise Fill Using classification

Since patients with CKD and without CKD may show different characteristics, we filled missing values **grouped by the classification column**:

```
df['rbc'] = df['rbc'].fillna(df.groupby('classification')['rbc'].transform(lambda x: x.mode()[0]))
```

## For Numerical Columns:

We handled skewness and missing values like this:

- Used `histplot` to visualize distribution.
- Based on skewness:
  - **Median** for skewed columns (e.g., `sc`, `bgr`)
  - **Mean** for normally distributed ones (e.g., `age`)

## Using Median for Skewed Columns

Outliers affect the mean but not median.

So when you:

```
df['bp'] = df['bp'].fillna(df['bp'].median())
```

It **avoids the influence of extreme outliers**, because the **median stays stable** even if your column has very large/small values. So it won't *remove* outliers, but you're **not letting them distort your missing value replacement either**.

Outlier removal (like dropping rows or capping values) can cause **loss of important medical variation**.

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

This method **retains all rows** and **avoids outlier distortion indirectly**, making it safe for medical applications.

```
df.isnull().sum()
```

```
id          0
age         0
bp          0
sg          0
al          0
su          0
rbc         0
pc          0
pcc         0
ba          0
bgr         0
bu          0
sc          0
sod         0
pot         0
hemo        0
pcv         0
wc          0
rc          0
htn         0
dm          0
cad         0
appet       0
pe          0
ane         0
classification 0
dtype: int64
```

```
[18]: df.duplicated().value_counts()
```

```
[18]: False    400
      Name: count, dtype: int64
```

## Milestone 3: Exploratory Data Analysis

### 📌 Activity 1: Descriptive Statistical

Descriptive statistical analysis is performed to understand the basic properties and summary of the dataset. It provides insights into the distribution, central tendency, and spread of both numerical and categorical features.

In this project, we used the `pandas.describe()` function to analyze the continuous (numerical) columns. This function returns important statistics such as:

- **Mean:** Average value
- **Standard deviation (std):** Spread of the data
- **Min and Max:** Range of values

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

- **25%, 50%, 75%:** Percentile distributions (quartiles)

	id	age	bp	sg	al	su	bgr	bu	sc	sod	pot	hemo
count	400.000000	391.000000	388.000000	353.000000	354.000000	351.000000	356.000000	381.000000	383.000000	313.000000	312.000000	348.000000
mean	199.500000	51.483376	76.469072	1.017408	1.016949	0.450142	148.036517	57.425722	3.072454	137.528754	4.627244	12.526437
std	115.614301	17.169714	13.683637	0.005717	1.352679	1.099191	79.281714	50.503006	5.741126	10.408752	3.193904	2.912587
min	0.000000	2.000000	50.000000	1.005000	0.000000	0.000000	22.000000	1.500000	0.400000	4.500000	2.500000	3.100000
25%	99.750000	42.000000	70.000000	1.010000	0.000000	0.000000	99.000000	27.000000	0.900000	135.000000	3.800000	10.300000
50%	199.500000	55.000000	80.000000	1.020000	0.000000	0.000000	121.000000	42.000000	1.300000	138.000000	4.400000	12.650000
75%	299.250000	64.500000	80.000000	1.020000	2.000000	0.000000	163.000000	66.000000	2.800000	142.000000	4.900000	15.000000
max	399.000000	90.000000	180.000000	1.025000	5.000000	5.000000	490.000000	391.000000	76.000000	163.000000	47.000000	17.800000

## 📊 Activity 2: Visual Analysis

Visual analysis is the process of exploring data using plots and graphs to identify patterns, relationships, outliers, and trends. It helps in interpreting complex data distributions and making data-driven decisions.

In this project, we used various plots from **Seaborn** and **Matplotlib** to visually understand how the medical features relate to Chronic Kidney Disease.

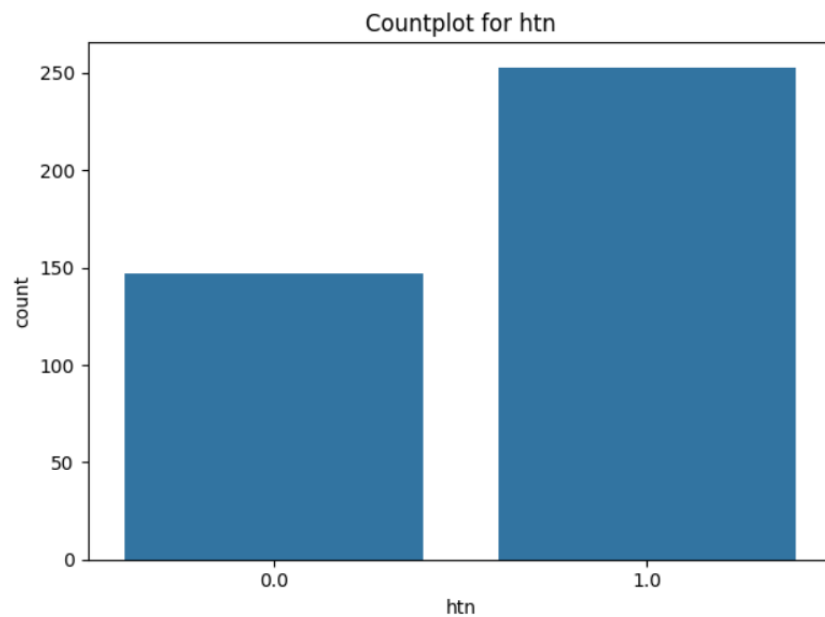
### 📊 Activity 2.1: Univariate Analysis

Univariate analysis involves analyzing one feature at a time to understand its distribution. For this, we used **count plots** and **pie charts** for categorical columns.

#### Countplot for the Target Feature – classification

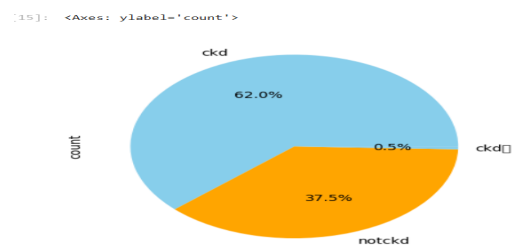
We used `sns.countplot()` to check the distribution of the target variable (CKD or Not CKD):

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



## Pie Chart Representation

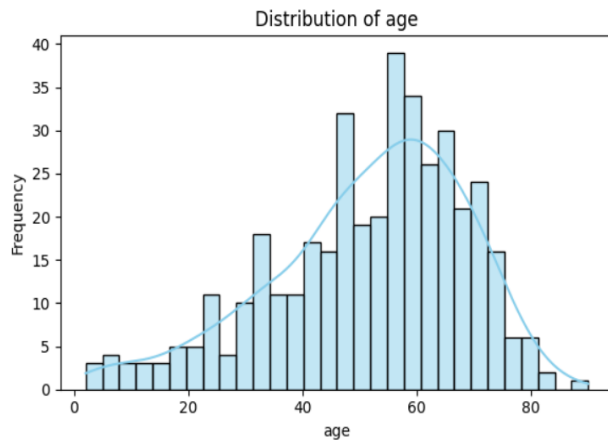
We also created a pie chart to visualize the percentage of CKD vs Not CKD classes



## Age Distribution (Numerical Feature)

We used a **histogram with KDE plot** to analyze the distribution of the `age` feature using Seaborn's `histplot()`:

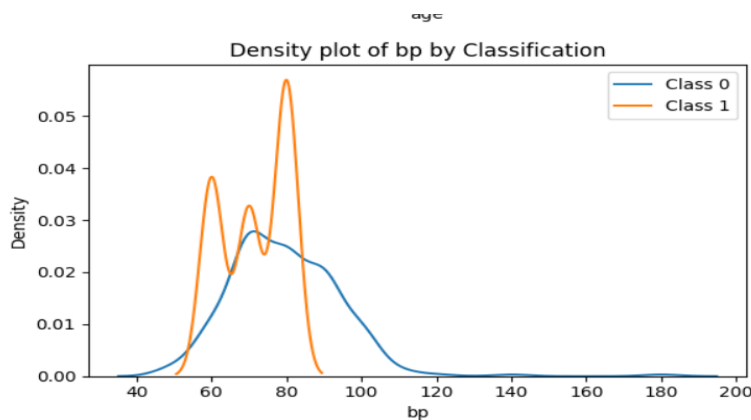
# Early Prediction for Chronic Kidney Disease Detection: A Progressive Approach to Health Management



## ✂ Activity 2.2: Bivariate Analysis

Bivariate analysis is used to understand the relationship between two variables. In this project, we focused on how different medical features vary across the target variable `classification` (CKD or Not CKD).

We used **barplots** for comparing categorical and numerical features with the target class.

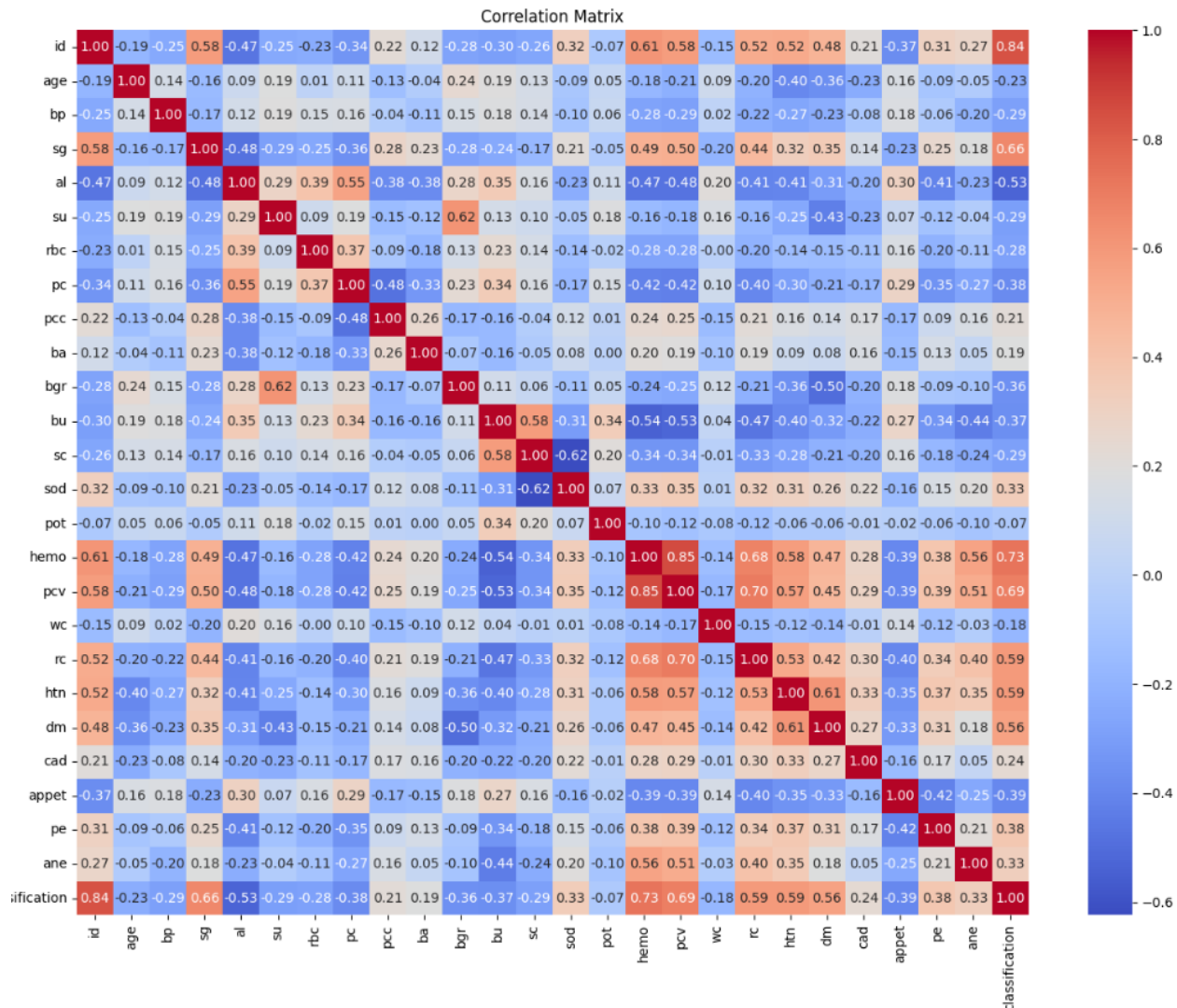


## ✂ Activity 2.3: Multivariate Analysis

In simple words, multivariate analysis is used to study the relationship between **multiple features** simultaneously. It helps us understand how features are related to one another and to the target variable.

In this project, we used a **correlation heatmap** provided by the Seaborn library to visualize the strength of relationships between numeric features

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



## Encoding the Categorical Features

- Most machine learning models require **numerical input**, and cannot work with string-based categorical data directly.
- In our CKD dataset, many categorical columns had **binary values** such as `yes/no`, `normal/abnormal`, or `present/notpresent`.
- So instead of using Label Encoding, we used a **simpler and more intuitive binary mapping technique** using Python's `.map()` function.
- .

```
df['ba']=df['ba'].map({'present':0 , 'notpresent':1})
```

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

## 🔍 Note:

- We ensured all string values were stripped of extra spaces using `.strip()` before applying `.map()`.
- This encoding was performed **after handling missing values**

## *Splitting Data into Train and Test Sets*

To evaluate the performance of our machine learning model, we first need to split the dataset into **training** and **testing** sets. This helps us test the model on unseen data and check how well it generalizes.

### Create input and output variables:

- `X` contains all **independent features** (input features)
- `y` contains the **target feature** (classification)

```
# Drop the target column to get features
X = df.drop(['classification'], axis=1)

# Target column
y = df['classification']
```

```
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=42, stratify=y)
```

### 2. Import the function and split the data:

We used the `train_test_split()` function from `sklearn.model_selection` to split the dataset.

- `test_size=0.2`: 20% of the data is used for testing
- `random_state=42`: Ensures reproducibility
- `stratify=y`: Ensures equal class distribution in train and test sets

## Scaling the Features

- Many machine learning models, especially **SVM**, are sensitive to the scale of input features.
- Some features in our dataset had large ranges (e.g., `bgr`, `sc`) while others were small, which could affect model performance.

To fix this, we applied **Standard Scaling**, which transforms the features to have:

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

- Mean = 0
- Standard deviation = 1

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

## Activity 1: Training the Model in Multiple Algorithms

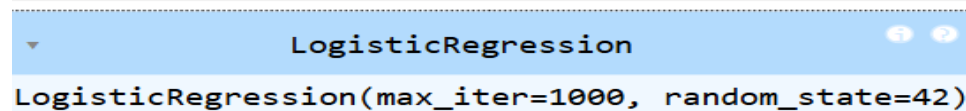
We trained our data on several supervised machine learning models. The goal was to compare their performance and select the best model based on **accuracy** and **evaluation metrics**.

### Activity 1.1: Logistic Regression Model

- LogisticRegression was imported from `sklearn.linear_model`.
- It is a simple and efficient linear classifier.
- The model was trained using `.fit()` and predictions were made using `.predict()`.

```
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression(max_iter=1000, random_state=42)
logreg.fit(X_train_scaled, y_train)
```



The screenshot shows a Jupyter Notebook cell with a blue header bar containing the text "LogisticRegression" and two icons (a magnifying glass and a refresh icon). Below the header, the code `LogisticRegression(max_iter=1000, random_state=42)` is displayed.

```
y_pred = logreg.predict(X_test_scaled)
```

### Activity 1.2: K-Nearest Neighbors (KNN) Model

- KNeighborsClassifier was imported from `sklearn.neighbors`.
- It classifies based on the majority class of the k nearest neighbors.
- Evaluated using the accuracy and confusion matrix.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, precision_score, recall_score, f1_score

# Step 1: Initialize and train the model
knn = KNeighborsClassifier(n_neighbors=5) # you can tune n_neighbors later
knn.fit(X_train_scaled, y_train)

# Step 2: Predict
y_pred_knn = knn.predict(X_test_scaled)
acc_knn = accuracy_score(y_test, y_pred_knn)
```



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

## Activity 1.3: Naïve Bayes Model

- GaussianNB was used from `sklearn.naive_bayes`.
- It assumes feature independence and is good for small datasets.
- The model was quick to train and gave decent accuracy.

```
#Naive Bayes
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report,

# 1. Initialize and train the model
nb = GaussianNB()
nb.fit(X_train_scaled, y_train)

# 2. Predict
y_pred_nb = nb.predict(X_test_scaled)
acc_nb = accuracy_score(y_test, y_pred_nb)
```

## Activity 1.4: Decision Tree Model

- DecisionTreeClassifier was used from `sklearn.tree`.
- It splits data based on feature thresholds.
- Very interpretable, but prone to overfitting.

```
#DECISION TREE
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report,

# 1. Initialize and train the model
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train_scaled, y_train)

# 2. Predict
y_pred_dt = dt.predict(X_test_scaled)
acc_dt = accuracy_score(y_test, y_pred_dt)
```

## Activity 1.5: Random Forest (Ensemble) Model

- RandomForestClassifier was used from `sklearn.ensemble`.
- Combines multiple decision trees to reduce overfitting.
- Improved performance over a stand-alone Decision Tree.

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

```
#Random Forest
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# 1. Initialize and train the model
rf = RandomForestClassifier(random_state=42, n_estimators=100)
rf.fit(X_train_scaled, y_train)

# 2. Predict
y_pred_rf = rf.predict(X_test_scaled)
acc_rf = accuracy_score(y_test, y_pred_rf)
```

## Activity 1.6: Gradient Boosting Model

- GradientBoostingClassifier was imported from sklearn.ensemble.
- It builds models sequentially and optimizes for loss.
- Gave high accuracy with longer training time.

```
#Gradient Boosting
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report,

# 1. Initialize and train the model
gb_model = GradientBoostingClassifier(random_state=42)
gb_model.fit(X_train_scaled, y_train)

# 2. Predict
y_pred_gb = gb_model.predict(X_test_scaled)
acc_gb = accuracy_score(y_test, y_pred_gb)
```

## Activity 1.7: AdaBoost Model

- AdaBoostClassifier was imported from sklearn.ensemble.
- It boosts weak learners (like Decision Trees) into a strong ensemble.
- Suitable for binary classification tasks like CKD.

```
#ADABOOST
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score, classification_report,

# 1. Initialize and train
ada_model = AdaBoostClassifier()
ada_model.fit(X_train, y_train)

# 2. Predict
y_pred = ada_model.predict(X_test)
acc_ada = accuracy_score(y_test, y_pred)
```

## Activity 1.8: Artificial Neural Network (ANN)

- ANN was built using MLPClassifier from sklearn.neural\_network.

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

- It's a feed-forward neural network with one or more hidden layers.
- Provided good accuracy and learned nonlinear patterns.

```
from sklearn.neural_network import MLPClassifier

# 1. Initialize and train
ann_model = MLPClassifier(hidden_layer_sizes=(100,), max_iter=500, random_state=42)
ann_model.fit(X_train_scaled, y_train)

# 2. Predict
y_pred = ann_model.predict(X_test_scaled)
acc_ann = accuracy_score(y_test, y_pred)
acc_svm = accuracy_score(y_test, y_pred)
```

## Activity 1.9: Support Vector Machine (SVM) Model

- SVC from `sklearn.svm` was used.
- It finds the optimal hyperplane to separate classes.
- This model gave the **highest accuracy** in our project.

```
#SVM
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# 1. Initialize and train
svm_model = SVC(kernel='rbf') # You can try 'linear', 'poly', 'sigmoid' too
svm_model.fit(X_train_scaled, y_train)

# 2. Predict
y_pred = svm_model.predict(X_test_scaled)
```

## Activity 2: Testing the Model

After training all classification models, we tested each one using the `.predict()` function from scikit-learn. This allowed us to evaluate how well each model performed on unseen test data.

## Milestone 5: Performance Testing & Hyperparameter Tuning

After training all models, we evaluated their performance using various **classification metrics**. This helped us understand how each model performs not just in terms of accuracy, but also in handling false positives and false negatives — which is crucial in medical predictions like CKD.

### *Activity 1.1: Compare the model*

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

we have 9 models lets compare each metrics here and visulaize it.

## 1.)Logistic Regression

Accuracy: 0.9875

```
Classification Report:
              precision    recall  f1-score   support

     0           1.00       0.98       0.99         50
     1           0.97       1.00       0.98         30

   accuracy          0.9875
  macro avg          0.9850
 weighted avg          0.9875
```

```
Confusion Matrix:
[[49  1]
 [ 0 30]]
Precision: 0.967741935483871
Recall: 1.0
F1 Score: 0.9836065573770492
```

## 2.)K-NN

KNN Classifier Results

Accuracy: 0.975

```
Classification Report:
              precision    recall  f1-score   support

     0           1.00       0.96       0.98         50
     1           0.94       1.00       0.97         30

   accuracy          0.9750
  macro avg          0.9700
 weighted avg          0.9700
```

```
Confusion Matrix:
[[48  2]
 [ 0 30]]
Precision: 0.9375
Recall: 1.0
F1 Score: 0.967741935483871
```

## 3.)Navie Bayes

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

## Naive Bayes Results

-----

Accuracy: 0.975

### Classification Report:

	precision	recall	f1-score	support
0	1.00	0.96	0.98	50
1	0.94	1.00	0.97	30
accuracy			0.97	80
macro avg	0.97	0.98	0.97	80
weighted avg	0.98	0.97	0.98	80

### Confusion Matrix:

```
[[48  2]
 [ 0 30]]
Precision: 0.9375
Recall: 1.0
F1 Score: 0.967741935483871
```

## 4.)Descision Tree

### Decision Tree Results

-----

Accuracy: 0.975

### Classification Report:

	precision	recall	f1-score	support
0	0.96	1.00	0.98	50
1	1.00	0.93	0.97	30
accuracy			0.97	80
macro avg	0.98	0.97	0.97	80
weighted avg	0.98	0.97	0.97	80

### Confusion Matrix:

```
[[50  0]
 [ 2 28]]
Precision: 1.0
Recall: 0.9333333333333333
F1 Score: 0.9655172413793104
```

## 5.)Random Forest

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

## Random Forest Results

Accuracy: 0.9625

### Classification Report:

	precision	recall	f1-score	support
0	0.94	1.00	0.97	50
1	1.00	0.90	0.95	30
accuracy			0.96	80
macro avg	0.97	0.95	0.96	80
weighted avg	0.96	0.96	0.96	80

### Confusion Matrix:

[[50 0]

[ 3 27]]

Precision: 1.0

Recall: 0.9

F1 Score: 0.9473684210526315

## 6.)Gradient Boosting

### Gradient Boosting Results

Accuracy: 0.9625

### Classification Report:

	precision	recall	f1-score	support
0	0.94	1.00	0.97	50
1	1.00	0.90	0.95	30
accuracy			0.96	80
macro avg	0.97	0.95	0.96	80
weighted avg	0.96	0.96	0.96	80

### Confusion Matrix:

[[50 0]

[ 3 27]]

Precision: 1.0

Recall: 0.9

F1 Score: 0.9473684210526315

## 7.)ADABoost

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

Accuracy: 0.9875

Classification Report:

	precision	recall	f1-score	support
0	0.98	1.00	0.99	50
1	1.00	0.97	0.98	30
accuracy			0.99	80
macro avg	0.99	0.98	0.99	80
weighted avg	0.99	0.99	0.99	80

Confusion Matrix:

```
[[50  0]
```

```
[ 1 29]]
```

Precision: 1.0

Recall: 0.9666666666666667

F1 Score: 0.9830508474576272

## 8.)ANN

Accuracy: 1.0

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	50
1	1.00	1.00	1.00	30
accuracy			1.00	80
macro avg	1.00	1.00	1.00	80
weighted avg	1.00	1.00	1.00	80

Confusion Matrix:

```
[[50  0]
```

```
[ 0 30]]
```

Precision: 1.0

Recall: 1.0

F1 Score: 1.0

## 9.)SVM

Accuracy: 1.0

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	50
1	1.00	1.00	1.00	30
accuracy			1.00	80
macro avg	1.00	1.00	1.00	80
weighted avg	1.00	1.00	1.00	80

Confusion Matrix:

```
[[50  0]
```

```
[ 0 30]]
```

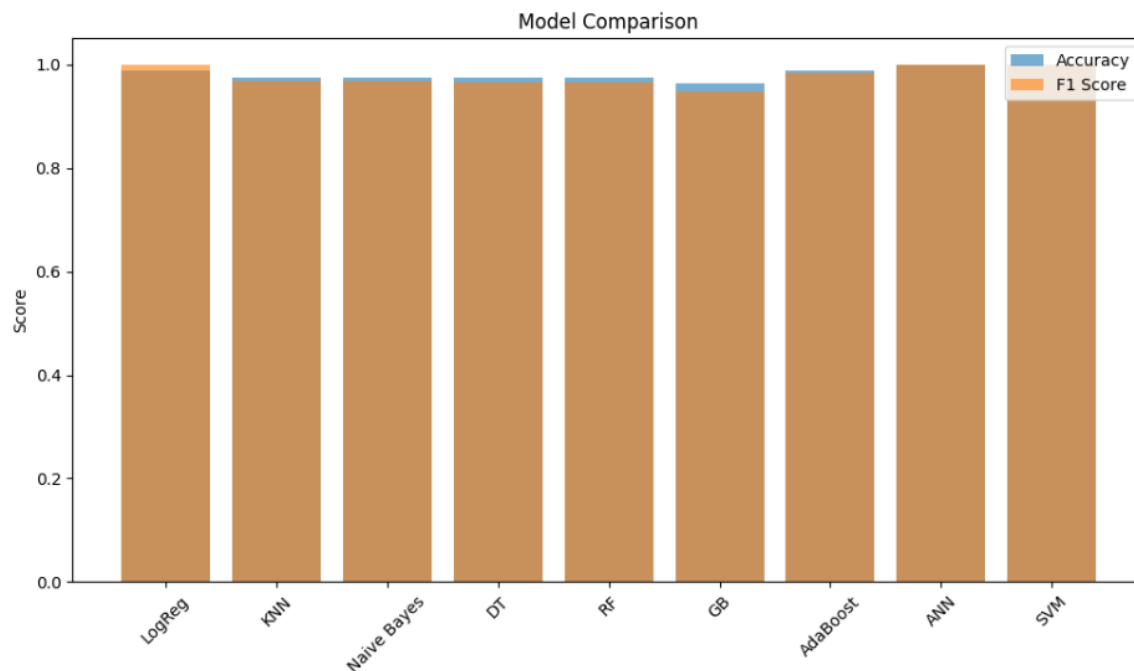
Precision: 1.0

Recall: 1.0

F1 Score: 1.0

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

Let's Compare each Model using graph



Since we got same metrics for ANN and SVM Lets do cross evaluation

## Cross-Validation to Compare Multiple Models

To ensure fair and robust evaluation of all the classification algorithms, we applied **5-fold cross-validation** on each model using the training data.

Cross-validation helps in:

- Reducing overfitting by testing the model on multiple data splits.
- Providing a better estimate of the model's real-world performance.

We used `cross_val_score` from `sklearn.model_selection` to evaluate all models under the same settings.

- Creates a **pipeline** for each model with **StandardScaler** + classifier.
- Performs **5-fold cross-validation** for each model on the training set.
- Stores and prints the **mean accuracy** for each model.
- Cross-validation provided more **reliable accuracy scores** for model comparison.
- This helped us **choose the best model** (e.g., SVM or ANN) based on average performance across multiple data splits.
- These scores were later used to finalize the best-performing algorithm for deployment.



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

```
# Define models in dictionary
models = {
    'Logistic Regression': LogisticRegression(),
    'KNN': KNeighborsClassifier(),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'Naive Bayes': GaussianNB(),
    'SVM (RBF)': SVC(kernel='rbf', C=1, gamma=0.1),
    'Gradient Boosting': GradientBoostingClassifier(),
    'AdaBoost': AdaBoostClassifier(),
    'ANN (MLP)': MLPClassifier(max_iter=1000)
}

# Loop and perform CV
cv_results = {}

for name, model in models.items():
    pipeline = make_pipeline(StandardScaler(), model)
    scores = cross_val_score(pipeline, X_train, y_train, cv=5, scoring='accuracy')
    mean_score = np.mean(scores)
    cv_results[name] = mean_score
    print(f"{name}: CV Mean Accuracy = {mean_score:.4f}")

Logistic Regression: CV Mean Accuracy = 0.9938
KNN: CV Mean Accuracy = 0.9656
Decision Tree: CV Mean Accuracy = 0.9750
Random Forest: CV Mean Accuracy = 0.9906
Naive Bayes: CV Mean Accuracy = 0.9625
SVM (RBF): CV Mean Accuracy = 0.9969
Gradient Boosting: CV Mean Accuracy = 0.9844
AdaBoost: CV Mean Accuracy = 0.9812
ANN (MLP): CV Mean Accuracy = 0.9938
```

From the above results, we observed that:

- **SVM (RBF)** achieved the highest cross-validation mean accuracy of **99.69%**
- Logistic Regression and ANN also performed well, with **99.38%**
- Random Forest and Gradient Boosting showed strong and consistent results too

## Hyperparameter Tuning using GridSearchCV

After selecting **SVM** as our final model based on cross-validation accuracy, we performed **hyperparameter tuning** to further optimize its performance.

Hyperparameter tuning is the process of finding the best set of model parameters that maximize performance. Unlike normal training parameters (which the model learns), **hyperparameters must be manually set** before training.

In SVM, important hyperparameters include:

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

- `C`: Controls the trade-off between achieving a low training error and a low testing error.
- `gamma`: Defines how far the influence of a single training example reaches.
- `kernel`: Determines the type of SVM used (e.g., 'rbf', 'linear').

```
# Create pipeline with best hyperparams
final_model = make_pipeline(
    StandardScaler(),
    SVC(C=1, gamma=0.1, kernel='rbf') # from your earlier tuning
)

# Train on full training set
final_model.fit(X_train, y_train)

# Predict on test set
y_pred = final_model.predict(X_test)
```

## Milestone 6: Model Deployment

Once we finalized the best-performing model (SVM with tuned parameters), the next step was to **deploy** it using a **Flask-based web application**. This made the model usable by non-technical users via a simple and interactive UI.

### Activity 1: Save the Best Model

After evaluating and tuning the SVM model, we saved it using the `joblib` library for future use without retraining.

```
import joblib

# Save the full pipeline with scaler + model
joblib.dump(final_model, 'svm_classifier_pipeline.pkl')
print("✅ Model saved as 'svm_classifier_pipeline.pkl'")
```

✅ Model saved as 'svm\_classifier\_pipeline.pkl'

### Activity 2: Integrate with Web Framework

We built a **web application using Flask** to interact with our model. This includes building both the frontend (UI) and backend (server-side script).

#### Activity 2.1: Building HTML Page

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

We created the following HTML files and saved them in a folder named `templates/`:

- `index.html` – The homepage for the app
- `form.html` – A detailed input form with 24 CKD-related input fields
- `result.html` – Displays the prediction result to the user

These HTML pages were styled using custom CSS and images placed in a `static/` folder.

## Activity 2.2: Build Python Script with Flask

We created `app.py` which includes:

### Import Libraries & Load Model

```
1 from flask import Flask, render_template, request
2 import numpy as np
3 import pandas as pd
4 import joblib # or use pickle
5
```

### Route to Home Page

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

### Route to Form Page

```
@app.route('/form')
def show_form():
    return render_template('form.html')
```

### Route to Prediction

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

```
def predict():
    if request.method == 'POST':
        try:
            # Collect inputs in the same order as training
            data = [
                float(request.form['age']),
                float(request.form['bp']),
                float(request.form['sg']),
                float(request.form['al']),
                float(request.form['su']),
                int(request.form['rbc']),
                int(request.form['pc']),
                int(request.form['pcc']),
                int(request.form['ba']),
                float(request.form['bgr']),
                float(request.form['bu']),
                float(request.form['sc']),
                float(request.form['sod']),
                float(request.form['pot']),
                float(request.form['hemo']),
                float(request.form['pcv']),
                float(request.form['wc']),
                float(request.form['rc']),
                int(request.form['htn']),
                int(request.form['dm']),
                int(request.form['cad']),
                int(request.form['appet']),
                int(request.form['pe']),
                int(request.form['ane'])
            ]

            # Column names matching the training dataset
            columns = ['age', 'bp', 'sg', 'al', 'su', 'rbc', 'pc', 'pcc', 'ba', 'bgr',
                       'bu', 'sc', 'sod', 'pot', 'hemo', 'pcv', 'wc', 'rc',
                       'htn', 'dm', 'cad', 'appet', 'pe', 'ane']

            # Create DataFrame
            df_input = pd.DataFrame([data], columns=columns)

            # DEBUG: Print the final input to the model
            print("✅ FINAL INPUT TO MODEL:")
            print(df_input)

            # Get prediction and probability
            prediction = model.predict(df_input)[0]

            # If your model supports predict_proba, print probabilities
            if hasattr(model, 'predict_proba'):
                proba = model.predict_proba(df_input)[0]
                print(f"🔍 Prediction: {prediction}, Probabilities: {proba}")
            else:
                print(f"🔍 Prediction: {prediction}")

            return render_template('result.html', prediction=prediction)

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

- The `predict()` function **retrieves all the values** entered in the HTML form using `request.form[]`.
- These values are **stored in a list**, reshaped as a DataFrame, and **passed to the `model.predict()` function**.
- The model returns a prediction (0 for CKD, 1 for Not CKD).
- The prediction is then rendered and **displayed on the result page** (`result.html`).

This allows for real-time prediction based on user inputs.

## Main Function

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

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

## Activity 2.3: Run the Web Application

Once the Flask application was fully developed and integrated with the trained CKD prediction model, we proceeded to run and test the web application locally.

### Steps to Run the Application:

1. Open **Anaconda Prompt** from the Start menu (or use any terminal/command prompt).

2. Navigate to the project folder where the `app.py` file is located. For example:

```
(base) C:\Users\yvenk>cd "C:\Users\yvenk\Documents\CKD_Deployment"
```

3. Run the **Flask app** using the following command:

```
(base) C:\Users\yvenk\Documents\CKD_Deployment>python app.py  
* 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 watchdog (windowsapi)  
* Debugger is active!  
* Debugger PIN: 327-619-767
```

4. Open your browser and enter these

`http://127.0.0.1:5000`

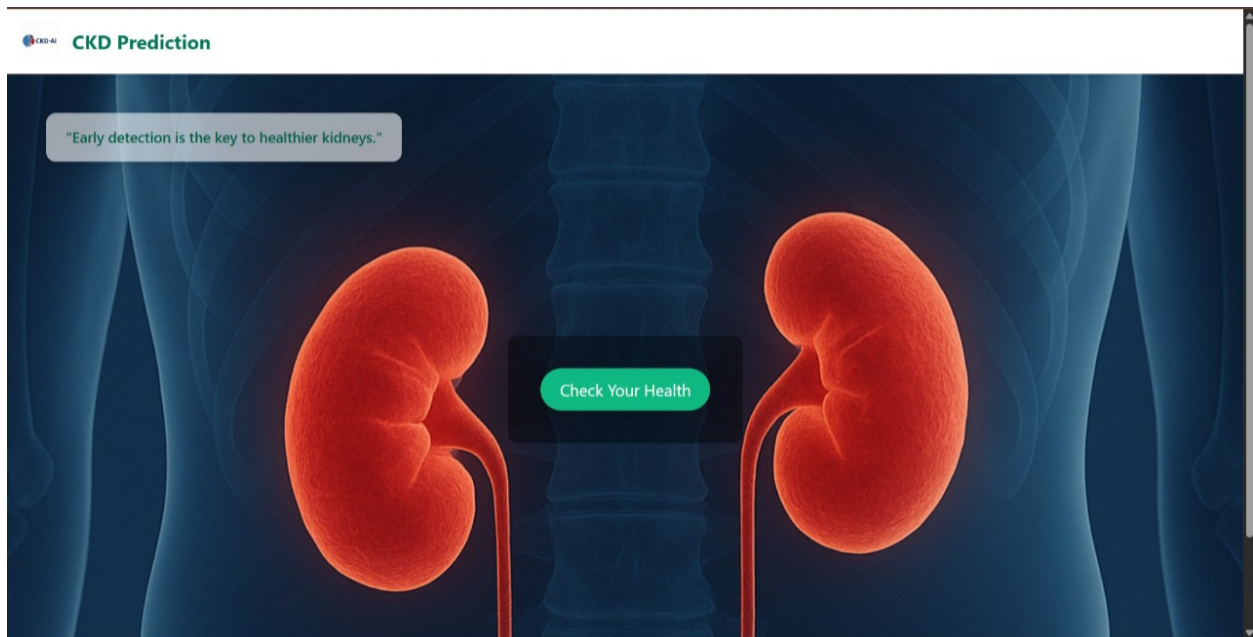
5. On the homepage:

- Click the **"Check Your Health"** button (or your form navigation).
- Enter all the required CKD-related inputs.
- Click the **"Submit"** button.


6. The prediction result (CKD or Not CKD) will be **displayed on the results page** (`result.html`) based on your inputs.

HOME PAGE:

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



## Example For Healthy Form



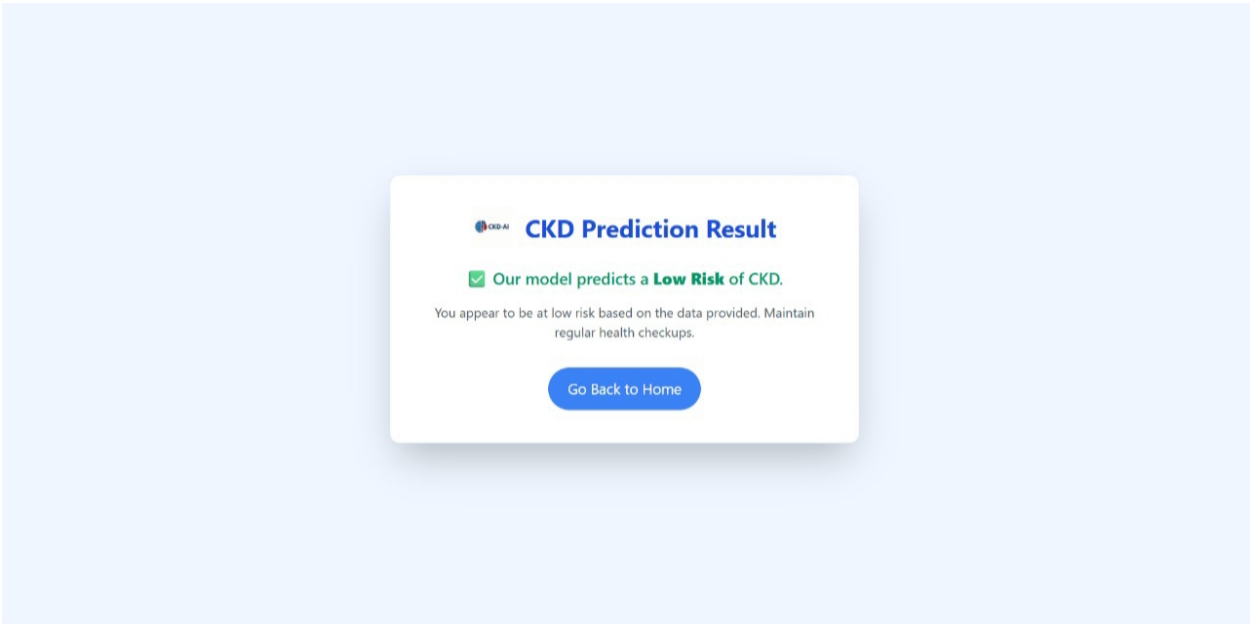
 **CKD Health Information Form**

Age45	Blood Pressure80	Specific Gravity1.020
Albumin1	Sugar0	RBC (0=Normal, 1=Abnormal) 0
PC (0=Normal, 1=Abnormal) 0	PCC (0=Present, 1=Not Present) 1	BA (0=Present, 1=Not Present) 1
Blood Glucose140	Blood Urea30	Serum Creatinine1.2
Sodium135	Potassium4.2	Hemoglobin13.5
PCV40	WBC8200	RBC Count4.8
Hypertension (0=Yes, 1=No) 1	Diabetes (0=Yes, 1=No) 1	CAD (0=Yes, 1=No) 1
Appetite (0=Good, 1=Poor) 0	Pedal Edema (0=Yes, 1=No) 1	Anemia (0=Yes, 1=No) 1



Results:

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



Example Form For CKD:



 **CKD Health Information Form**

Age65

Albumin3

PC (0=Normal, 1=Abnormal) 1

Blood Glucose120

Sodium135

PCV30

Hypertension (0=Yes, 1=No) 0

Appetite (0=Good, 1=Poor) 1

Blood Pressure70

Sugar2

PCC (0=Present, 1=Not Present) 0

Blood Urea60

Potassium5

WBC11000

Diabetes (0=Yes, 1=No) 0

Pedal Edema (0=Yes, 1=No) 0

Specific Gravity1.010

RBC (0=Normal, 1=Abnormal) 1

BA (0=Present, 1=Not Present) 0

Serum Creatinine2.6

Hemoglobin9.5

RBC Count3

CAD (0=Yes, 1=No) 0

Anemia (0=Yes, 1=No) 0

 Predict CKD Risk

Results:

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

---



This is a **simple and user-friendly web application** designed to predict **Chronic Kidney Disease (CKD)** based on 24 medical input values.

The interface is clean and easy to use — users just enter their health details, click submit, and get an instant prediction.

The backend uses a highly accurate **SVM machine learning model**, trained and optimized to give reliable results.

This app shows how AI can be used in healthcare to assist early detection, making it both **accessible and impactful** for everyone.

---



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