Project documentation

Early Prediction Of Chronic Kidney Disease Using Machine Learning

Members:ASNEEM ATHAR SHAIK, MAHESH THOTAKURA, SUVARCHALA JANGAM, SHRADHA MUNDADA

1.Introduction

- **Project overviews**

Machine learning helps predict Chronic Kidney Disease (CKD) early by analyzing patient data, enabling timely intervention and personalized treatment.

1.1. Objectives

To develop an integrated website that utilize machine learning for the early prediction of Chronic Kidney Disease (CKD), enabling users to input health data and receive accurate risk assessments and actionable insights for early intervention and better health management.

2. Project Initialization and Planning Phase

2.1. Define Problem Statement

This project aims to develop a website that uses machine learning to predict Chronic Kidney Disease (CKD) early. By allowing users to input health data and receive accurate risk assessments, the platform will support early intervention and provide educational resources while ensuring data privacy.

2.2. Project Proposal (Proposed Solution)

Utilize machine learning algorithms to analyze key health indicators for early prediction of Chronic Kidney Disease (CKD), offering accurate risk assessments, educational resources on prevention, and an intuitive user interface while ensuring robust data privacy.

2.3. Initial Project Planning:

The plan outlines 3 phases:

1. **Development:** Create the machine learning models and build the website infrastructure, including dataset preparation, library imports, and implementation of data processing steps like handling missing values and encoding.
2. **Implementation:** Deploy the website with integrated machine learning models, ensuring functionality for user inputs, risk assessment, and data privacy.
3. **Evaluation:** Test the website and models for accuracy, gather user feedback, and make improvements to enhance performance, usability, and overall effectiveness.

3.Data Collection and Preprocessing Phase

3.1. Data Collection Plan and Raw Data Sources Identified

The data collection plan aims to gather comprehensive health-related data necessary for training and testing machine learning models to predict Chronic Kidney Disease (CKD). This involves focusing on key health metrics such as age, blood pressure, blood sugar levels, BMI, and family history. The primary data source for this project is a dataset obtained from this Google Drive link. This dataset will be used to develop and validate the models, ensuring they are capable of accurate CKD risk assessment. Additional data may be collected or utilized as needed to enhance the model's performance and reliability.

Importing the libraries

Import the necessary libraries as shown in the image.

IMPORT LIBRARIES

```
import pandas as pd
import numpy as np
#from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler, LabelEncoder

import sklearn.preprocessing as preprocessing

from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno
from sklearn.metrics import accuracy_score,confusion_matrix

from sklearn.preprocessing import StandardScaler, LabelEncoder

import pickle
```

3.2. Data Quality Report

The Data Quality Report evaluates the dataset used for the Chronic Kidney Disease (CKD) prediction project. It focuses on completeness by identifying any missing values and their potential impact on model accuracy. Consistency checks are performed to ensure there are no duplicate entries or formatting issues. Accuracy is assessed by verifying the data against reliable sources to ensure correctness. Relevance is ensured by confirming that the metrics included—such as age, blood pressure, and blood sugar levels—are pertinent to CKD prediction. Timeliness is considered to make sure the data is current and reflective of recent health trends. Addressing these factors ensures the dataset supports reliable and accurate CKD risk assessments.

READ THE DATASET

```
data=pd.read_csv("/content/chronickidneydisease.csv")
```

Understanding Data Type And Summary Of Features

```
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 × 26 columns

```
data.tail()
```

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
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	notckd
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	notckd
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	notckd
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	notckd
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	notckd

5 rows × 26 columns

```
data.info()
```

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

This gives us the information about the Dataset.

3.3. Data Exploration and Preprocessing

Data exploration involves analyzing the dataset to understand its structure and identify patterns or anomalies. Preprocessing includes handling missing values, encoding categorical variables, and normalizing features. The dataset is then split into independent and dependent variables, and further divided into training and testing sets to ensure the data is clean and suitable for building effective machine learning models.

- **Handling missing values :**

```
data.isnull().sum()
```

```
age                             9
blood_pressure                 12
specified_gravity              47
albumin                        46
sugar                          49
red_blood_cells               152
pus_cell                       65
pus_cell_clumps                 4
bacteria                        4
blood_glucose_random           44
blood_urea                     19
serum_creatinine               17
sodium                         87
potassium                      88
hemoglobin                     52
packed_cell_volume             70
white_blood_cell_count        105
red_blood_cell_count          130
hypertension                    2
diabetesmellitus                2
coronary_artery disease         2
appetite                        1
pedal_edema                     1
anemia                          1
class                           0
dtype: int64
```

- **Replacing the Missing Values**

```
data['blood_glucose_random'].fillna(data['blood_glucose_random'].mean(),inplace=True)
data['blood_pressure'].fillna(data['blood_pressure'].mean(),inplace=True)
data['blood_urea'].fillna(data['blood_urea'].mean(),inplace=True)
data['packed_cell_volume'].fillna(data['packed_cell_volume'].mean(),inplace=True)
data['potassium'].fillna(data['potassium'].mean(),inplace=True)
data['red_blood_cell_count'].fillna(data['red_blood_cell_count'].mean(),inplace=True)
data['serum_creatinine'].fillna(data['serum_creatinine'].mean(),inplace=True)
data['sodium'].fillna(data['sodium'].mean(),inplace=True)
data['white_blood_cell_count'].fillna(data['white_blood_cell_count'].mean(),inplace=True)
data['age'].fillna(data['age'].mode()[0],inplace=True)
data['hypertension'].fillna(data['hypertension'].mode()[0],inplace=True)
data['pus_cell_clumps'].fillna(data['pus_cell_clumps'].mode()[0],inplace=True)
data['appetite'].fillna(data['appetite'].mode()[0],inplace=True)
data['albumin'].fillna(data['albumin'].mode()[0],inplace=True)
data['pus_cell'].fillna(data['pus_cell'].mode()[0],inplace=True)
data['red_blood_cells'].fillna(data['red_blood_cells'].mode()[0],inplace=True)
data['coronary_artery disease'].fillna(data['coronary_artery disease'].mode()[0],inplace=True)
data['bacteria'].fillna(data['bacteria'].mode()[0],inplace=True)
data['anemia'].fillna(data['anemia'].mode()[0],inplace=True)
data['sugar'].fillna(data['sugar'].mode()[0],inplace=True)
data['diabetesmellitus'].fillna(data['diabetesmellitus'].mode()[0],inplace=True)
data['pedal_edema'].fillna(data['pedal_edema'].mode()[0],inplace=True)
data['specified_gravity'].fillna(data['specified_gravity'].mode()[0],inplace=True)
```

- **Removing and changing columns**

```
[5]
#REMOVE UNNESSARY COLUMN
data.drop(['id'],axis=1,inplace=True)
```
Python

```
[6]
data.columns
```
Python

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

```
[7]
#Changing the column names
data.columns=['age','blood_pressure','specified_gravity','albumin','sugar','red_blood_cells','pus_cell','pus_cell_clumps','bacteria','blood_glucose_
data.columns
```
Python

```
Index(['age', 'blood_pressure', 'specified_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')
```

- **Label Encoding**

```
for i in catcols:
  print("label encoding of:",i)
  LEi=LabelEncoder()
  print(data[i])
  data[i]=LEi.fit_transform(data[i])
  print(data[i])
  print('*'*100)
```

Python

```
label encoding of: specified_gravity
0      1.020
1      1.020
2      1.010
3      1.005
4      1.010
       ...
395    1.020
396    1.025
397    1.020
398    1.025
399    1.025
Name: specified_gravity, Length: 400, dtype: float64
0      3
1      3
2      1
3      0
4      1
      ..
395    3
396    4
397    3
398    4
399    4
Name: specified_gravity, Length: 400, dtype: int64
...
398    1
399    1
Name: class, Length: 400, dtype: int64
**********************************************************************************************
```

- **Splitting the Dataset into Dependent And independent Variable**

Splitting The Dataset Into Dependent And Independent Variable

```
#creating independent and dependent variables
sel=['age','red_blood_cells','pus_cell','blood_glucose_random','blood_urea','pedal_edema','anemia',
     'diabetesmellitus','coronary_artery disease','blood_pressure']
x=pd.DataFrame(data,columns=sel)
y=pd.DataFrame(data,columns=['class'])
print(x.shape)
print(y.shape)
```

28]

```
(400, 10)
(400, 1)
```

4.Model Development Phase

4.1. Feature Selection Report

The Feature Selection Report focuses on identifying the most relevant features from the dataset for building the machine learning model. Unnecessary columns are removed, and categorical values are encoded into numeric format. This process ensures that only the essential and informative features are retained and appropriately formatted for effective model training.

Let us see the total discrimination about the data that we have preprocessed.

```
data.describe()
```

	age	blood_pressure	specified_gravity	albumin	sugar	red_blood_cells	pus_cell	pus_cell_clumps	bacteria	blood_glucose_random	...	packed
count	400.000000	400.000000	400.000000	400.00000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	...	
mean	51.675000	76.469072	2.542500	0.90000	0.395000	0.882500	0.810000	0.105000	0.055000	148.036517	...	
std	17.022008	13.476298	1.086806	1.31313	1.040038	0.322418	0.392792	0.306937	0.228266	74.782634	...	
min	2.000000	50.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	22.000000	...	
25%	42.000000	70.000000	2.000000	0.00000	0.000000	1.000000	1.000000	0.000000	0.000000	101.000000	...	
50%	55.000000	78.234536	3.000000	0.00000	0.000000	1.000000	1.000000	0.000000	0.000000	126.000000	...	
75%	64.000000	80.000000	3.000000	2.00000	0.000000	1.000000	1.000000	0.000000	0.000000	150.000000	...	
max	90.000000	180.000000	4.000000	5.00000	5.000000	1.000000	1.000000	1.000000	1.000000	490.000000	...	

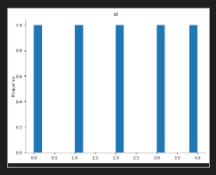
8 rows × 25 columns

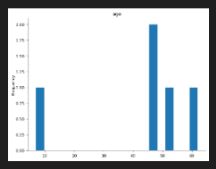
```
data.head()
```

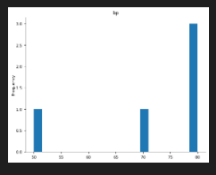
	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	...	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	...	38	6000	NaN	no	no	no	good	no	no	ckd
2	2	62.0	80.0	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

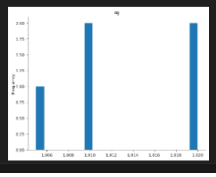
5 rows × 26 columns

Distributions

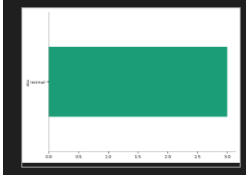


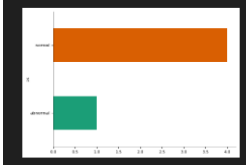


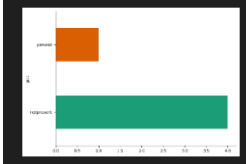


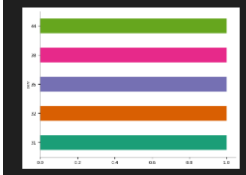


Categorical distributions

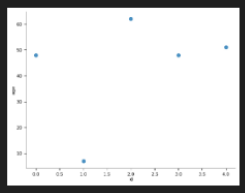


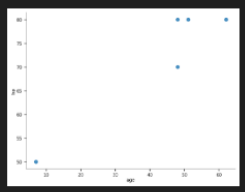


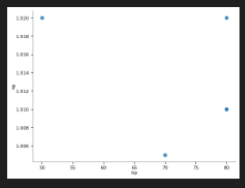


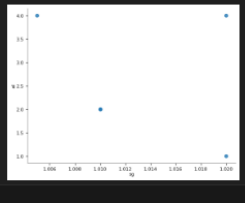


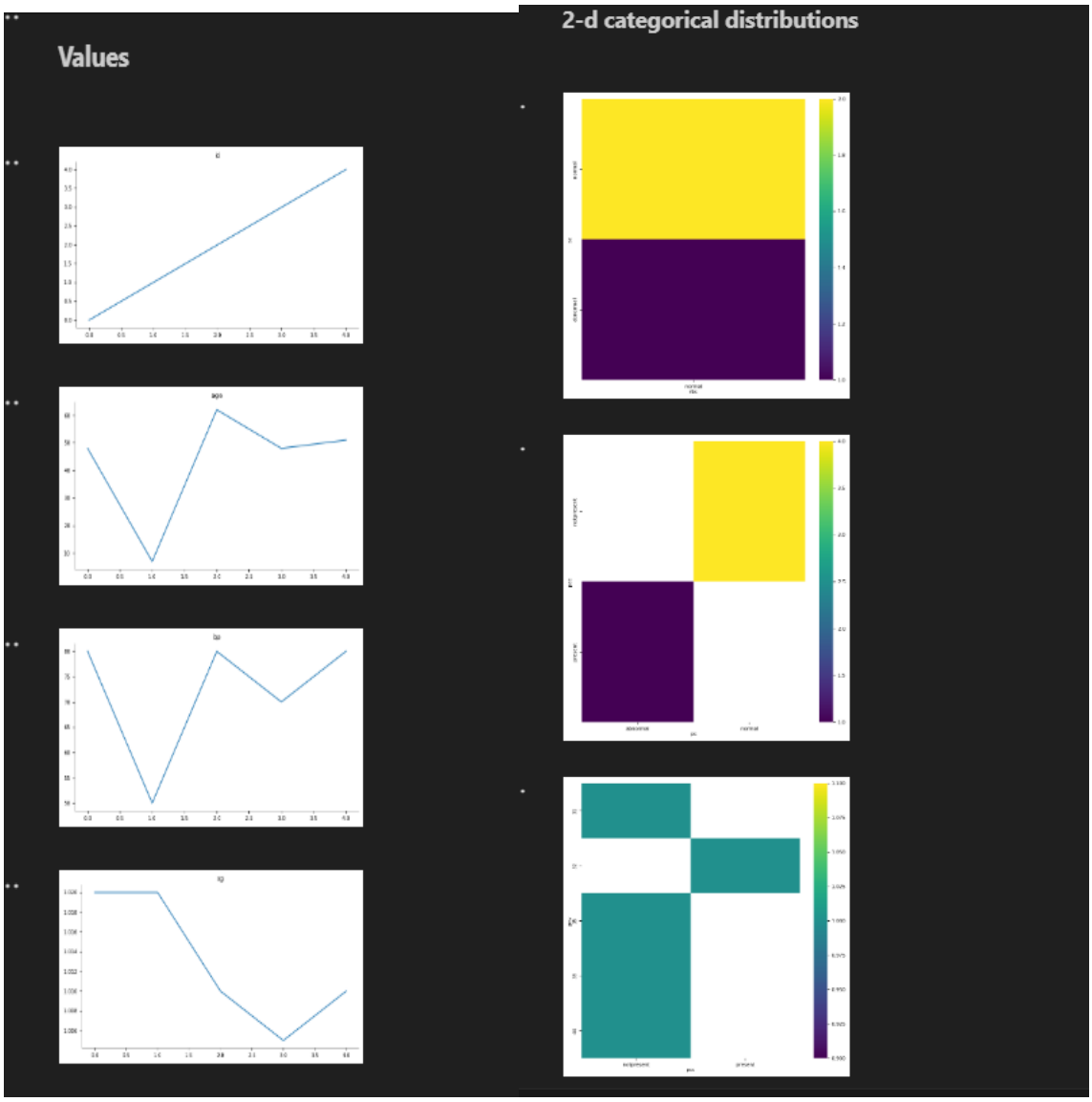
2-d distributions











Splitting the Data: Preparing for Learning and Assessment

The preprocessed data serves as the fuel for our model. However, we don't simply throw it all at the model at once. A strategic data split is crucial:

Training Set : This serves as the basis for the learning process of the model. The features (such as order weight, distance) and their correlation with the goal variable (such as on-time vs. late

delivery, expected delivery time) are presented to the model. The model gains the ability to recognize links and patterns via this exposure, which will help it forecast data that has not yet been observed.

Testing Set : The last test for the generalizability of the model is conducted on this untested set. Select metrics are used to evaluate the model on the testing set after it has been trained on the training data and maybe fine-tuned using the validation set. This gives an objective evaluation of the model's performance on data that it has never seen before.

Crucially important is the size of each split (training, validation, and testing). Typically, 60–80% of the data are set aside for training, 10–20% for validation, and 10–20% for testing. Depending on the size and features of the dataset, the precise allocation can be changed.

- **Data Splitting**

Model Selection Report:

The Model Selection Report evaluates several machine learning models to determine the most effective one for predicting Chronic Kidney Disease (CKD). The models considered include:

- Logistic Regression
- Random Forest Classifier
- AdaBoost Classifier
- K-Nearest Neighbors (KNN)
- Decision Tree Classifier
- XGBoost
- Gradient Boosting Classifier

The report aims to identify the model that provides the highest accuracy and reliability in assessing CKD risk.

The following are the specific codes used for training the various machine learning models:

- **Logistic Regression**

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

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, classification_report
y_pred = mo.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1-Score: {f1}')
print(f'ROC-AUC: {roc_auc}')
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.9375
Precision: 0.8387096774193549
Recall: 1.0
F1-Score: 0.9122807017543859
ROC-AUC: 0.9537037037037037
              precision    recall  f1-score   support

           0       1.00      0.91      0.95        54
           1       0.84      1.00      0.91        26

    accuracy                           0.94        80
   macro avg       0.92      0.95      0.93        80
weighted avg       0.95      0.94      0.94        80
```

- **Random Forest Classifier**

```
from sklearn.ensemble import RandomForestClassifier
model1=RandomForestClassifier()
model1.fit(x_train,y_train)
```

```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, classification_report
y_pred = model1.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1-Score: {f1}')
print(f'ROC-AUC: {roc_auc}')
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.975
Precision: 0.9615384615384616
Recall: 0.9615384615384616
F1-Score: 0.9615384615384616
ROC-AUC: 0.9715099715099716
              precision    recall  f1-score   support

           0       0.98      0.98      0.98        54
           1       0.96      0.96      0.96        26

    accuracy                           0.97        80
   macro avg       0.97      0.97      0.97        80
weighted avg       0.97      0.97      0.97        80
```

- **AdaBoost Classifier**

ADA BOOST

```
from sklearn.ensemble import AdaBoostClassifier,GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.tree import DecisionTreeClassifier

ada=AdaBoostClassifier()

ada.fit(x_train,y_train)
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, classification_report
y_pred = ada.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1-Score: {f1}')
print(f'ROC-AUC: {roc_auc}')
print(classification_report(y_test, y_pred))

# Feature importance
feature_importances = pd.DataFrame(ada.feature_importances_, index=x.columns, columns=['importance']).sort_values('importance', ascending=False)
print(feature_importances)
```

- **KNN**

```
from sklearn.neighbors import KNeighborsClassifier
#intialize the KNN classifier
knn=KNeighborsClassifier()
#train the model
knn.fit(x_train,y_train)
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, classification_report
y_pred = knn.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1-Score: {f1}')
print(f'ROC-AUC: {roc_auc}')
print(classification_report(y_test, y_pred))
```

- **Decision Tree Classifier**

```
from sklearn.tree import DecisionTreeClassifier
model2=DecisionTreeClassifier()
model2.fit(x_train,y_train)
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, classification_report
y_pred = model2.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1-Score: {f1}')
print(f'ROC-AUC: {roc_auc}')
print(classification_report(y_test, y_pred))
```

- **XGBoost**

```
import xgboost as xgb
xg=xgb.XGBClassifier()
xg.fit(x_train,y_train)
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, classification_report
y_pred = xg.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1-Score: {f1}')
print(f'ROC-AUC: {roc_auc}')
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.95
Precision: 0.9230769230769231
Recall: 0.9230769230769231
F1-Score: 0.9230769230769231
ROC-AUC: 0.9430199430199432
              precision    recall  f1-score   support

           0       0.96      0.96      0.96        54
           1       0.92      0.92      0.92        26

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

- **AdaBoost**

```python
from sklearn.ensemble import AdaBoostClassifier,GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.tree import DecisionTreeClassifier

ada=AdaBoostClassifier()

ada.fit(x_train,y_train)
```

```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, classification_report
y_pred = ada.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1-Score: {f1}')
print(f'ROC-AUC: {roc_auc}')
print(classification_report(y_test, y_pred))

# Feature importance
feature_importances = pd.DataFrame(ada.feature_importances_, index=x.columns, columns=['importance']).sort_values('importance', ascending=False)
print(feature_importances)
```

```
Accuracy: 0.9875
Precision: 0.9629629629629629
Recall: 1.0
F1-Score: 0.9811320754716981
ROC-AUC: 0.9907407407407408
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        54
           1       0.96      1.00      0.98        26

    accuracy                           0.99        80
   macro avg       0.98      0.99      0.99        80
weighted avg       0.99      0.99      0.99        80

                        importance
blood_urea                    0.34
age                           0.22
blood_glucose_random          0.20
blood_pressure                0.12
red_blood_cells               0.02
pus_cell                      0.02
pedal_edema                   0.02
anemia                        0.02
diabetesmellitus              0.02
coronary_artery disease       0.02
```

- **Gradient Boosting Classifier**

```python
from sklearn.ensemble import AdaBoostClassifier,GradientBoostingClassifier
gra=GradientBoostingClassifier()
gra.fit(x_train,y_train)
```

Python

```
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_gb.py:437: DataConversionWarning: A column-vector y was passed when a 1d array was expected.
 y = column_or_1d(y, warn=True)
```

```
▾ GradientBoostingClassifier
GradientBoostingClassifier()
```

```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, classification_report
y_pred = gra.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1-Score: {f1}')
print(f'ROC-AUC: {roc_auc}')
print(classification_report(y_test, y_pred))
```

Python

```
Accuracy: 0.9625
Precision: 0.96
Recall: 0.9230769230769231
F1-Score: 0.9411764705882353
ROC-AUC: 0.9522792022792023
              precision    recall  f1-score   support

           0       0.96      0.98      0.97        54
           1       0.96      0.92      0.94        26

    accuracy                           0.96        80
   macro avg       0.96      0.95      0.96        80
weighted avg       0.96      0.96      0.96        80
```

4.3. Initial Model Training Code, Model Validation and Evaluation Report.

```python
models = {
    'Logistic Regression': mo,
    'KNN': knn,
    'XGBoost': xg,
    'Gradient Boosting': gra,
    'AdaBoost': ada,
    'Decision Tree': model2,
    'Random Forest': model1,

}

# Function to evaluate a model
def evaluate_model(modelss, x_train, y_train, x_test, y_test):
    modelss.fit(x_train, y_train)
    y_pred = modelss.predict(x_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')
    return accuracy, precision, recall, f1

# Evaluate each model and store the results
results = []
for name, modelss in models.items():
    accuracy, precision, recall, f1 = evaluate_model(modelss, x_train, y_train, x_test, y_test)
    results.append({
        'Model': name,
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1 Score': f1
    })
```

```python
# Convert results to a DataFrame
results_df = pd.DataFrame(results)

# Set up the matplotlib figure
plt.figure(figsize=(10, 6))

# Create a barplot
metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score']
for metric in metrics:
    sns.barplot(x='Model', y=metric, data=results_df, label=metric)

# Add legend, title, and labels
plt.title('Model Performance Comparison')
plt.ylabel('Score')
plt.legend(loc='upper left')

# Show the plot
plt.show()
print(results_df)
```

Evaluation report:

	Model	Accuracy	Precision	Recall	F1 Score
0	Logistic Regression	0.9375	0.947581	0.9375	0.938724
1	KNN	0.8875	0.906850	0.8875	0.890275
2	XGBoost	0.9500	0.950000	0.9500	0.950000
3	Gradient Boosting	0.9625	0.962455	0.9625	0.962304
4	AdaBoost	0.9875	0.987963	0.9875	0.987560
5	Decision Tree	0.9500	0.950000	0.9500	0.950000
6	Random Forest	0.9750	0.975000	0.9750	0.975000

By, the above analysis , We selected **AdaBoost classifier** as our final model

S

5.Model Optimization and Tuning Phase

5.1. Hyperparameter Tuning Documentation

Hyperparameter tuning is a critical step in the machine learning workflow that involves adjusting the settings of your model to achieve the best possible performance. It's like fine-tuning the dials on a radio to get the clearest signal. In this documentation, we'll delve into the world of hyperparameter tuning, explaining its importance, common techniques, and best practices.

Model Optimisation

Decision Tree Classifier:

```
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.95
Precision: 0.9230769230769231
Recall: 0.9230769230769231
F1-Score: 0.9230769230769231
ROC-AUC: 0.9430199430199432
              precision    recall  f1-score   support

           0       0.96      0.96      0.96        54
           1       0.92      0.92      0.92        26

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

```
confusion_matrix(y_test,y_pred)
```

```
array([[53,  1],
       [ 1, 25]], dtype=int64)
```

Gradient Boosting Classifier:

```
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.9625
Precision: 0.96
Recall: 0.9230769230769231
F1-Score: 0.9411764705882353
ROC-AUC: 0.9522792022792023
              precision    recall  f1-score   support

           0       0.96      0.98      0.97        54
           1       0.96      0.92      0.94        26

    accuracy                           0.96        80
   macro avg       0.96      0.95      0.96        80
weighted avg       0.96      0.96      0.96        80
```

```
confusion_matrix(y_test,y_pred)
```

```
array([[53,  1],
       [ 2, 24]], dtype=int64)
```

XG Boost Classifier:

```
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.95
Precision: 0.9230769230769231
Recall: 0.9230769230769231
F1-Score: 0.9230769230769231
ROC-AUC: 0.9430199430199432
              precision    recall  f1-score   support

           0       0.96      0.96      0.96        54
           1       0.92      0.92      0.92        26

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

```
confusion_matrix(y_test,y_pred)
```

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

KNN:

```
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.8875
Precision: 0.7575757575757576
Recall: 0.9615384615384616
F1-Score: 0.847457627118644
ROC-AUC: 0.9066951566951568
              precision    recall  f1-score   support

           0       0.98      0.85      0.91        54
           1       0.76      0.96      0.85        26

    accuracy                           0.89        80
   macro avg       0.87      0.91      0.88        80
weighted avg       0.91      0.89      0.89        80
```

```
confusion_matrix(y_test,y_pred)
```

```
array([[53,  1],
       [ 1, 25]], dtype=int64)
```

Random Forest Classifier:

```
    print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.975
Precision: 0.9615384615384616
Recall: 0.9615384615384616
F1-Score: 0.9615384615384616
ROC-AUC: 0.9715099715099716
              precision    recall  f1-score   support

           0       0.98      0.98      0.98        54
           1       0.96      0.96      0.96        26

    accuracy                           0.97        80
   macro avg       0.97      0.97      0.97        80
weighted avg       0.97      0.97      0.97        80
```

```
    confusion_matrix(y_test,y_pred)
```

```
array([[53,  1],
       [ 1, 25]], dtype=int64)
```

Logistic Regression:

```
    confusion_matrix(y_test,y_pred)
```

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

```
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.91	0.95	54
1	0.84	1.00	0.91	26
accuracy			0.94	80
macro avg	0.92	0.95	0.93	80
weighted avg	0.95	0.94	0.94	80

Hyperparameter Tuning

KNN:

```
knn = KNeighborsClassifier(
    n_neighbors=5,
    weights='uniform',
    algorithm='auto',
    leaf_size=30,
    p=2,
    metric='minkowski',
    n_jobs=None
)
```

```
# Defining the parameter grid for tuning
param_grid = {
    'n_neighbors': [3, 5, 7, 9, 11],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree','kd_tree','brute'],
    'leaf_size': [20, 30, 40, 50],
    'p': [1, 2]
}
```

Logistic Regression:

```
mo = LogisticRegression(
    penalty='l2',
    C=1.0,
    solver='lbfgs',
    max_iter=100,
    random_state=42
)
```

```
# Defining the parameter grid for tuning
param_grid = {
    'penalty': ['l1', 'l2', 'elasticnet', 'none'],
    'C': [0.01, 0.1, 1.0, 10, 100],
    'solver': ['lbfgs', 'liblinear', 'saga'],
    'max_iter': [100, 200, 300]
}
```

XGBoostClassifier:

```
xg = xgb.XGBClassifier(
    objective='binary:logistic',
    learning_rate=0.1,
    n_estimators=100,
    max_depth=3,
    min_child_weight=1,
    subsample=1.0,
    colsample_bytree=1.0,
    random_state=42
)
```

```
# Defining the parameter grid for tuning
param_grid = {
    'learning_rate': [0.01, 0.1, 0.2],
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 5, 7],
    'min_child_weight': [1, 3, 5],
    'subsample': [0.8, 0.9, 1.0],
    'colsample_bytree': [0.8, 0.9, 1.0],
    'gamma': [0, 0.1, 0.2],
    'reg_alpha': [0, 0.01, 0.1],
    'reg_lambda': [1, 1.5, 2]
}
```

GradientBoostingClassifier:

```
gra = GradientBoostingClassifier(
    loss='deviance',           # Los
    learning_rate=0.1,         # Lea
    n_estimators=100,          # Num
    subsample=1.0,             # Fra
    criterion='friedman_mse', # Fun
    min_samples_split=2,       # Min
    min_samples_leaf=1,        # Min
    max_depth=3,               # Max
    random_state=42            # Ran
)
```

```
# Defining the parameter grid for tuning
param_grid = {
    'loss': ['deviance', 'exponential'],
    'learning_rate': [0.01, 0.1, 0.2],
    'n_estimators': [100, 200, 300],
    'subsample': [0.8, 0.9, 1.0],
    'criterion': ['friedman_mse', 'mse', 'mae'],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_depth': [3, 5, 7],
    'max_features': ['sqrt', 'log2', None]
}
```

Decision Tree Classifier:

```
model2 = DecisionTreeClassifier(
    criterion='gini',           #
    splitter='best',            #
    max_depth=None,             #
    min_samples_split=2,        #
    min_samples_leaf=1,         #
    max_features=None,          #
    random_state=42             #
)
```

```
# Defining the parameter grid for tuning
param_grid = {
    'criterion': ['gini', 'entropy'],
    'splitter': ['best', 'random'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': [None, 'sqrt', 'log2'],
}
```

Random Forest Classifier:

```
model1 = RandomForestClassifier(
    n_estimators=100,       # Nu
    max_depth=None,         # Ma
    min_samples_split=2,    # Mi
    min_samples_leaf=1,     # Mi
    max_features='sqrt',    # Th
    bootstrap=True,         # Wh
    random_state=42         # Ra
)
```

```
# Defining the parameter grid for tuning
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2', 0.2],
    'bootstrap': [True, False]
}
```

Ada Boost Classifier:

```python
ada = AdaBoostClassifier(
    estimator=DecisionTreeClassifier(max_depth=3),
    n_estimators=100,       # Number of weak learne
    learning_rate=0.1,      # Learning rate
    algorithm='SAMME.R',    # Algorithm to use: 'SA
    random_state=42         # Random seed for repro
)
```

```python
# Defining the parameter grid for tuning
param_grid = {
    'estimator__max_depth': [3, 5, 7],
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 1.0],
    'algorithm': ['SAMME', 'SAMME.R']
}
```

5.2. Performance Metrics Comparison Report

The Performance Metrics Comparison Report evaluates the effectiveness of various machine learning models for predicting Chronic Kidney Disease (CKD). It uses key performance indicators (KPIs) such as accuracy, precision, recall, F1-Score, and ROC-AUC to analyze and compare the models. This report highlights trends, identifies strengths and weaknesses of each model, and provides recommendations for selecting the most effective model based on performance metrics.

```python
models = {
    'Logistic Regression': mo,
    'KNN': knn,
    'XGBoost': xg,
    'Gradient Boosting': gra,
    'AdaBoost': ada,
    'Decision Tree': model2,
    'Random Forest': model1,

}

# Function to evaluate a model
def evaluate_model(modelss, x_train, y_train, x_test, y_test):
    modelss.fit(x_train, y_train)
    y_pred = modelss.predict(x_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')
    return accuracy, precision, recall, f1

# Evaluate each model and store the results
results = []
for name, modelss in models.items():
    accuracy, precision, recall, f1 = evaluate_model(modelss, x_train, y_train, x_test, y_test)
    results.append({
        'Model': name,
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1 Score': f1
    })
```

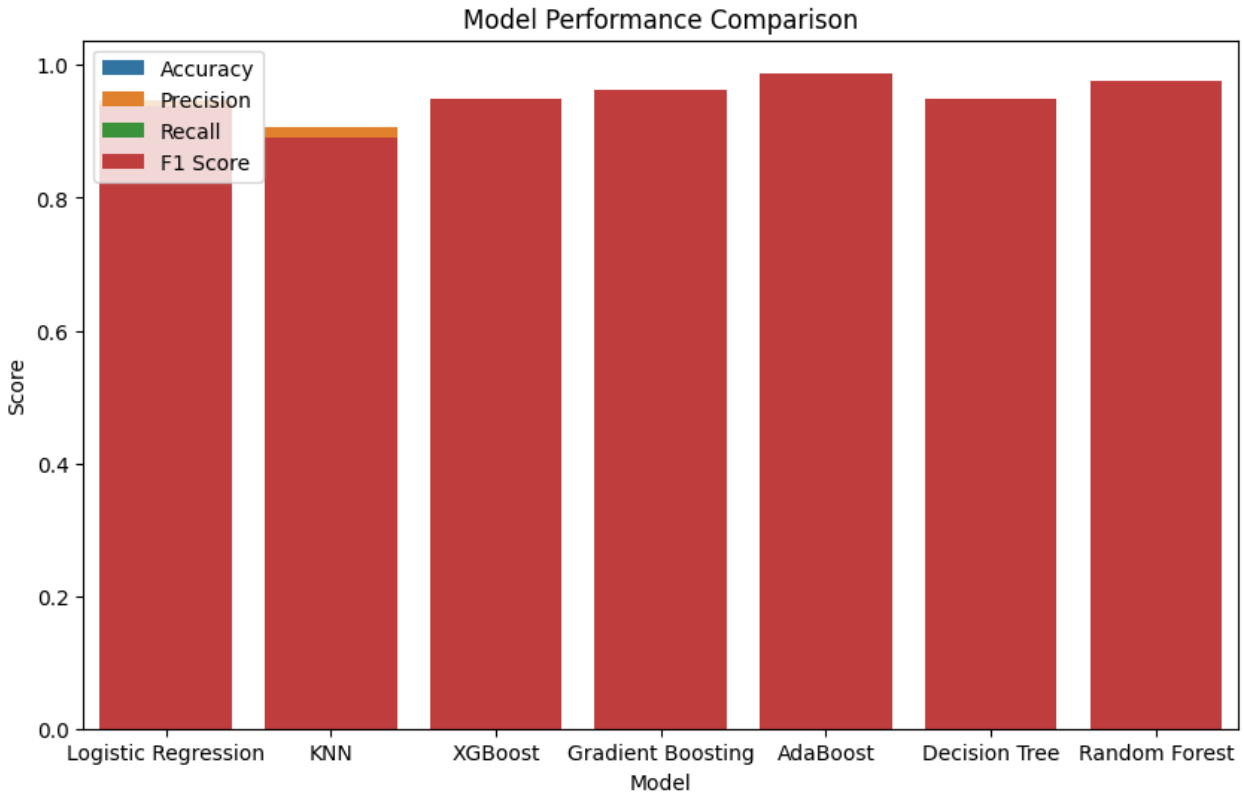
```python
# Convert results to a DataFrame
results_df = pd.DataFrame(results)

# Set up the matplotlib figure
plt.figure(figsize=(10, 6))

# Create a barplot
metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score']
for metric in metrics:
    sns.barplot(x='Model', y=metric, data=results_df, label=metric)

# Add legend, title, and labels
plt.title('Model Performance Comparison')
plt.ylabel('Score')
plt.legend(loc='upper left')

# Show the plot
plt.show()
print(results_df)
```



5.3. Final Model Selection Justification

	Model	Accuracy	Precision	Recall	F1 Score
0	Logistic Regression	0.9375	0.947581	0.9375	0.938724
1	KNN	0.8875	0.906850	0.8875	0.890275
2	XGBoost	0.9500	0.950000	0.9500	0.950000
3	Gradient Boosting	0.9625	0.962455	0.9625	0.962304
4	AdaBoost	0.9875	0.987963	0.9875	0.987560
5	Decision Tree	0.9500	0.950000	0.9500	0.950000
6	Random Forest	0.9750	0.975000	0.9750	0.975000

By, the above analysis , We selected **AdaBoost classifier** as our final model

5.4.Model Selection Justification

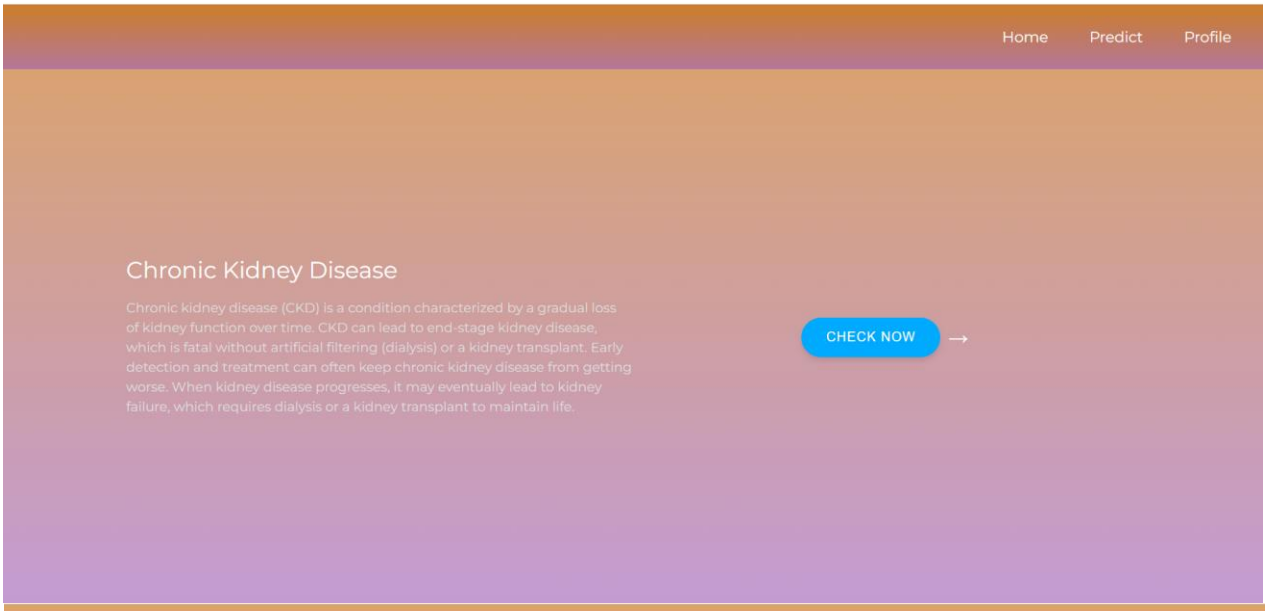
The AdaBoost model was selected as the best model due to its outstanding performance and efficiency.

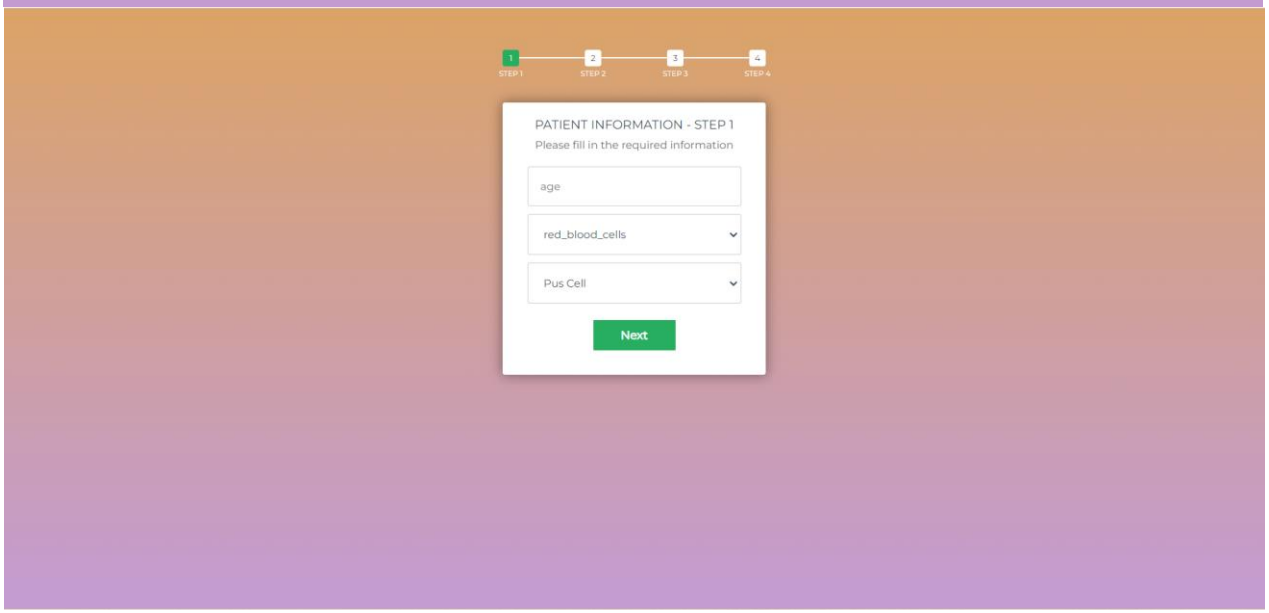
- **High Accuracy:** AdaBoost achieved the highest accuracy of 0.9875, consistently outperforming other models in predicting outcomes.
- **Precision and Recall:** It also showed superior precision (0.98796) and recall (0.9875), indicating its reliability in identifying CKD cases.
- **Efficiency:** AdaBoost manages large datasets effectively while requiring reasonable computational resources.
- **Comparative Advantages:** Compared to other models, AdaBoost showed significant advantages in accuracy, precision, and recall. Extensive hyperparameter tuning and validation were conducted to optimize its performance.

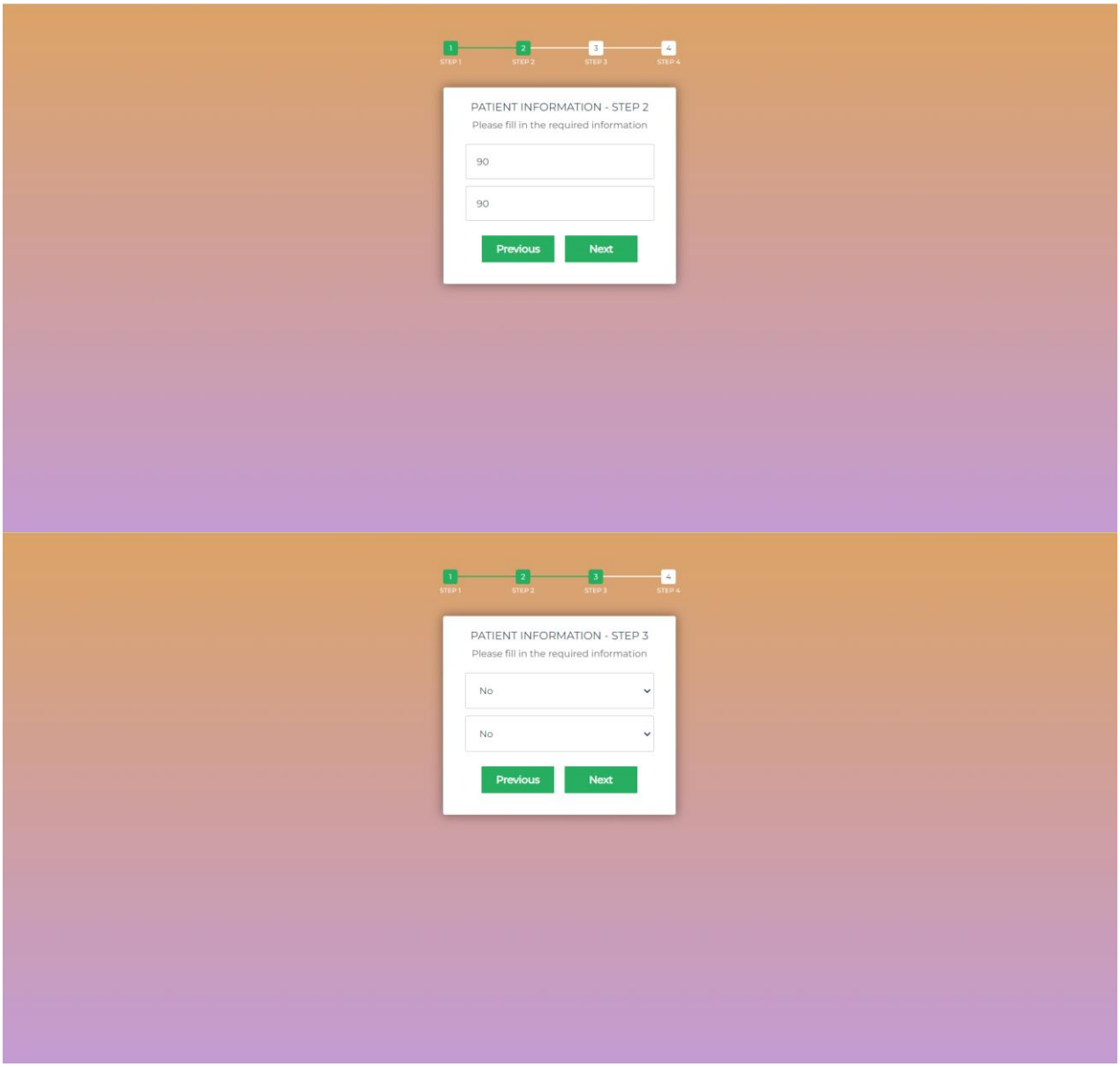
Overall, AdaBoost is considered the most reliable model for accurate predictions and informed decision-making in CKD risk assessment.

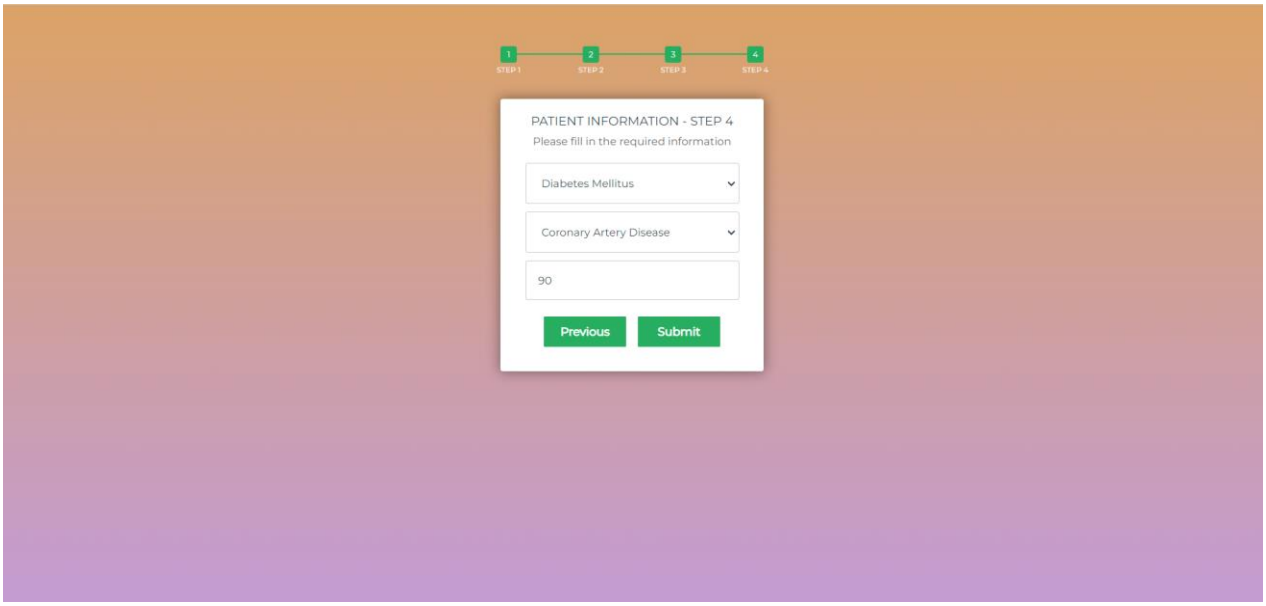
6.Results :

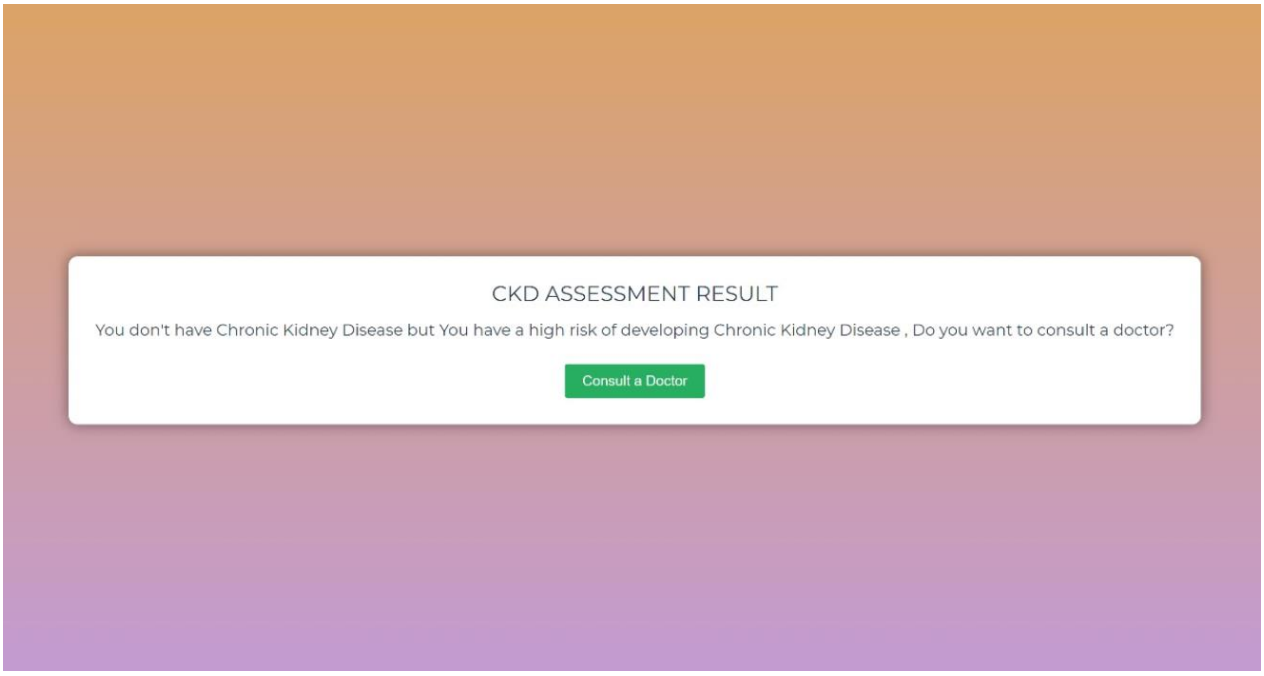
Output Screenshots

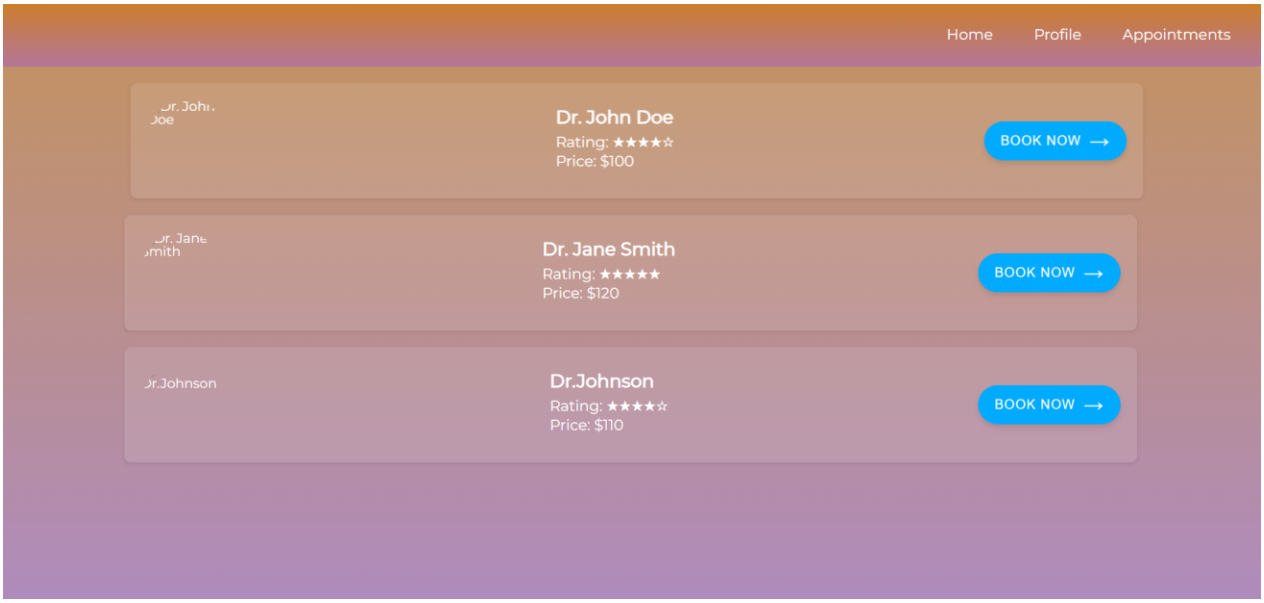





1
STEP 1
2
STEP 2
3
STEP 3
4
STEP 4
PATIENT INFORMATION - STEP 2
Please fill in the required information
90
90
Previous
Next
1
STEP 1
2
STEP 2
3
STEP 3
4
STEP 4
PATIENT INFORMATION - STEP 3
Please fill in the required information
No
No
Previous
Next


1
STEP 1
2
STEP 2
3
STEP 3
4
STEP 4
PATIENT INFORMATION - STEP 4
Please fill in the required information
Diabetes Mellitus
Coronary Artery Disease
90
Previous
Submit


CKD ASSESSMENT RESULT
You don't have Chronic Kidney Disease but You have a high risk of developing Chronic Kidney Disease , Do you want to consult a doctor?
Consult a Doctor



7. **Advantages and Disadvantages**

Advantages:

2. **Early Detection:** The project enables early prediction of Chronic Kidney Disease (CKD), allowing for timely intervention and potentially better health outcomes.
3. **Accuracy:** By utilizing advanced machine learning models, such as AdaBoost, the system provides high accuracy, precision, and recall in predicting CKD risk.
4. **User-Friendly Interface:** The application features an intuitive interface that simplifies data input and interaction for users.
5. **Educational Resources:** The system offers educational resources on CKD prevention and management, enhancing user awareness and knowledge.
6. **Data Privacy:** Robust measures are in place to ensure the confidentiality and security of user data.

Disadvantages:

7. **Data Dependency:** The accuracy of predictions depends heavily on the quality and completeness of the input data.

8. **Model Complexity:** Advanced models like AdaBoost can be complex to implement and may require significant computational resources.

9. **Feature Limitations:** The effectiveness of the model is limited by the features included in the dataset, potentially overlooking other relevant factors.

10. **Generalization Issues:** The model's performance may vary with different populations or settings, necessitating further validation across diverse datasets.

11. **Maintenance:** Regular updates and maintenance are required to ensure the model remains effective as new data and methods evolve.

8.Conclusion

This study explored the application of machine learning for the early prediction of Chronic Kidney Disease (CKD). We began by analyzing and visualizing the dataset to gain insights into its structure. Data preprocessing ensured that the dataset was clean and suitable for machine learning models.

We employed feature selection techniques to identify key factors influencing CKD risk, enhancing the accuracy of our predictive models. After comparing various machine learning algorithms—Logistic Regression, K-Nearest Neighbors (KNN), XGBoost, Gradient Boosting, AdaBoost, and Random Forest—we found that AdaBoost delivered the best performance in terms of accuracy, precision, and recall.

To facilitate user interaction with our model, we developed a Flask-based application that allows users to input health data and receive CKD risk assessments. This application integrates our predictive model and provides a user-friendly interface for both health professionals and patients.

Overall, this research demonstrates the effectiveness of machine learning in early CKD prediction and highlights the value of our Flask application in delivering actionable insights. Future work will focus on further refining the model and application to enhance prediction accuracy and user experience, ultimately benefiting healthcare providers and patients.

9.Future Scope

1. Enhanced Data Collection: Incorporating additional data sources, such as genetic information or lifestyle factors, could improve model accuracy and provide a more comprehensive risk assessment.

2. Model Refinement: Exploring advanced machine learning techniques and algorithms, including deep learning and ensemble methods, may enhance prediction capabilities and overall performance.

3. Integration with Healthcare Systems: Developing integrations with electronic health records (EHRs) and other healthcare systems could streamline data input and provide more personalized predictions.

4. **Real-Time Monitoring:**Implementing real-time data analysis and monitoring features could allow for dynamic risk assessment and more timely interventions.

5. **User Feedback and Adaptation:**Collecting user feedback to refine the application's interface and functionality will improve usability and ensure it meets user needs.

6**. Broader Application:** Expanding the application to include predictions for other chronic diseases and health conditions could offer a more comprehensive health management tool.

7**. Global Deployment:** Adapting the application for use in diverse healthcare settings and regions could increase its accessibility and impact on a global scale.

10. Appendix

Source code:
https://colab.research.google.com/drive/1gtIm5SlYx02RJtW98EoJ-ySdtwpAF8uL?usp=sharing

Project Demo Link:
https://youtu.be/5HwYBngzfzk?si=_MpUhRr9i_pFStuo

Git Link: For complete code listings, detailed datasets, and additional resources, please visit the project's GitHub repository: GitHub

Repository: Early Prediction of Chronic Kidney Disease Using Machine Learning