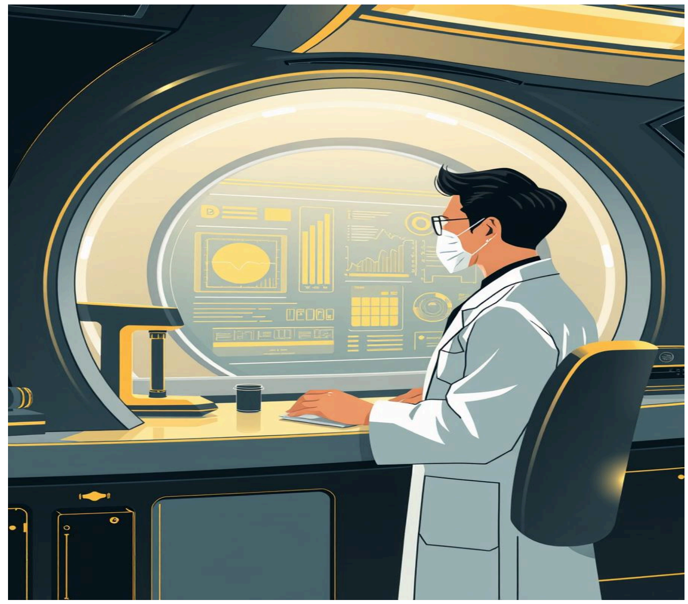
# project:

# Early Prediction for Chronic Kidney Disease Detection

Prepared by
**Mohit prajapati**
Ganpat University

# ***Early Prediction for Chronic Kidney Disease Detection***

**A Progressive Approach to Health Management**



# Welcome


**Name: Mohit prajapati**

**collage name: Ganpat University**

**Course:  AMPICS (BCA(3rd)) year**

**Roll number: 23032431034**

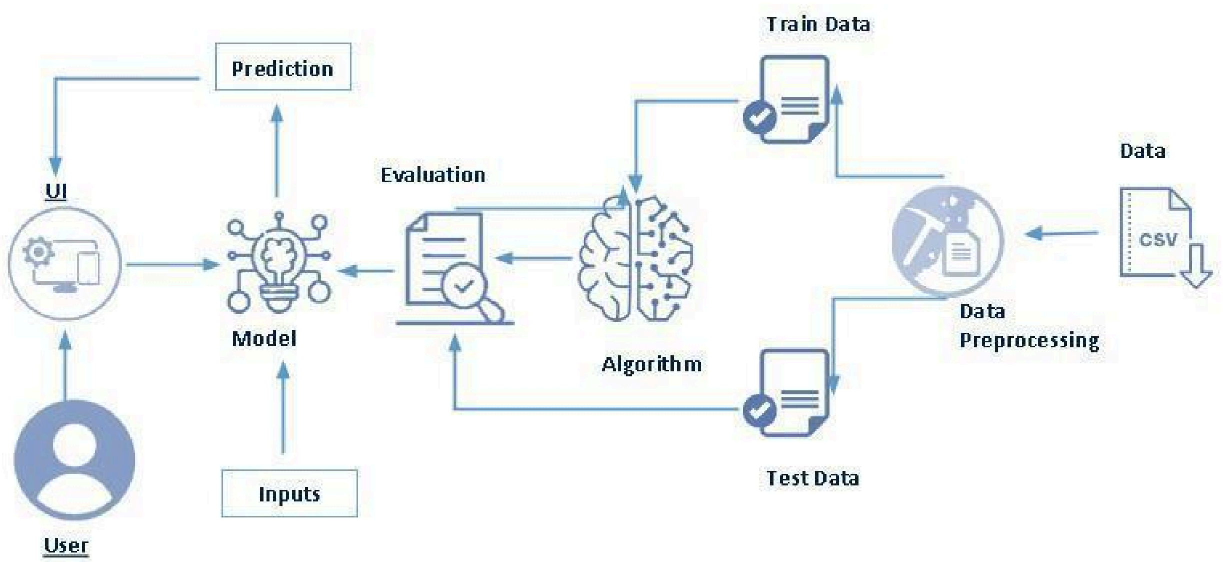
**contact Number: 6352967486**

**Emai Id : mohitprajapati4412@gmail.com / 23032431034@gnu.ac.**

# Early Prediction for Chronic Kidney Disease Detection using Machin learning

Machine learning (ML) provides a powerful way to detect Chronic Kidney Disease (CKD) at its earliest stages, long before severe symptoms appear. By analyzing complex medical data, ML models can identify hidden patterns and risk factors that even an experienced doctor might miss. This early detection is crucial because it allows for timely intervention, which can slow down the disease's progression and significantly improve patient outcomes.

## Technical Architecture:




# Project Flow

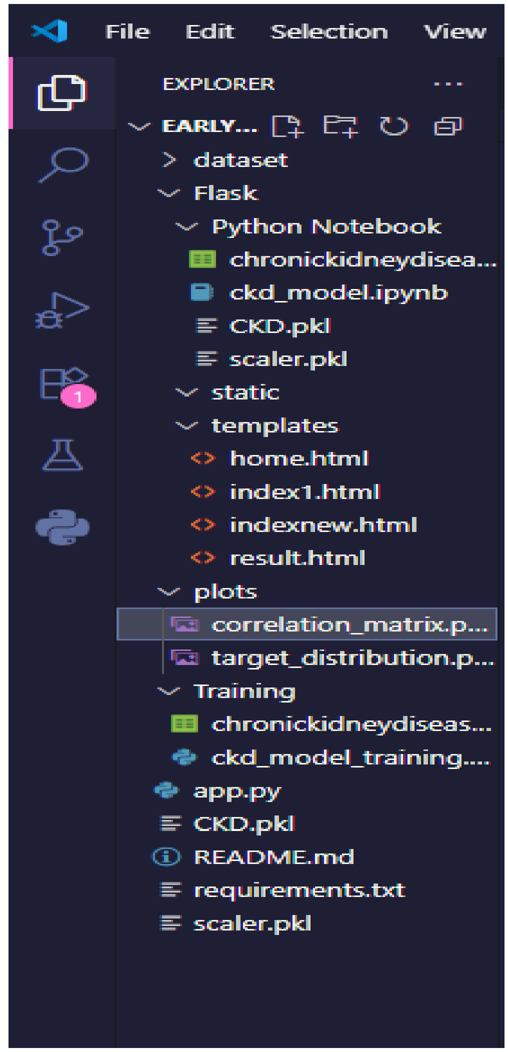
- User interacts with the UI to enter the input.
- Entered input is analysed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI

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

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

# Project Structure

Create the Project folder which contains files as shown below




We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.

CKD.pkl is our saved model. Further we will use this model for flask integration.

Training folder contains a model training file.

# Milestone 1: Define Problem / Problem Understanding

- **Business Problem:** You've clearly linked the project to the real-world challenge of Chronic Kidney Disease (CKD) and the need for early detection using existing medical test data.
- **Key Requirements:** Your requirements are specific and measurable. Focusing on minimizing false positives/negatives and ensuring model transparency is crucial in a medical context.
- **Literature Survey:** You've provided solid background information, establishing the significance of the problem and acknowledging that machine learning is a common and effective approach.
- **Impact:** You've successfully outlined the significant positive social and business impacts, such as improving patient quality of life and reducing healthcare costs.

# Milestone 2: Data Collection & Preparation

## Activity 1: Collect the dataset


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

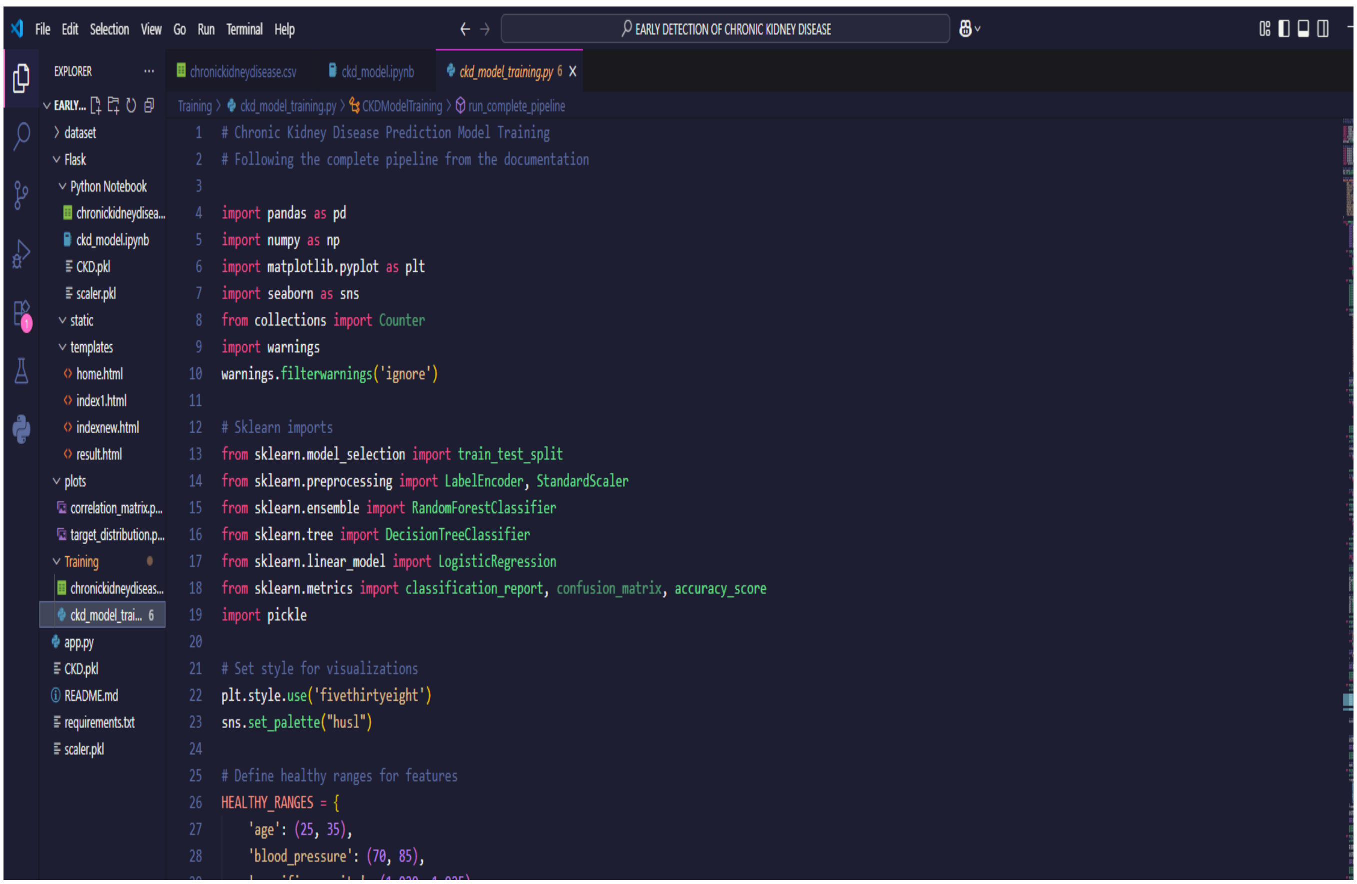
In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link: [https://www.kaggle.com/datasets/mansoordaku/ckdisease](https://www.kaggle.com/datasets/mansoordaku/ckdisease)

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

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

### Activity 1.1: Importing the libraries




## Activity 1.2: Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of the csv file.


```python
chronic_dataset= pd.read_csv("kidney_disease.csv")
```

```python
chronic_dataset
```

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

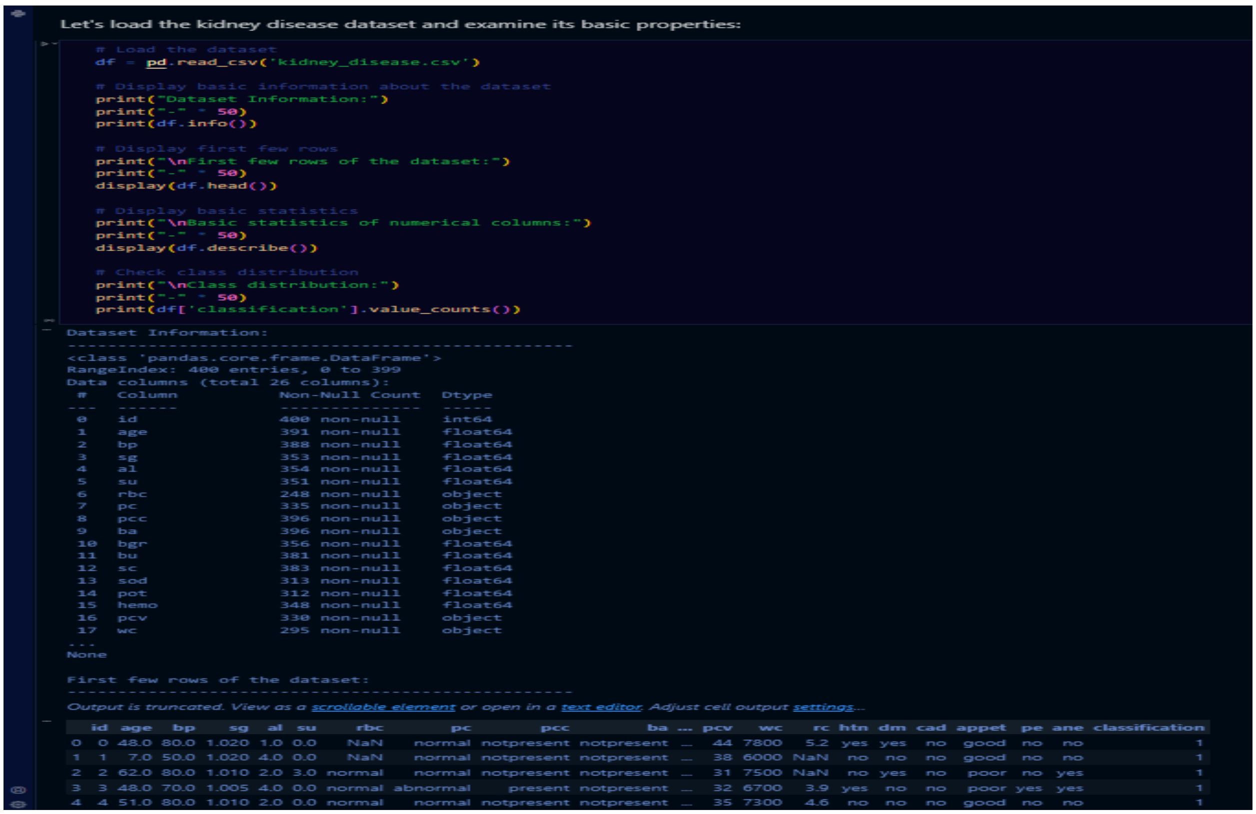
## Activity 2: Data Preparation

As we have understood how the data is, let's pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

- Rename the columns
- Handling missing values
- Handling categorical data
- Handling Numerical data

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.



Let's load the kidney disease dataset and examine its basic properties:

```python
# Load the dataset
df = pd.read_csv('kidney_disease.csv')

# Display basic information about the dataset
print("Dataset Information:")
print("-" * 50)
print(df.info())

# Display first few rows
print("\nFirst few rows of the dataset:")
print("-" * 50)
display(df.head())

# Display basic statistics
print("\nBasic statistics of numerical columns:")
print("-" * 50)
display(df.describe())

# Check class distribution
print("\nClass distribution:")
print("-" * 50)
print(df['classification'].value_counts())
```

```
Dataset Information:
--------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 26 columns):
 #    Column           Non-Null Count   Dtype
---   ------           --------------   -----
 0    id               400 non-null     int64
 1    age              391 non-null     float64
 2    bp               388 non-null     float64
 3    sg               353 non-null     float64
 4    al               354 non-null     float64
 5    su               351 non-null     float64
 6    rbc              248 non-null     object
 7    pc               335 non-null     object
 8    pcc              396 non-null     object
 9    ba               396 non-null     object
 10   bgr              356 non-null     float64
 11   bu               381 non-null     float64
 12   sc               383 non-null     float64
 13   sod              313 non-null     float64
 14   pot              312 non-null     float64
 15   hemo             348 non-null     float64
 16   pcv              330 non-null     object
 17   wc               295 non-null     object
...
None

First few rows of the dataset:
--------------------------------------------------
```

*Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...*

| | 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 | 1 |
| 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 | 1 |
| 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 | 1 |
| 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 | 1 |
| 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 | 1 |

## Activity 2.1: Handling missing values

- 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.

app.py | Kidney_disease.ipynb | ckd_model_training.py 3 | new_model_training.py | retrain_model.py

CKD_Prediction_Project > python Notebook > Kidney_disease.ipynb > chronic_dataset= pd.read_csv("kidney_disease.csv")

Generate + Code + Markdown | Run All Restart Clear All Outputs | Jupyter Variables Outline ...

[46]

```
chronic_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 26 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   id              400 non-null    int64
 1   age             391 non-null    float64
 2   bp              388 non-null    float64
 3   sg              353 non-null    float64
 4   al              354 non-null    float64
 5   su              351 non-null    float64
 6   rbc             248 non-null    object
 7   pc              335 non-null    object
 8   pcc             396 non-null    object
 9   ba              396 non-null    object
 10  bgr             356 non-null    float64
 11  bu              381 non-null    float64
 12  sc              383 non-null    float64
 13  sod             313 non-null    float64
 14  pot             312 non-null    float64
 15  hemo            348 non-null    float64
 16  pcv             330 non-null    object
 17  wc              295 non-null    object
 18  rc              270 non-null    object
 19  htn             398 non-null    object
...
 24  ane             399 non-null    object
 25  classification  400 non-null    int64
dtypes: float64(11), int64(2), object(13)
memory usage: 81.4+ KB
```

### Activity 2.4: Handling Numerical columns

```
contcols=set(data.dtypes[data.dtypes!='O'].index.values)# only fetech the float and int type columns
#contcols=pd.DataFrame(data,columns=contcols)
print(contcols)
```

```
{'blood_urea', 'serum_creatinine', 'albumin', 'blood_pressure', 'blood glucose random', 'sugar', 'sodium', 'hemoglobin', 'specific_gravit
y', 'age', 'potassium'}
```

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

```
for i in contcols:
    print("Continous Columns :",i)
    print(c(data[i]))
    print('*'*120+'\n')
```

If we observe the output of the above code we can observe that some columns have few values or you can say classes which can be considered as categorical columns. So , let's remove it and add the columns which we observed into their respective variables.

```
contcols.remove('specific_gravity')
contcols.remove('albumin')
contcols.remove('sugar')
print(contcols)

```

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

```
contcols.add('red_blood_cell_count') # using add we can add the column
contcols.add('packed_cell_volume')
contcols.add('white_blood_cell_count')
print(contcols)
```

```
{'blood_urea', 'serum_creatinine', 'packed_cell_volume', 'blood_pressure', 'blood glucose random', 'sodium', 'hemoglobin', 'red_blood_cell
_count', 'age', 'potassium', 'white_blood_cell_count'}
```

```
catcols.add('specific_gravity')
catcols.add('albumin')
catcols.add('sugar')
print(catcols)
```

```
{'hypertension', 'class', 'albumin', 'coronary_artery_disease', 'anemia', 'sugar', 'red_blood_cells', 'specific_gravity', 'bacteria', 'ped
al_edema', 'appetite', 'pus_cell', 'diabetesmellitus', 'pus_cell_clumps'}
```

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

```
data['coronary_artery_disease'] = data.coronary_artery_disease.replace('\tno','no') # replacing \tno wi
c(data['coronary_artery_disease'])
```

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

```
data['diabetesmellitus'] = data.diabetesmellitus.replace(to_replace={'\tno':'no','\tyes':'yes',' yes':'
c(data['diabetesmellitus'])
```

```
Counter({'yes': 137, 'no': 261, nan: 2})
```

# Milestone 3: Exploratory Data Analysis

## Activity 1: Descriptive statistical Analysis

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

```
chronic_dataset.describe()
```

| | 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 |

```
chronic_dataset.shape
```

```
(400, 26)
```

```
chronic_dataset['classification'].value_counts()
```

```
classification
1    250
0    150
Name: count, dtype: int64
```

```
chronic_dataset['classification']
```

# Activity 2: Visual analysis

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

## Activity 2.1: Univariate analysis

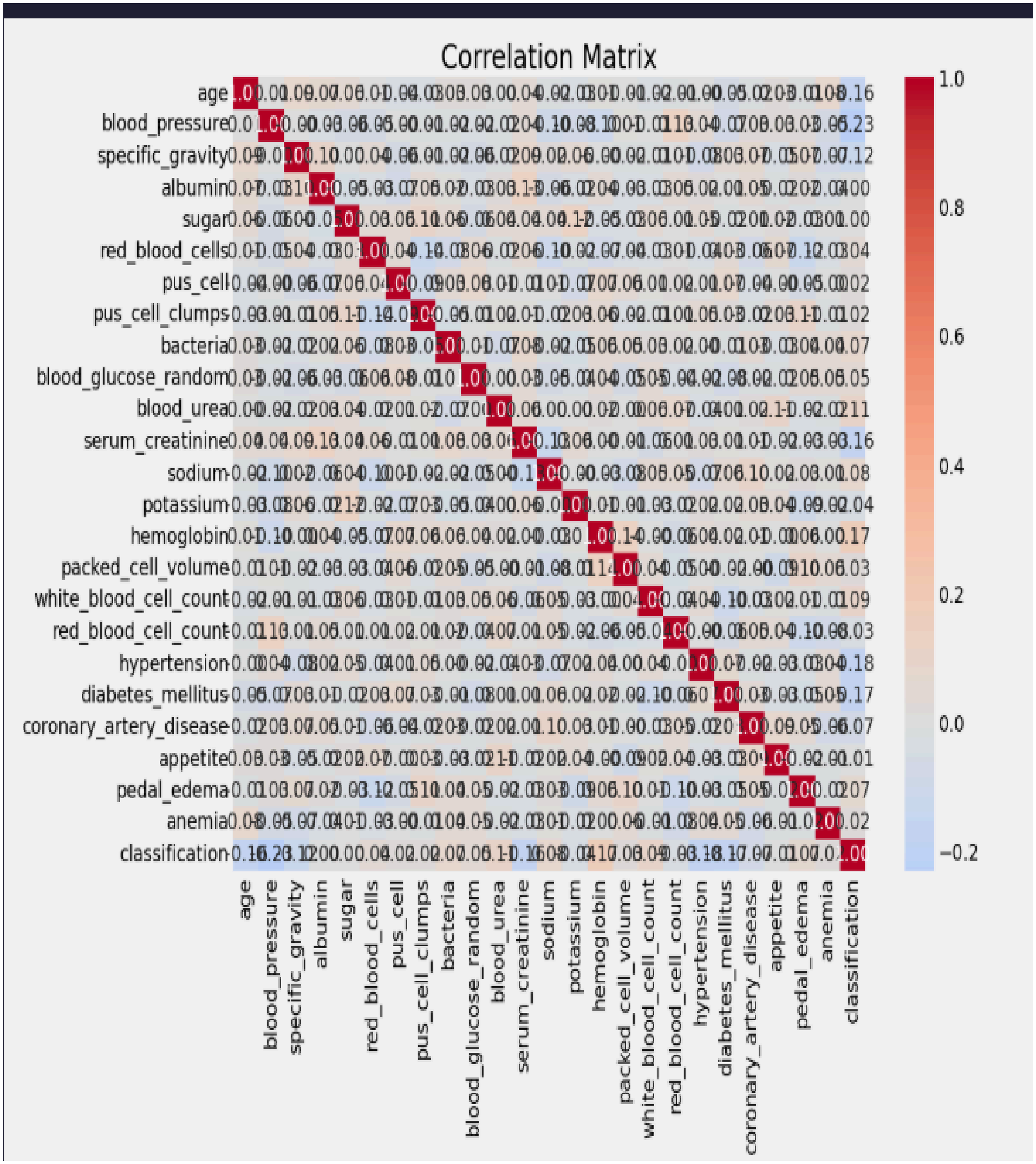
In simple words, univariate analysis is understanding the data with a single feature. Here we have displayed two different graphs such as distplot and countplot.

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

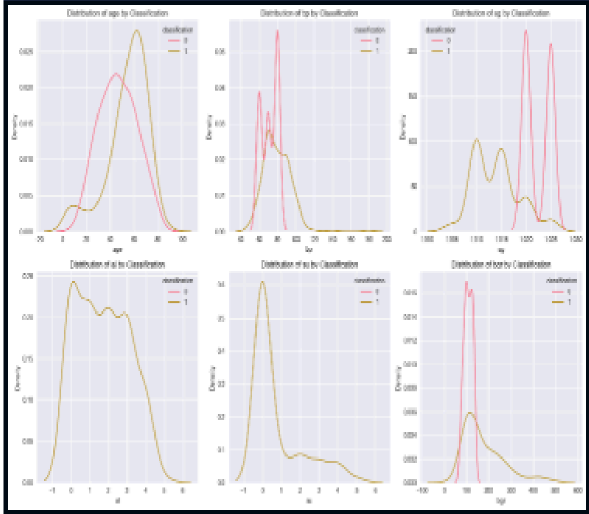
# Data Analysis and Visualization

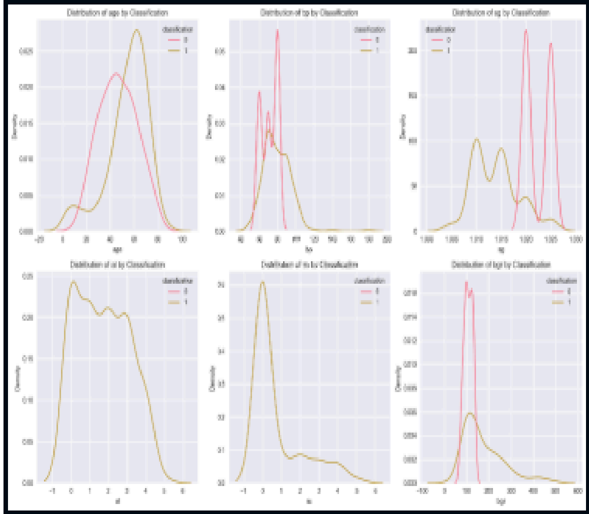
Let's analyze the relationships between different features and their impact on kidney disease:

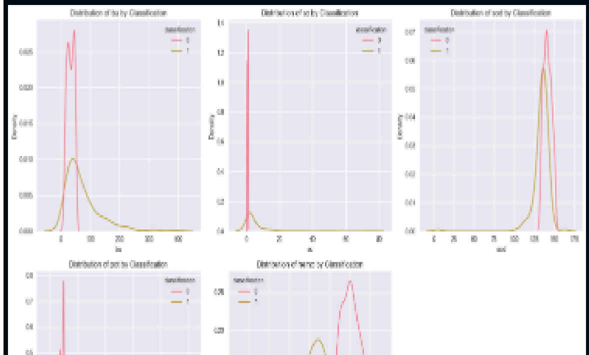
```
# Set up the matplotlib figure size for better visibility
plt.figure(figsize=(12, 8))

# Select only numeric columns for correlation
numeric_df = df.select_dtypes(include=[np.number])

# Create correlation matrix
correlation_matrix = numeric_df.corr()

# Create heatmap
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0)
plt.title('Correlation Matrix of Numerical Features')
plt.tight_layout()
plt.show()

# Display the most correlated features with the target variable
correlation_with_target = correlation_matrix['classification'].sort_values(ascending=False)
print("\nCorrelation with Classification (Target Variable):")
print("-" * 50)
print(correlation_with_target)
```


Correlation Matrix
age
blood_pressure
specific_gravity
albumin
sugar
red_blood_cells
pus_cell
pus_cell_clumps
bacteria
blood_glucose_random
blood_urea
serum_creatinine
sodium
potassium
hemoglobin
packed_cell_volume
white_blood_cell_count
red_blood_cell_count
hypertension
diabetes_mellitus
coronary_artery_disease
appetite
pedal_edema
anemia
classification
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

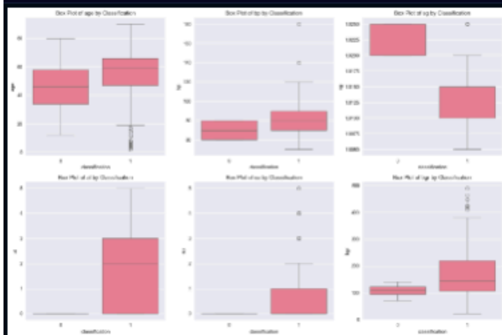
```python
# Select important numerical features (only numeric columns)
numerical_features = [col for col in ['age', 'bp', 'sg', 'al', 'su', 'bgr', 'bu', 'sc', 'sod', 'pot', 'hemo', 'pcv', 'wc', 'rc']
                      if pd.api.types.is_numeric_dtype(df[col])]

# Create distribution plots
plt.figure(figsize=(15, 10))
for i, feature in enumerate(numerical_features[:6], 1):
    plt.subplot(2, 3, i)
    sns.kdeplot(data=df, x=feature, hue='classification', common_norm=False)
    plt.title(f'Distribution of {feature} by Classification')
plt.tight_layout()
plt.show()

# Create second set of distribution plots
plt.figure(figsize=(15, 10))
for i, feature in enumerate(numerical_features[6:12], 1):
    plt.subplot(2, 3, i)
    sns.kdeplot(data=df, x=feature, hue='classification', common_norm=False)
    plt.title(f'Distribution of {feature} by Classification')
plt.tight_layout()
plt.show()
```

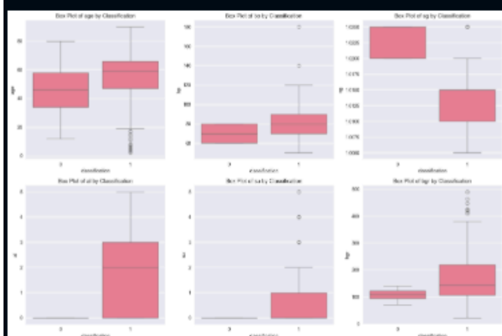


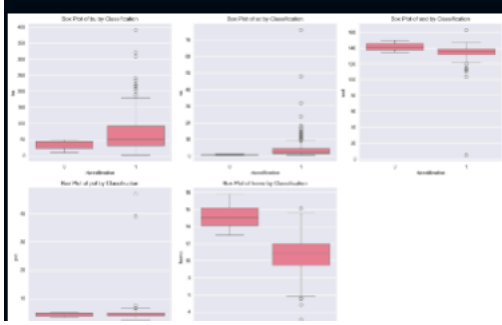



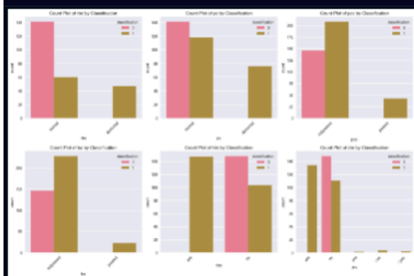
```python
# Create box plots for numerical features
plt.figure(figsize=(15, 10))
for i, feature in enumerate(numerical_features[:6], 1):
    plt.subplot(2, 3, i)
    sns.boxplot(data=df, x='classification', y=feature)
    plt.title(f'Box Plot of {feature} by Classification')
plt.tight_layout()
plt.show()

# Create second set of box plots
plt.figure(figsize=(15, 10))
for i, feature in enumerate(numerical_features[6:12], 1):
    plt.subplot(2, 3, i)
    sns.boxplot(data=df, x='classification', y=feature)
    plt.title(f'Box Plot of {feature} by Classification')
plt.tight_layout()
plt.show()
```

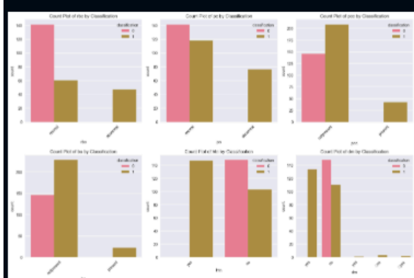




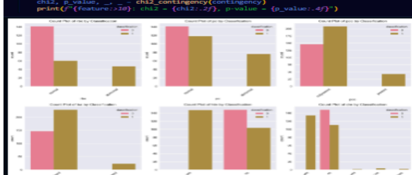


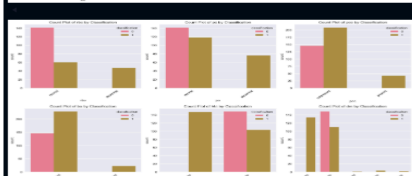
```python
# Analyze categorical features
categorical_features = ['rbc', 'pc', 'pcc', 'ba', 'htn', 'dm', 'cad', 'appet', 'pe', 'ane']

# Create count plots for categorical features
plt.figure(figsize=(15, 10))
for i, feature in enumerate(categorical_features[:6], 1):
    plt.subplot(2, 3, i)
    sns.countplot(data=df, x=feature, hue='classification')
    plt.title(f'Count Plot of {feature} by Classification')
    plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Create second set of count plots
plt.figure(figsize=(15, 6))
for i, feature in enumerate(categorical_features[6:], 1):
    plt.subplot(2, 2, i)
    sns.countplot(data=df, x=feature, hue='classification')
    plt.title(f'Count Plot of {feature} by Classification')
    plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Calculate chi-square test for categorical variables
from scipy.stats import chi2_contingency

print("\nChi-square Test Results for Categorical Features:")
print("-" * 50)
for feature in categorical_features:
    contingency = pd.crosstab(df[feature], df['classification'])
    chi2, p_value, _, _ = chi2_contingency(contingency)
    print(f"{feature:>10}: chi2 = {chi2:.2f}, p-value = {p_value:.4f}")
```





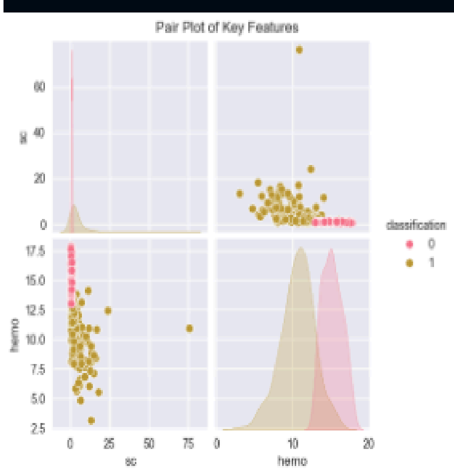

```python
categorical_features = ['rbc', 'pc', 'pcc', 'ba', 'htn', 'dm', 'cad', 'appet', 'pe', 'ane']

# Create count plots for categorical features
plt.figure(figsize=(15, 10))
for i, feature in enumerate(categorical_features[:6], 1):
    plt.subplot(2, 3, i)
    sns.countplot(data=df, x=feature, hue='classification')
    plt.title(f'Count Plot of {feature} by Classification')
    plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Create second set of count plots
plt.figure(figsize=(15, 6))
for i, feature in enumerate(categorical_features[6:], 1):
    plt.subplot(2, 2, i)
    sns.countplot(data=df, x=feature, hue='classification')
    plt.title(f'Count Plot of {feature} by Classification')
    plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Calculate chi-square test for categorical variables
from scipy.stats import chi2_contingency

print("\nChi-square Test Results for Categorical Features:")
print("-" * 50)
for feature in categorical_features:
    contingency = pd.crosstab(df[feature], df['classification'])
    chi2, p_value, _, _ = chi2_contingency(contingency)
    print(f"{feature:>10}: chi2 = {chi2:.2f}, p-value = {p_value:.4f}")
```



```python
# Create pair plot
plt.figure(figsize=(12, 12))
sns.pairplot(df[important_features], hue='classification', diag_kind='kde')
plt.suptitle('Pair Plot of Key Features', y=1.02)
plt.show()

# Summary of findings
print("\nKey Findings from Data Analysis:")
print("-" * 50)
print("1. Correlation Analysis:")
print("   - Strong correlations observed between specific features")
print("   - Key predictors identified through correlation with target variable")
print("\n2. Distribution Analysis:")
print("   - Clear differences in distributions between chronic and non-chronic cases")
print("   - Several features show distinct patterns for each class")
print("\n3. Categorical Features:")
print("   - Significant associations found through chi-square tests")
print("   - Some categorical variables strongly indicate disease status")
```

```
Figure size 1200x1200 with 0 Axes>

Figure size 1200x1200 with 0 Axes>
```




```
ey Findings from Data Analysis:
--------------------------------------------------
. Correlation Analysis:
  - Strong correlations observed between specific features
  - Key predictors identified through correlation with target variable

. Distribution Analysis:
  - Clear differences in distributions between chronic and non-chronic cases
  - Several features show distinct patterns for each class
```

# Milestone 4: Model Building

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project we are applying four classification algorithms. The best model is saved based on its performance.

## Activity 1.1: ANN Model

Building and training an Artificial Neural Network (ANN) using the Keras library with TensorFlow as the backend. The ANN is initialised as an instance of the Sequential class, which is a linear stack of layers. Then, the input layer and two hidden layers are added to the model using the Dense class, where the number of units and activation function are specified. The output layer is also added using the Dense class with a sigmoid activation function. The model is then compiled with the Adam optimizer, binary cross-entropy loss function, and accuracy metric. Finally, the model is fit to the training data with a batch size of 100, 20% validation split, and 100 epochs

```
 Deep Learning imports
rom tensorflow.keras.models import Sequential
rom tensorflow.keras.layers import Dense
rom tensorflow.keras.optimizers import Adam

# Create output directory for plots
import os
if not os.path.exists('plots'):
    os.makedirs('plots')

# Descriptive statistics
print("Descriptive Statistics:")
print(self.df.describe())

# Target distribution
plt.figure(figsize=(8, 6))
target_counts = self.df['classification'].value_counts()
plt.pie(target_counts.values, labels=['No CKD', 'CKD'], autopct='%1.1f%%')
plt.title('Distribution of CKD Classification')
plt.savefig('plots/target_distribution.png')
plt.close()
```

# Activity 1.2: Random Forest model

A function named random Forest is created and train and test data are passed as the parameters. Inside the function, Random Forest Classifier algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with. predict() function and saved in a new variable. For evaluating the model, a confusion matrix and classification report is done.

```
rom sklearn.model_selection import train_test_split
rom sklearn.preprocessing import LabelEncoder, StandardScaler
rom sklearn.ensemble import RandomForestClassifier
rom sklearn.tree import DecisionTreeClassifier
rom sklearn.linear_model import LogisticRegression
rom sklearn.metrics import classification_report, confusion_matrix, accuracy_score
mport pickle
```

# Activity 1.3: Decision tree model

A function named decision Tree is created and train and test data are passed as the parameters. Inside the function, Decision Tree Classifier algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with. predict() function and saved in a new variable. For evaluating the model, a confusion matrix and classification report is done.

```
    print(f"Decision Tree Accuracy: {accuracy:.4f}")
    print("Classification Report:")
    print(classification_report(self.y_test, y_pred_dt))

ef build_logistic_regression(self):
    """Build Logistic Regression model"""
    print("\n=== BUILDING LOGISTIC REGRESSION MODEL ===")

    lr_model = LogisticRegression(random_state=42, max_iter=1000)
    lr_model.fit(self.X_train_scaled, self.y_train)

    y_pred_lr = lr_model.predict(self.X_test_scaled)
    accuracy = accuracy_score(self.y_test, y_pred_lr)

    self.models['Logistic Regression'] = lr_model
    self.model_scores['Logistic Regression'] = accuracy

    print(f"Logistic Regression Accuracy: {accuracy:.4f}")
    print("Classification Report:")
```

# Activity 1.4: Logistic Regression

```python
    def build_logistic_regression(self):
        """Build Logistic Regression model"""
        print("\n=== BUILDING LOGISTIC REGRESSION MODEL ===")

        lr_model = LogisticRegression(random_state=42, max_iter=1000)
        lr_model.fit(self.X_train_scaled, self.y_train)

        y_pred_lr = lr_model.predict(self.X_test_scaled)
        accuracy = accuracy_score(self.y_test, y_pred_lr)

        self.models['Logistic Regression'] = lr_model
        self.model_scores['Logistic Regression'] = accuracy

        print(f"Logistic Regression Accuracy: {accuracy:.4f}")
        print("Classification Report:")
        print(classification_report(self.y_test, y_pred_lr))

    def evaluate_models(self):
        """Evaluate and compare all models"""
        print("\n=== MODEL EVALUATION ===")
```

# Activity 2: Testing the model

```python
# Make predictions on training and test data
y_train_pred = svm_classifier.predict(X_train)
y_test_pred = svm_classifier.predict(X_test)

# Calculate accuracy scores
train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)

print("Model Performance Metrics:")
print("-" * 50)
print(f"Training Accuracy: {train_accuracy:.4f}")
print(f"Testing Accuracy: {test_accuracy:.4f}")

# Display detailed classification report
print("\nClassification Report:")
print("-" * 50)
print(classification_report(y_test, y_test_pred))

# Create confusion matrix
cm = confusion_matrix(y_test, y_test_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
```

## Milestone 5: Performance Testing & Evaluate the results

# 5. Model Evaluation

Evaluate the model's performance on both training and test datasets:

```
# Make predictions on training and test data
y_train_pred = svm_classifier.predict(X_train)
y_test_pred = svm_classifier.predict(X_test)

# Calculate accuracy scores
train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)

print("Model Performance Metrics:")
print("-" * 50)
print(f"Training Accuracy: {train_accuracy:.4f}")
print(f"Testing Accuracy: {test_accuracy:.4f}")

# Display detailed classification report
print("\nClassification Report:")
print("-" * 50)
print(classification_report(y_test, y_test_pred))

# Create confusion matrix
cm = confusion_matrix(y_test, y_test_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

## Activity 2: Evaluate the results

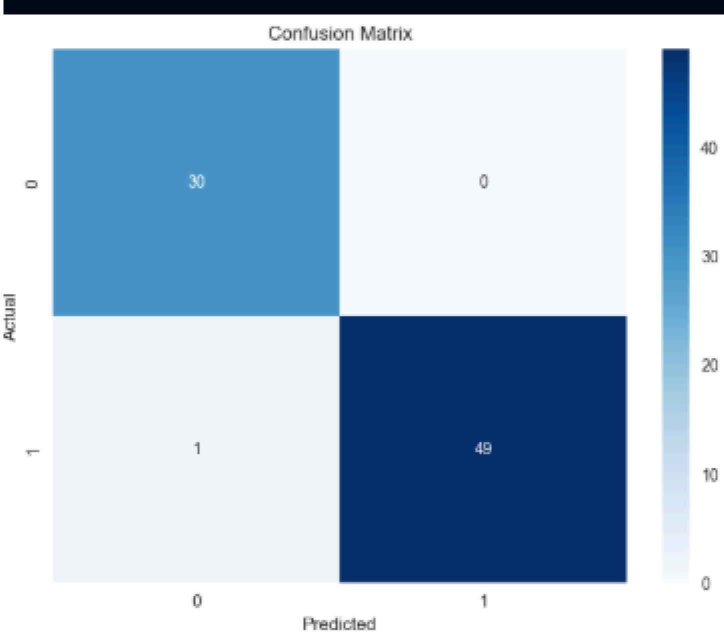
# 5. Model Evaluation

Evaluate the model's performance on both training and test datasets:

```
# Make predictions on training and test data
y_train_pred = svm_classifier.predict(X_train)
y_test_pred = svm_classifier.predict(X_test)

# Calculate accuracy scores
train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)

print("Model Performance Metrics:")
print("-" * 50)
print(f"Training Accuracy: {train_accuracy:.4f}")
print(f"Testing Accuracy: {test_accuracy:.4f}")

# Display detailed classification report
print("\nClassification Report:")
print("-" * 50)
print(classification_report(y_test, y_test_pred))

# Create confusion matrix
cm = confusion_matrix(y_test, y_test_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

# **Model Result:**

```
odel Performance Metrics:
--------------------------------------------------
raining Accuracy: 1.0000
esting Accuracy: 0.9875

lassification Report:
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.97      1.00      0.98        30
           1       1.00      0.98      0.99        50

    accuracy                           0.99        80
   macro avg       0.98      0.99      0.99        80
eighted avg       0.99      0.99      0.99        80
```



# Milestone 6: Model Deployment

## Activity 1: Save the best model

Create a function to make predictions on new data:

```python
def predict_kidney_disease(input_data):
    # Convert input data to numpy array
    input_array = np.asarray(input_data)

    # Reshape the array
    input_reshaped = input_array.reshape(1, -1)

    # Standardize the input
    input_standardized = scaler.transform(input_reshaped)

    # Make prediction
    prediction = svm_classifier.predict(input_standardized)

    if prediction[0] == 0:
        return "The person is Non-Chronic"
    else:
        return "The Person is Chronic"

# Example usage
# input_data = (38.0, 80.0, 1.02, 0.0, 0.0, 1, 1, 0, 0, 99.0, 19.0, 0.5, 147.0, 3.5, 13.6, 32, 68, 46, 0, 3, 1, 0, 0, 0)
input_data = (63.0, 90.0, 1.015,0.0, 0.0, 1, 1, 0, 0,123.0, 19.0, 2.0, 142.0, 3.8, 11.7, 34, 11400, 4.7,0,0,0,1,0,0)
result = predict_kidney_disease(input_data)
print("\nSample Prediction:")
print("-" * 50)
print(f"Input Data: {input_data}")
print(f"Prediction: {result}")
```

```
Sample Prediction:
--------------------------------------------------
Input Data: (63.0, 90.0, 1.015, 0.0, 0.0, 1, 1, 0, 0, 123.0, 19.0, 2.0, 142.0, 3.8, 11.7, 34, 11400, 4.7, 0, 0, 0, 1, 0, 0)
Prediction: The Person is Chronic
```

# Activity 2: Integrate with Web Framework

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

Building HTML Pages
Building server-side script
Run the web application

## Activity 2.1: Building Html Pages:

For this project create four HTML files namely

- home.html
- index1.html
- indexnew.html
- result.html

and save them in the templates folder.

## Activity 2.2: Build Python code:

Import the libraries

```
from flask import Flask, render_template, request, redirect, url_for
import pickle
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings('ignore')

```

**Load the saved model. Importing the flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as argument.**

```python
app = Flask(__name__)

# Load the trained model and scaler
try:
    with open('CKD.pkl', 'rb') as f:
        model = pickle.load(f)
    print("Model loaded successfully!")
except:
    print("Model file not found. Please run the training script first.")
    model = None

try:
    with open('scaler.pkl', 'rb') as f:
        scaler = pickle.load(f)
    print("Scaler loaded successfully!")
except:
    print("Scaler file not found. Using default scaler.")
    scaler = StandardScaler()
```

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

@app.route('/predict_page')
def predict_page():
    """Render the prediction input page"""
    return render_template('index1_new.html')

@app.route('/predict', methods=['POST'])
def predict():
    """Handle prediction request"""
    try:
        if model is None:
            return render_template('result.html',
                                   prediction_text="Model not loaded. Please train the model first.")

        # Get form data
        form_data = request.form

        # Create feature array
        features = []
```

Here we will be using a declared constructor to route to the HTML page which we have created earlier.

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

Retrieves the value from UI:

```python
# Process each feature with validation
try:
    age = float(form_data.get('age', 50))
    if not (0 < age < 120):
        return render_template('result.html', prediction_text="Error: Age must be between 0 and 120")
    features.append(age)

    bp = float(form_data.get('blood_pressure', 80))
    if not (40 < bp < 200):
        return render_template('result.html', prediction_text="Error: Blood pressure must be between 40 a
    features.append(bp)

    sg = float(form_data.get('specific_gravity', 1.020))
    if not (1.000 < sg < 1.040):
        return render_template('result.html', prediction_text="Error: Specific gravity must be between 1.
    features.append(sg)

    albumin = float(form_data.get('albumin', 0))
    if not (0 <= albumin <= 5):
        return render_template('result.html', prediction_text="Error: Albumin must be between 0 and 5")
    features.append(albumin)

    sugar = float(form_data.get('sugar', 0))
    if not (0 <= sugar <= 5):
        return render_template('result.html', prediction_text="Error: Sugar must be between 0 and 5")
    features.append(sugar)
except ValueError as e:
    return render_template('result.html', prediction_text=f"Error: Invalid input values - {str(e)}")
```

Here we are routing our app to predict() function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will be rendered to the text that we have mentioned in the submit.html page earlier.

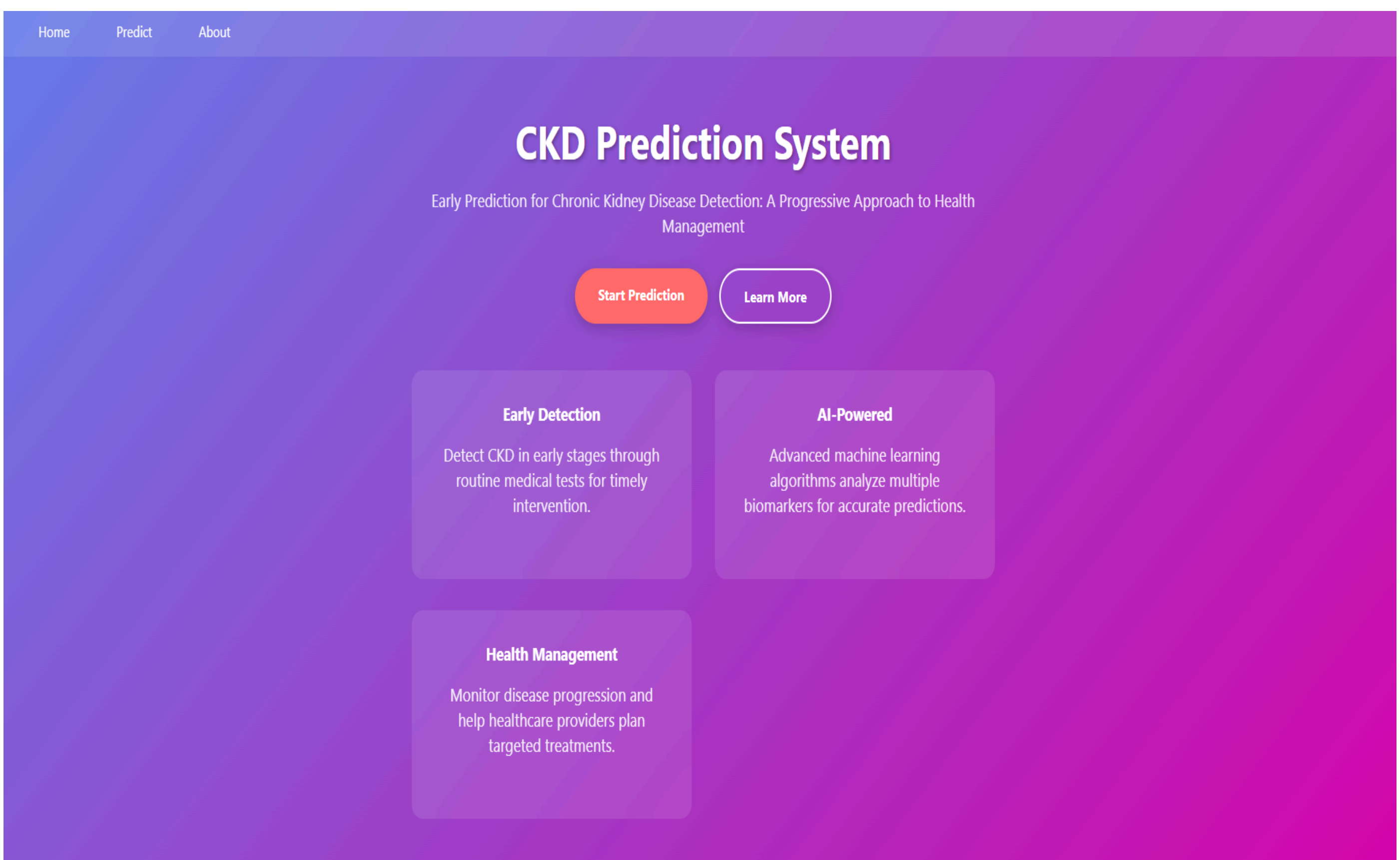
```python
        # Determine result based on both probability and risk factors
        if prob_ckd > 60 or (prob_ckd > 40 and risk_factors >= 3):
            result = "Chronic Kidney Disease Detected"
            risk_level = "High Risk"
            recommendation = "Please consult with a nephrologist immediately for proper diagnosis and treatment."
            result_class = "positive-result"  # For red styling
            congrats_message = ""
        else:
            result = "No Chronic Kidney Disease Found"
            risk_level = "Low Risk"
            if risk_factors >= 2:
                recommendation = "While CKD was not detected, you have some risk factors. Regular check-ups are recommended."
            else:
                recommendation = "Continue maintaining a healthy lifestyle and regular check-ups."
            result_class = "negative-result"  # For green styling
            congrats_message = "Great job! Keep up the healthy habits!"

        # Prepare probability text
        prob_text = ""
        if prediction_proba is not None:
            prob_ckd = prediction_proba[0][1] * 100
            prob_text = f"Probability of CKD: {prob_ckd:.1f}%"

        return render_template('result.html',
                               prediction_text=result,
                               risk_level=risk_level,
                               recommendation=recommendation,
                               probability=prob_text,
                               result_class=result_class,
                               congrats_message=congrats_message)

    except Exception as e:
        error_message = f"Error in prediction: {str(e)}"
        return render_template('result.html', prediction_text=error_message)

@app.route('/about')
def about():
    """Render about page"""
    return render_template('indexnew.html')

if __name__ == '__main__':
    app.run(debug=True, host='127.0.0.1', port=5000)
```

## Activity 2.3: Run the web application

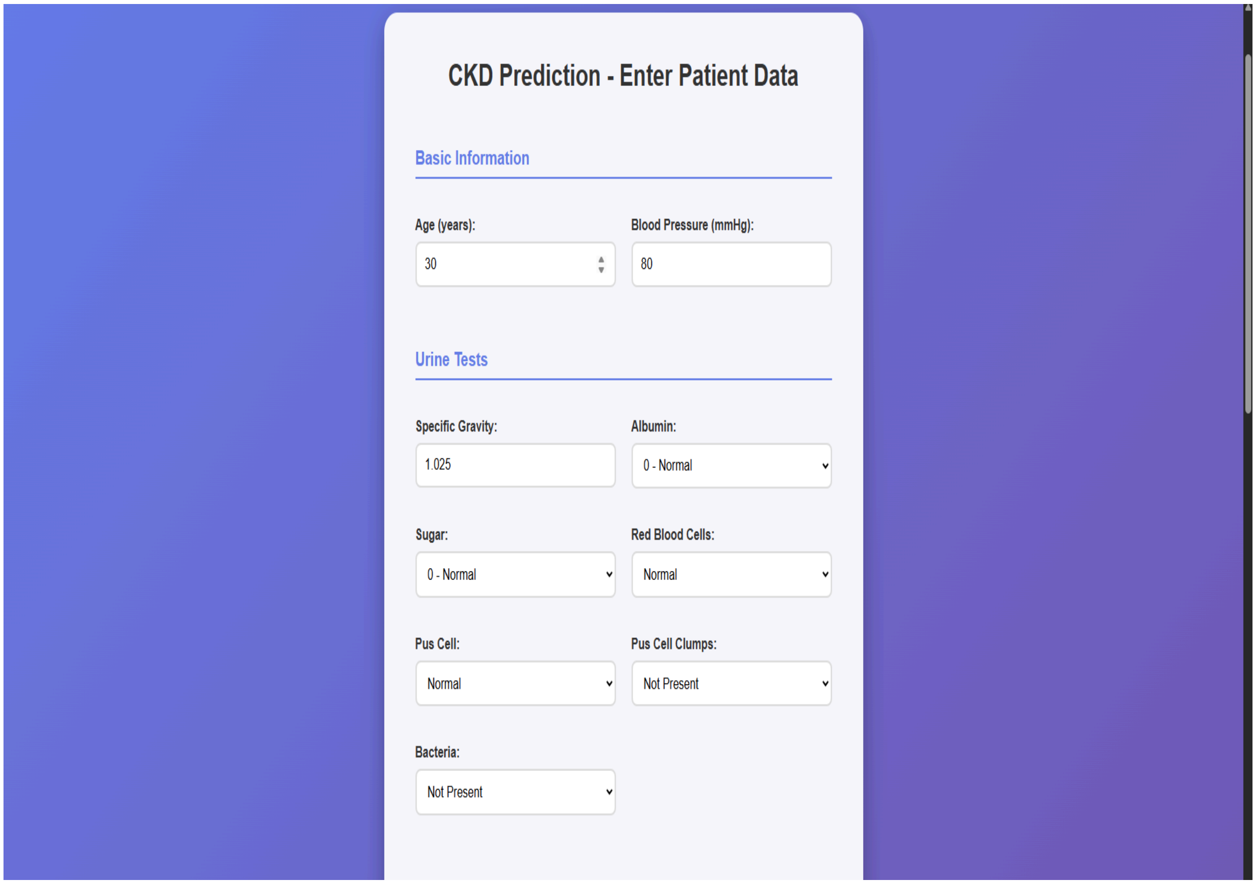
- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top left corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

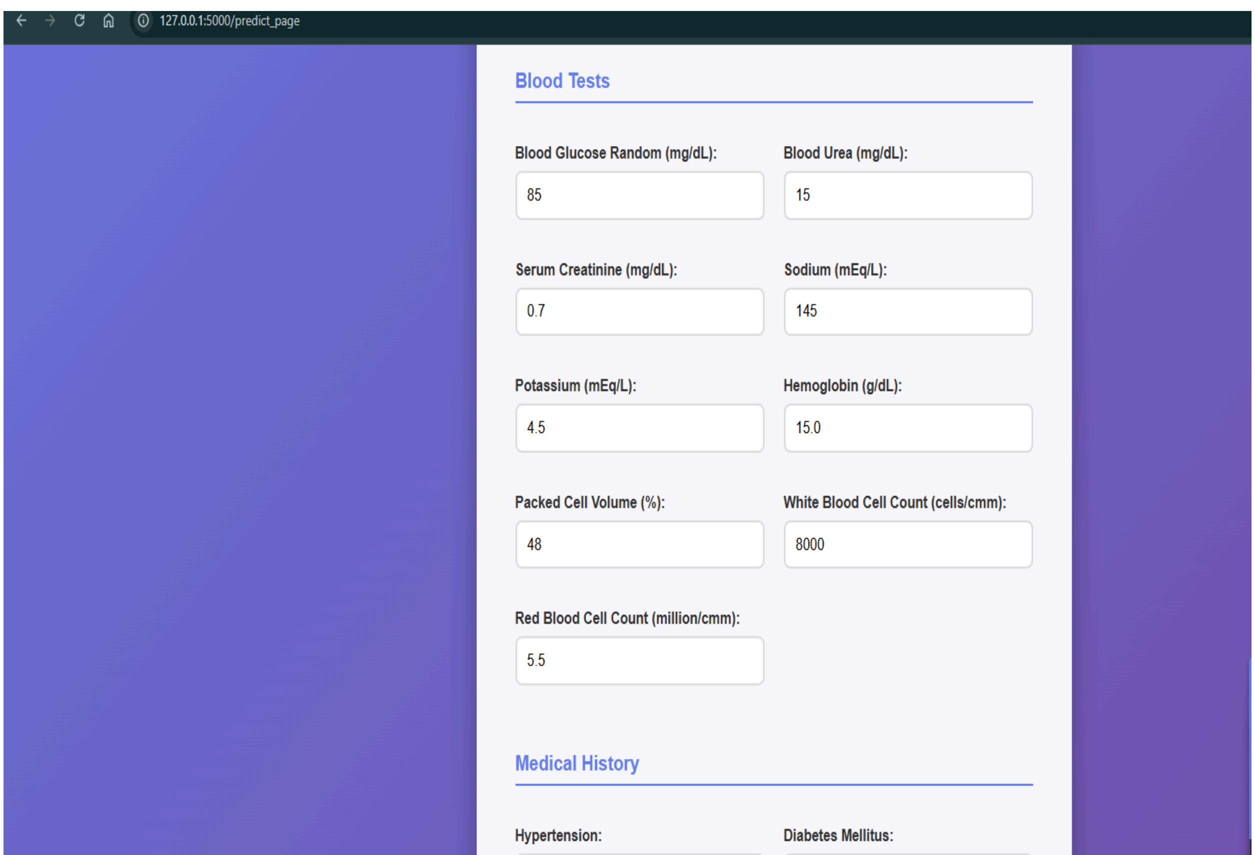
```
PS C:\ML Projects\Earlychronic_kidney project>  & 'c:\Python3.13\python.exe' 'c:\Users\mohit\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\li
8' '--' 'C:\ML Projects\Earlychronic_kidney project\CKD_Prediction_Project\app.py'
Model loaded successfully!
Scaler loaded successfully!
 * 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)
Model loaded successfully!
Scaler loaded successfully!
 * Debugger is active!
 * Debugger PIN: 104-649-460
127.0.0.1 - - [30/Jul/2025 17:24:03] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [30/Jul/2025 17:24:04] "GET /predict_page HTTP/1.1" 200 -
127.0.0.1 - - [30/Jul/2025 17:24:04] "GET /static/style.css HTTP/1.1" 304 -
127.0.0.1 - - [30/Jul/2025 17:24:07] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [30/Jul/2025 17:24:08] "GET /predict_page HTTP/1.1" 200 -
127.0.0.1 - - [30/Jul/2025 17:24:08] "GET /static/style.css HTTP/1.1" 304 -
127.0.0.1 - - [30/Jul/2025 17:24:16] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [30/Jul/2025 17:24:18] "GET /predict_page HTTP/1.1" 200 -
127.0.0.1 - - [30/Jul/2025 17:24:18] "GET /static/style.css HTTP/1.1" 304 -
127.0.0.1 - - [30/Jul/2025 17:24:22] "POST /predict HTTP/1.1" 200 -
```

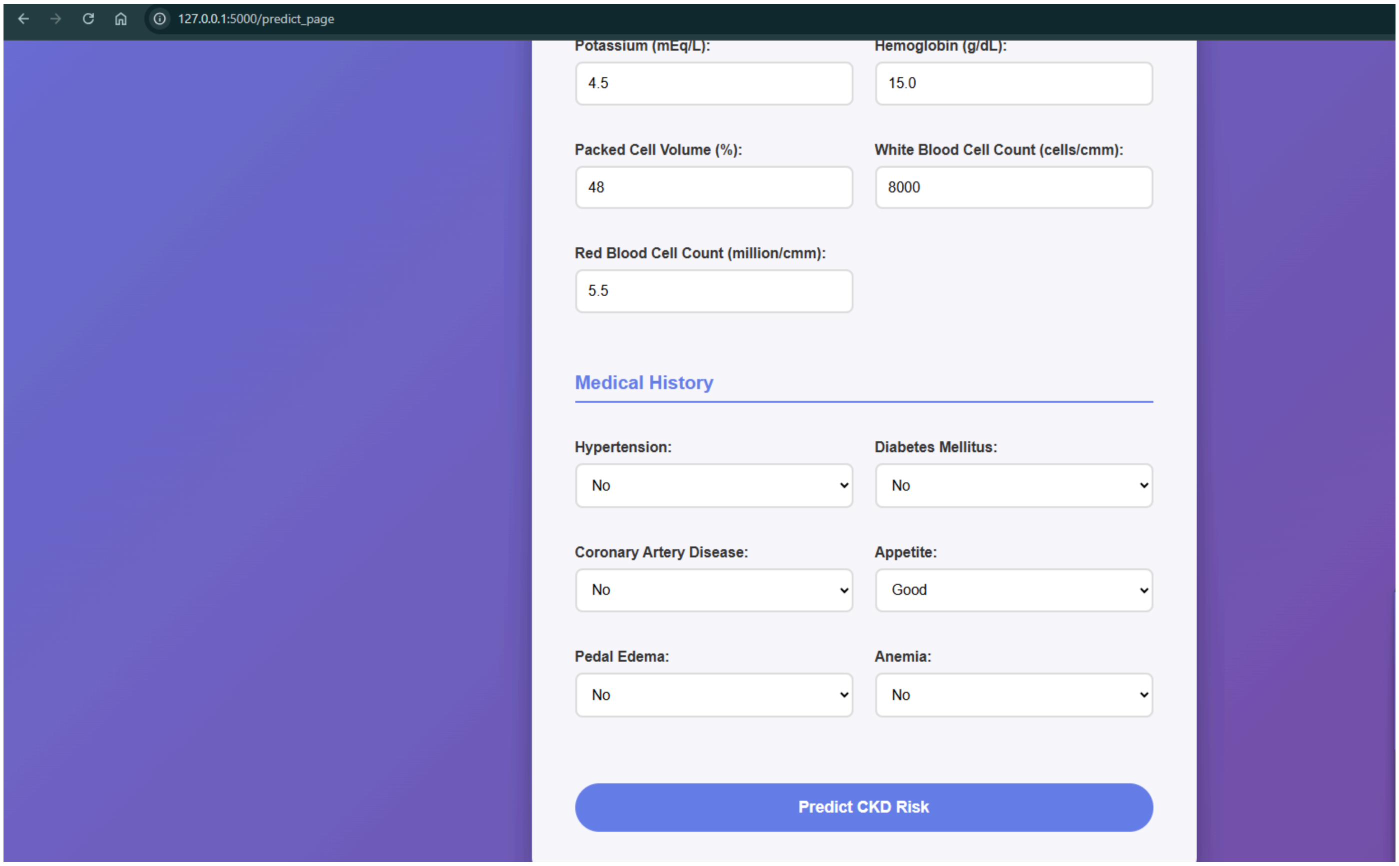
**Now,Go the web browser and write the localhost url (http://127.0.0.1:5000) to get the below result**

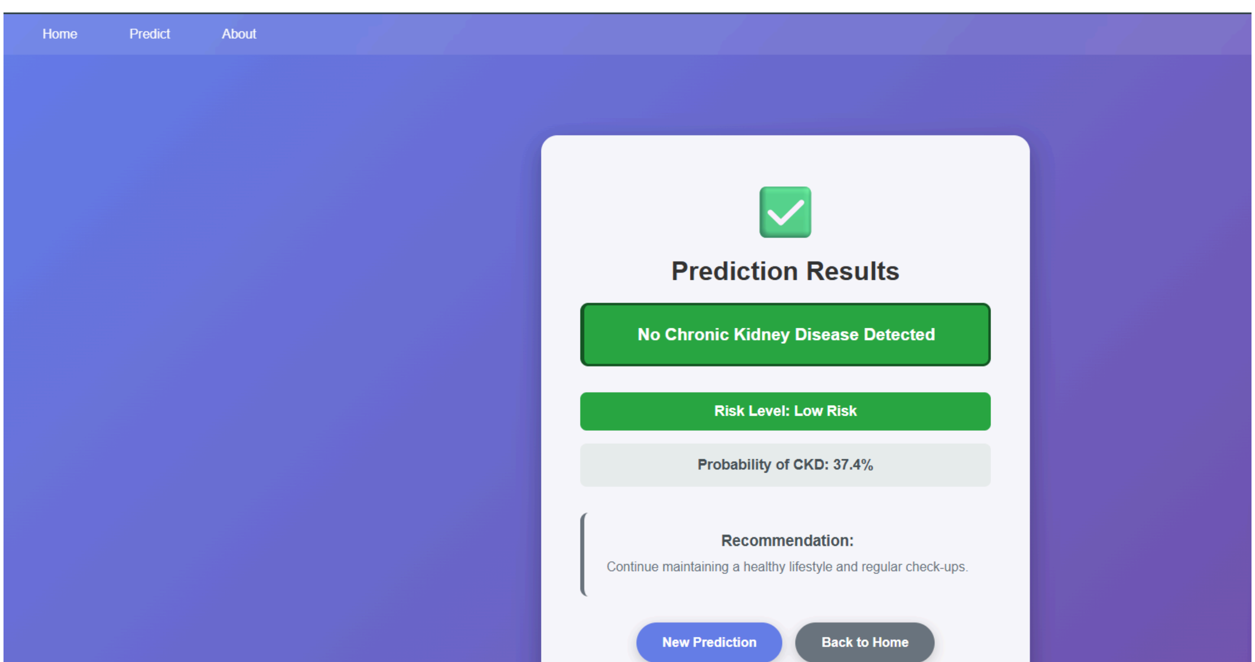


## The NoCKD(NON chronic kidney diseas Detacted)

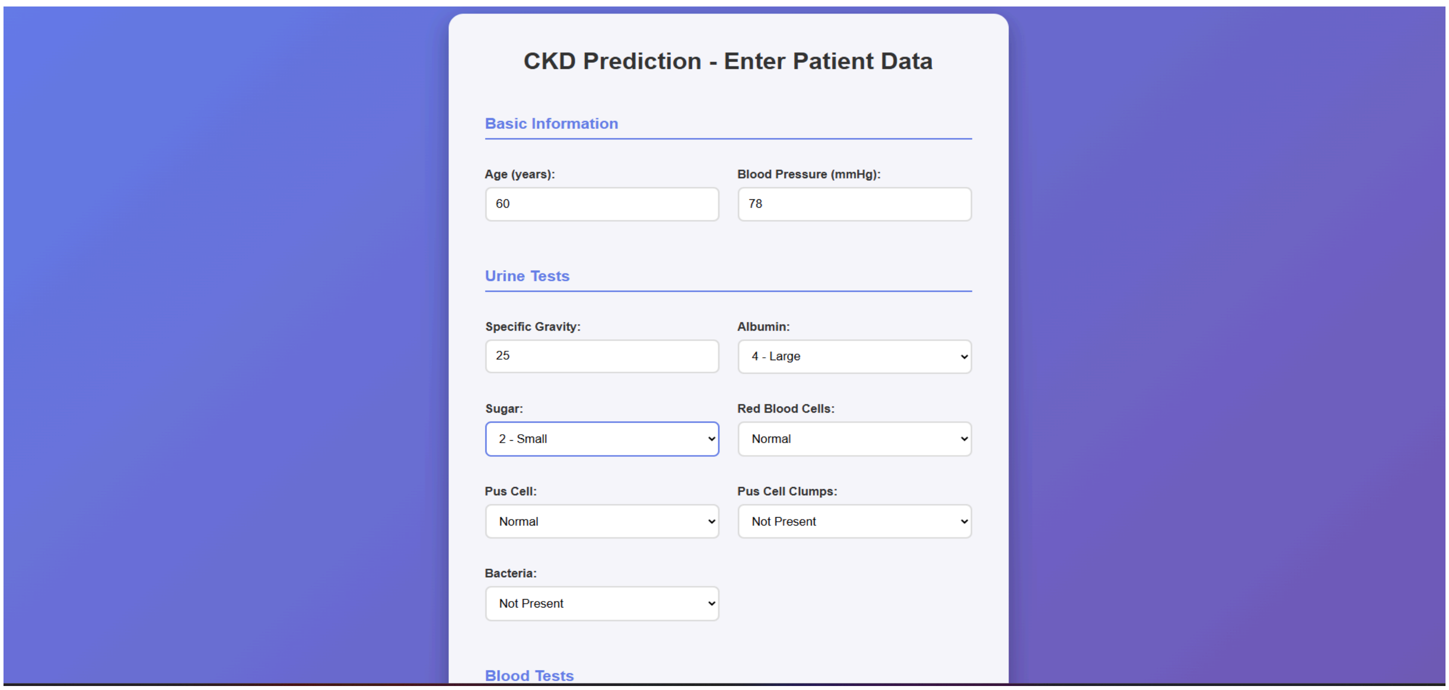


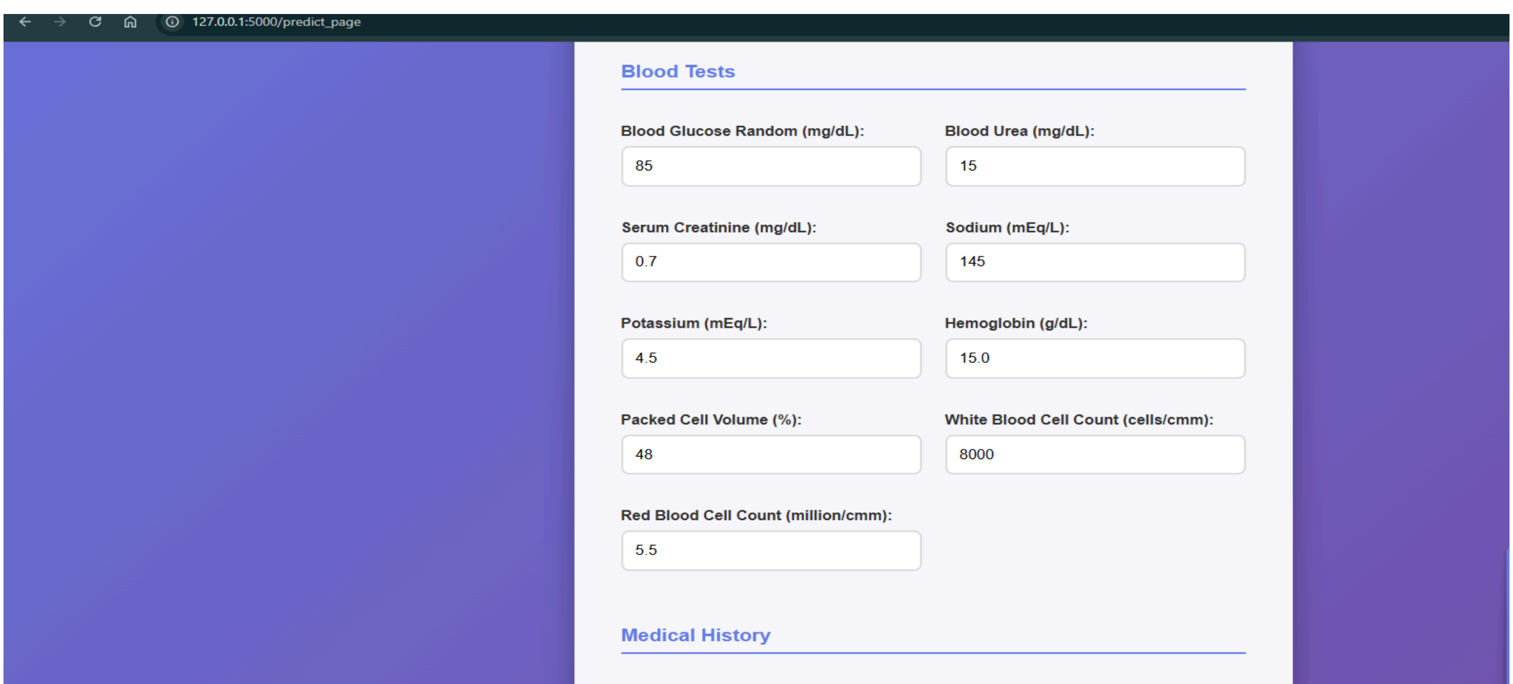



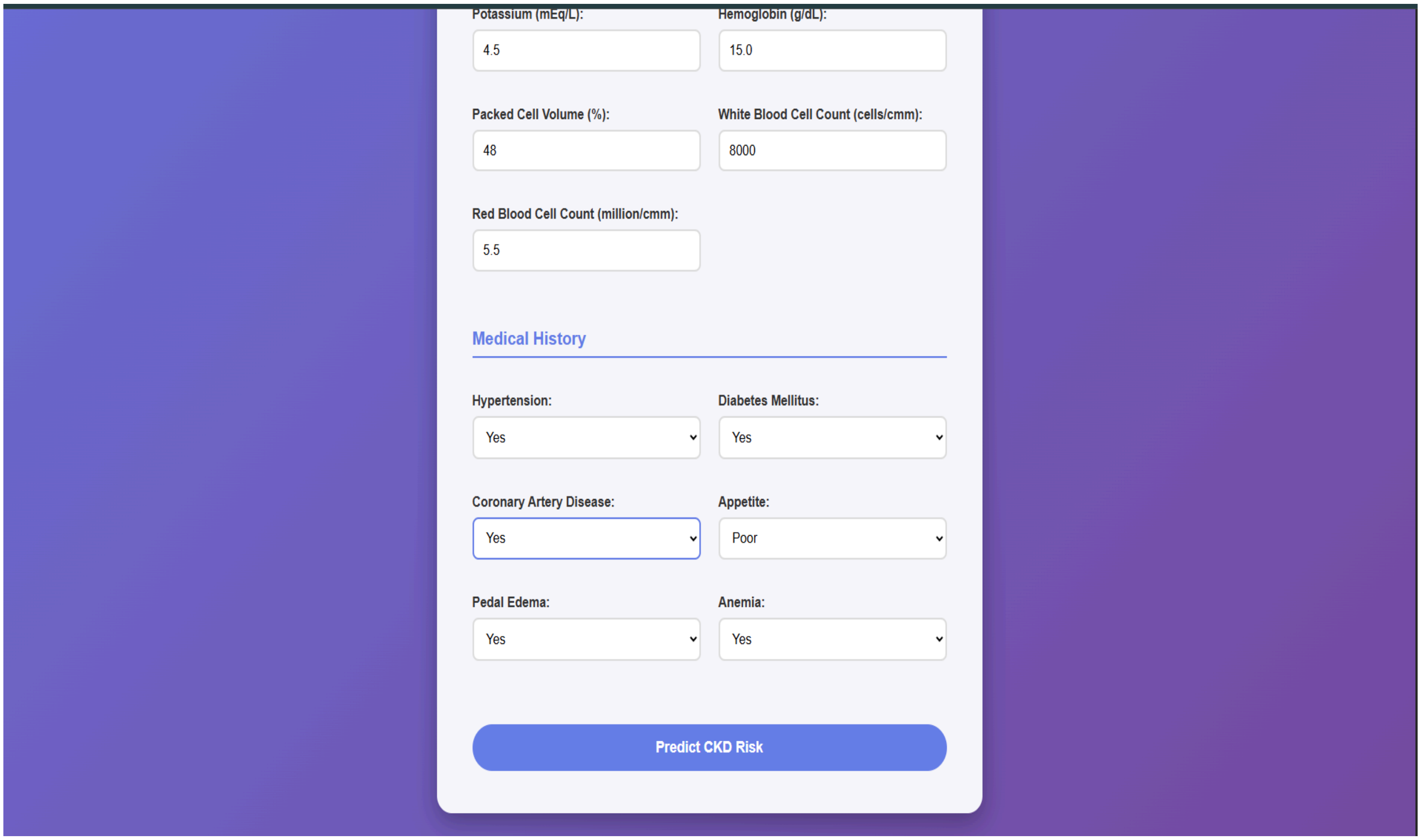

127.0.0.1:5000/predict_page
Potassium (mEq/L):
4.5
Hemoglobin (g/dL):
15.0
Packed Cell Volume (%):
48
White Blood Cell Count (cells/cmm):
8000
Red Blood Cell Count (million/cmm):
5.5
Medical History
Hypertension:
No
Diabetes Mellitus:
No
Coronary Artery Disease:
No
Appetite:
Good
Pedal Edema:
No
Anemia:
No
Predict CKD Risk

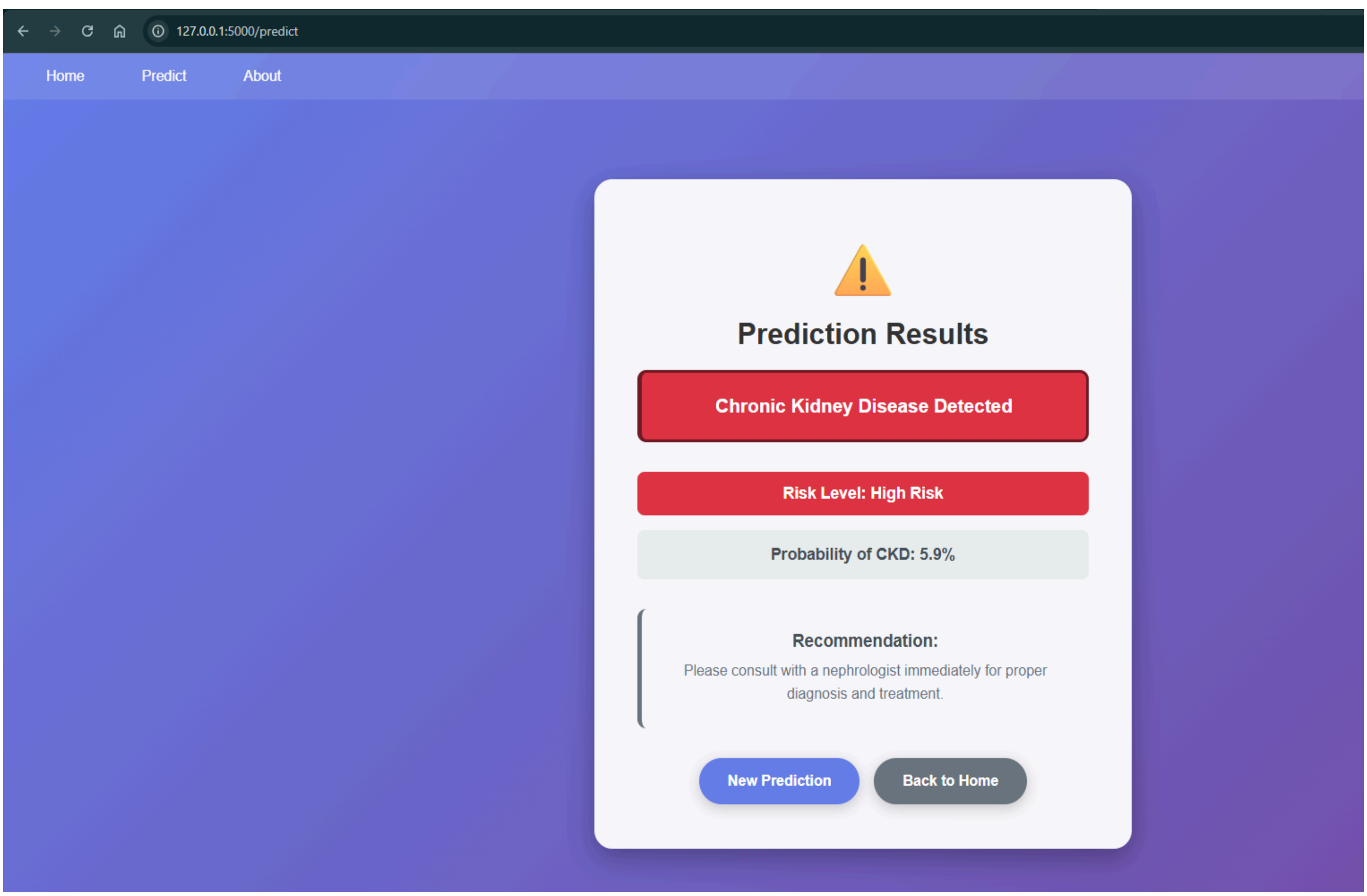

Home
Predict
About
Prediction Results
No Chronic Kidney Disease Detected
Risk Level: Low Risk
Probability of CKD: 37.4%
Recommendation:
Continue maintaining a healthy lifestyle and regular check-ups.
New Prediction
Back to Home

# CKD( cronic Kidney Diseas Detacted)




Potassium (mEq/L):
4.5
Hemoglobin (g/dL):
15.0
Packed Cell Volume (%):
48
White Blood Cell Count (cells/cmm):
8000
Red Blood Cell Count (million/cmm):
5.5
Medical History
Hypertension:
Yes
Diabetes Mellitus:
Yes
Coronary Artery Disease:
Yes
Appetite:
Poor
Pedal Edema:
Yes
Anemia:
Yes
Predict CKD Risk


127.0.0.1:5000/predict
Home
Predict
About
Prediction Results
Chronic Kidney Disease Detected
Risk Level: High Risk
Probability of CKD: 5.9%
Recommendation:
Please consult with a nephrologist immediately for proper diagnosis and treatment.
New Prediction
Back to Home

# Thank You