Chronic Kidney Disease Prediction Using Machine Learning

August 9, 2025

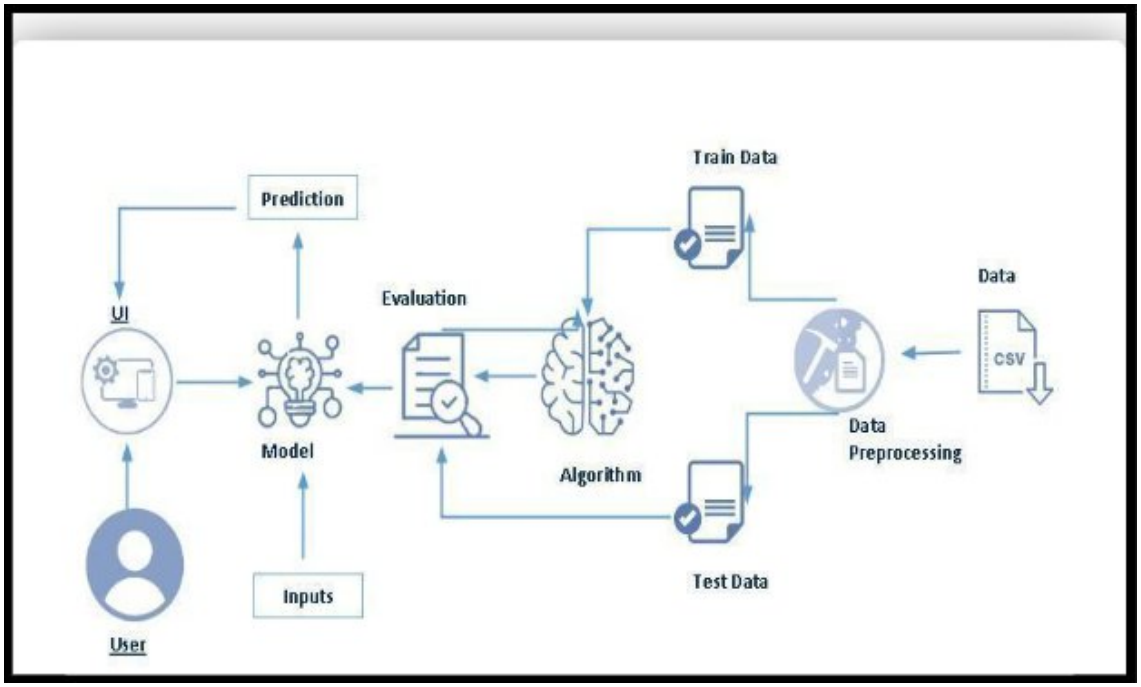
made for Smart bridge

Chronic Kidney Disease Prediction Using Machine Learning:

Chronic Kidney Disease (CKD) is a condition where the kidneys are progressively damaged, eventually leading to a decline in function and kidney failure.

However, CKD can be detected in its early stages through the use of specific biomarkers. Early detection is crucial, as it allows for interventions that can help slow the progression of the disease and reduce the risk of permanent kidney failure.

Technical Architecture:



Project Flow:

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

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

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

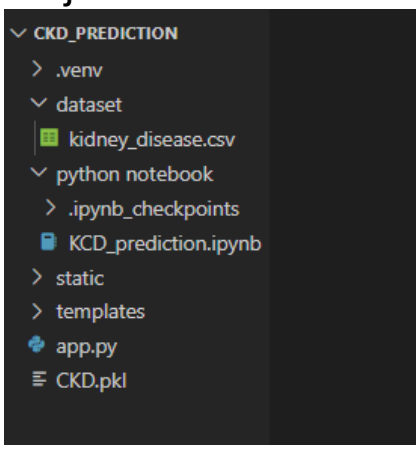
Prior Knowledge:

ML Concepts

- Supervised learning: https://www.javatpoint.com/supervised-machine-learning
- Unsupervised learning: https://www.javatpoint.com/unsupervised-machine-learning
- Decision tree: https://www.javatpoint.com/machine-learning-decision-tree-classification- algorithm/
- Evaluation metrics: https://www.analyticsvidhya.com/blog/2019/08/11-important-model- evaluation-error-metrics/
- Pandas basics: https://youtube.com/playlist?list=PLCC34OHNcOtqSz7Ke7kaYRf9CfviJgO55&feature=shared
- numpy basics: https://youtube.com/playlist?list=PLCC34OHNcOtpalASMlX2HHdsLNipyyhbK&feature=shared

- Flask basics: https://youtu.be/oA8brF3w5XQ?feature=shared
- ANN basics: https://youtu.be/0B5eIE_1vpU?feature=shared
- Sklearn: https://youtube.com/playlist?list=PLxCzCOWd7aiHQ9Qyo70JCTkTxczPnVC6s&feature=shared

Project Structure:



- 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

Activity 1: Specify the business problem:

Refer Project Description

Activity 2: Business requirements:

The business requirements for a machine learning model designed to predict chronic kidney disease (CKD) are focused on several key areas. First, the model must accurately predict a CKD diagnosis using the provided data. It is also crucial to minimize the number of false positives (incorrectly diagnosing a healthy patient) and false negatives (failing to identify a patient with the disease). Finally, the model must be able to provide a clear explanation for its predictions to ensure compliance with regulations and enhance transparency.

Activity3: Literature Survey:

Chronic kidney disease (CKD) is a significant public health issue, affecting an estimated 14% of the global population. The disease is characterized by a gradual loss of kidney function over time, which can lead to a range of serious health complications, including end-stage renal disease (ESRD) that requires dialysis or a kidney transplant. Therefore, early detection and management of CKD are crucial to prevent its progression to ESRD and improve patient outcomes.

In recent years, numerous studies have focused on developing accurate and efficient methods for predicting CKD progression, employing a variety of techniques such as machine learning, deep learning, and artificial neural networks.

Activity 4: Social or Business Impact.

From a societal perspective, early detection and prediction of CKD can significantly improve patient outcomes and quality of life. By identifying at-risk individuals, healthcare providers can intervene promptly to slow the disease's progression through lifestyle modifications, medication, and other treatments. This proactive approach can help prevent the need for costly and life-altering interventions such as dialysis or kidney transplantation. Furthermore, early prediction helps reduce the overall burden of CKD on the healthcare system by decreasing hospitalizations and emergency room visits.

Milestone 2: Data Collection & Preparation:

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

Activity 1: Collect the dataset:

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc. In this project we have used .csv data.

This data is downloaded from kaggle.com.

Please refer to the link given below to download the dataset.

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

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

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

Activity 1.1: Importing the libraries:

Import the necessary libraries as shown in the image.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf
import pickle
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from collections import Counter
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,classification_report
from sklearn import model_selection
from sklearn.metrics import confusion_matrix
```

Activity 1.2: Read the Dataset:

- In this project our dataset format is .csv. 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.

```
data=pd.DataFrame(pd.read_csv('kidney_disease.csv'))
```

```
data
```

	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

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.

●Handling missing values

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

Activity 2.1: 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 null values are present in every column accept column named class in our dataset.

●We have to fill those null values.

-
```
[6]: data.isnull().any()

[6]: age                          True
     blood_pressure               True
     specific_gravity             True
     albumin                      True
     sugar                        True
     red_blood_cells              True
     pus_cell                     True
     pus_cell_clumps              True
     bacteria                     True
     blood_glucose_random         True
     blood_urea                   True
     serum_creatinine             True
     sodium                       True
     potassium                    True
     hemoglobin                   True
     packed_cell_volume           True
     white_blood_cell_count       True
     red_blood_cell_count         True
     hypertension                 True
     diabetesmellitus             True
     coronary_artery_disease      True
     appetite                     True
     pedal_edema                  True
     anemia                       True
     class                       False
     dtype: bool
```
For the float type data we'll going to use .fillna() function to fill the null cell with mean() value of that column. As you can see in image provided below.

```
: 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['hemoglobin'].fillna(data['hemoglobin'].mean(),inplace=True)
  data['potassium'].fillna(data['potassium'].mean(),inplace=True)
  data['serum_creatinine'].fillna(data['serum_creatinine'].mean(),inplace=True)
  data['sodium'].fillna(data['sodium'].mean(),inplace=True)
```

- For object type data we'll going to use .fillna() function to fill the null cell with mode() value of the column after replacing random values with null values.

```
data['red_blood_cell_count']=data['red_blood_cell_count'].replace('\t?',np.nan)

data['white_blood_cell_count']=data['white_blood_cell_count'].replace('\t?',np.nan)
data['white_blood_cell_count']=data['white_blood_cell_count'].replace('\t6200','6200')
data['white_blood_cell_count']=data['white_blood_cell_count'].replace('\t8400',np.nan)

data['packed_cell_volume']=data['packed_cell_volume'].replace('\t?',np.nan)
data['packed_cell_volume']=data['packed_cell_volume'].replace('\t43',np.nan)



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['specific_gravity'].fillna(data['specific_gravity'].mode()[0],inplace=True)
data['red_blood_cell_count'].fillna(data['red_blood_cell_count'].mode()[0],inplace=True)
data['white_blood_cell_count'].fillna(data['white_blood_cell_count'].mode()[0],inplace=True)
data['packed_cell_volume'].fillna(data['packed_cell_volume'].mode()[0],inplace=True)
```

●After that there will be zero null values.

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

```
age                       0
blood_pressure            0
specific_gravity          0
albumin                   0
sugar                     0
red_blood_cells           0
pus_cell                  0
pus_cell_clumps           0
bacteria                  0
blood_glucose_random      0
blood_urea                0
serum_creatinine          0
sodium                    0
potassium                 0
hemoglobin                0
packed_cell_volume        0
white_blood_cell_count    0
red_blood_cell_count      0
hypertension              0
diabetesmellitus          0
coronary_artery_disease   0
appetite                  0
pedal_edema               0
anemia                    0
class                     0
dtype: int64
```

Milestone 3: Exploratory Data Analysis:

Activity 1: Descriptive statistical:

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.

data.describe()

	age	blood pressure	specific gravity	albumin	sugar	red blood cells	pus cell	pus cell clumps	bacteria	blood glucose random	...	sodi
count	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	...	400.000
mean	51.675000	76.469072	1.017712	0.900000	0.395000	0.882500	0.810000	0.105000	0.055000	148.036517	...	137.528
std	17.022000	13.476298	0.005714	1.31313	1.040038	0.322418	0.392792	0.306937	0.228366	74.782634	...	9.204
min	2.000000	50.000000	1.005000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	22.000000	...	4.500
25%	42.000000	70.000000	1.015000	0.000000	0.000000	1.000000	1.000000	0.000000	0.000000	101.000000	...	135.000
50%	55.000000	78.734536	1.020000	0.000000	0.000000	1.000000	1.000000	0.000000	0.000000	126.000000	...	137.528
75%	64.000000	80.000000	1.020000	2.000000	0.000000	1.000000	1.000000	0.000000	0.000000	150.000000	...	141.000
max	90.000000	180.000000	1.025000	5.000000	5.000000	1.000000	1.000000	1.000000	1.000000	490.000000	...	163.000

8 rows × 22 columns

Activity 2: Visual analysis:

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

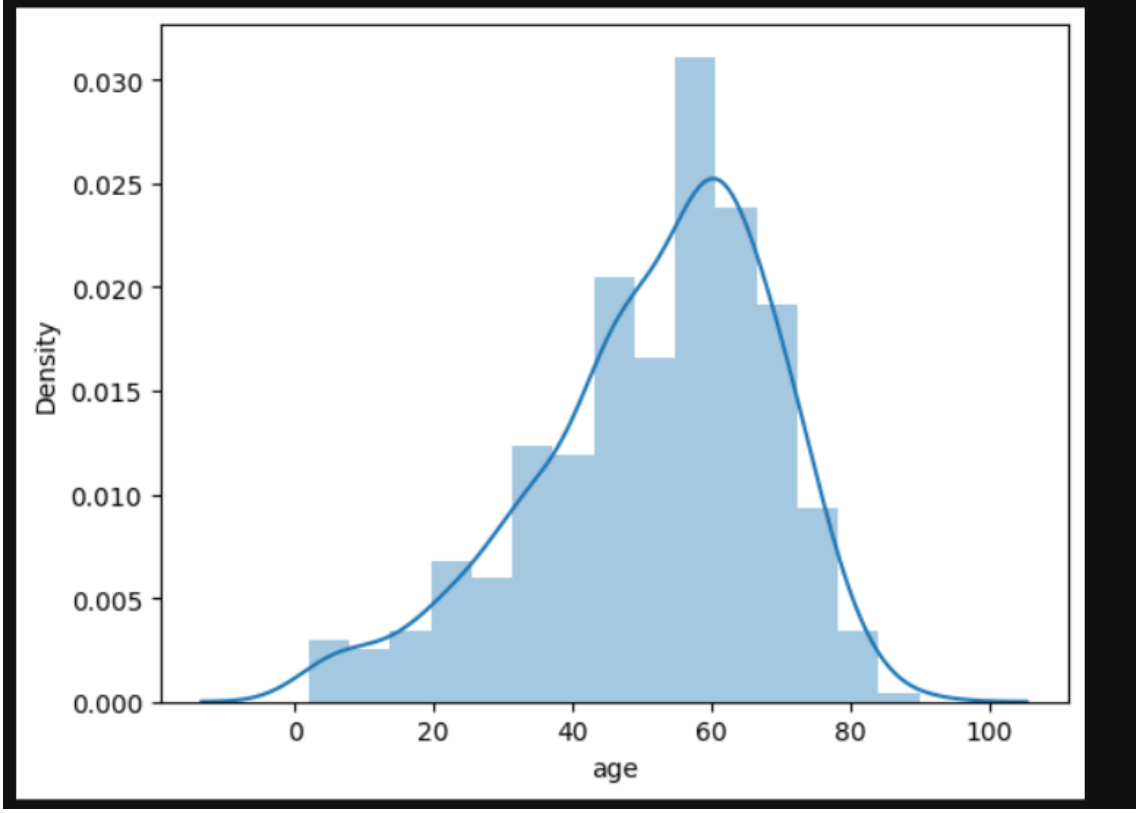
Activity 2.1: Univariate analysis:

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

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

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.distplot(data.age)
```

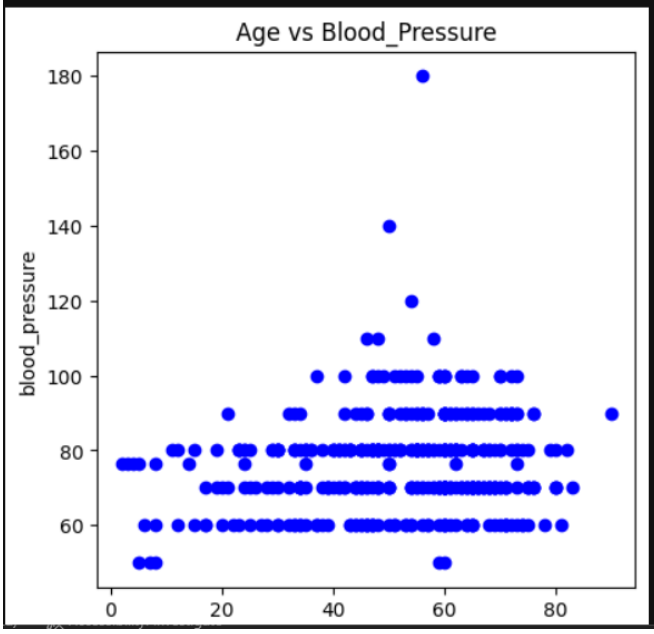
```
<Axes: xlabel='age', ylabel='Density'>
```



Activity 2.2: Bivariate analysis:

```
plt.figure(figsize=(5,5))
plt.scatter(x=data.age,y=data.blood_pressure,color='blue')
plt.xlabel('age')
plt.ylabel('blood_pressure')
plt.title('Age vs Blood_Pressure')
```

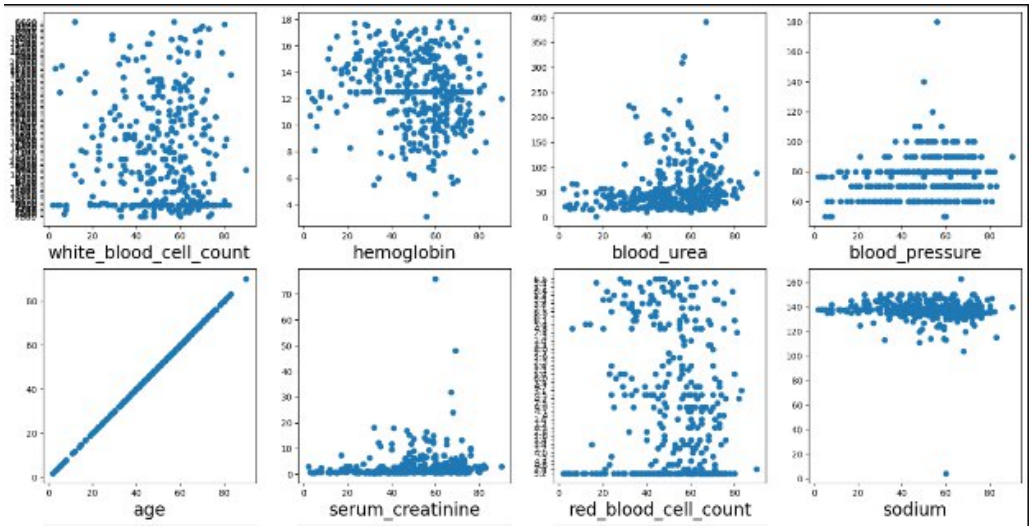
```
Text(0.5, 1.0, 'Age vs Blood_Pressure')
```

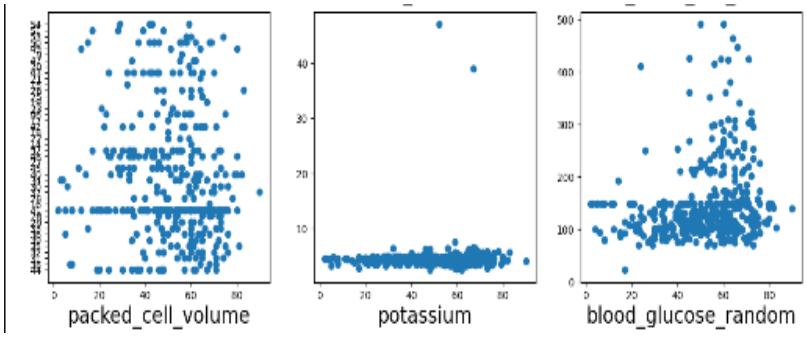


Activity 2.3: Multivariate analysis:

Age vs all continuous columns

```
plt.figure(figsize=(20,15),facecolor='white')
plot_num=1

for col in contcols:
    if plot_num<=11:
        ax=plt.subplot(3,4,plot_num)
        plt.scatter(data['age'],data[col])
        plt.xlabel(col,fontsize=20)
    plot_num+=1
plt.show()
```





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

Finding correlation between the independent Columns:

Correlation is a statistical relationship between two variables and it could be positive, meaning both variables move in the same direction, or negative, meaning that when one variable's value increases, the other variables' values decrease.

With the help of seaborn heatmap we will be plotting the heatmap and for finding

the correlation between variable we have corr() available.

```
f,ax=plt.subplots(figsize=(18,10))
sns.heatmap(data.corr(),annot=True,fmt=".2f",ax=ax,linewidths=0.5,linecolor='orange')
plt.xticks(rotation=45)
plt.yticks(rotation=45)
plt.show()
```

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

And correlation plays a very important role for extracting the correct features for build our model.

Encoding the Categorical Features:

• The categorical Features are can't be passed directly to the Machine Learning Model. So we convert them into Numerical data based on their order. This Technique is called Encoding.

• Here we are importing Label Encoder from the Sklearn Library.

•Here we are applying fit_transform to transform the categorical features to numerical features.

```
for i in catcols:
    print('Label for:',i)
    le=LabelEncoder()
    print('before encoding:',Counter(data[i]))
    data[i]=le.fit_transform(data[i])
    print('after encoding:',Counter(data[i]))
    print('*'*120+'\n')
```

Splitting data into train and test:

●Now let's split the Dataset into train and test sets.

●First split the dataset into X and y and then split the data set Here X and y variables are created.

●On x variable, df is passed with dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using train_test_split() function from sklearn.

●As parameters, we are passing x, y, test_size, random_state

```
selcols=['red_blood_cells','pus_cell','blood_glucose_random','blood_urea','pedal_edema','anemia','diabetesmellitus','coronary_artery_disease']

X=pd.DataFrame(data,columns=selcols)
y=data['class']
```

```
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)
```

Scaling :

- Scaling is a technique used to transform the values of a dataset to a similar scale to improve the performance of machine learning algorithms. Scaling is important because many machine learning algorithms are sensitive to the scale of the input features.
- Here we are using Standard Scaler.

```
X_train_scaled=sc.fit_transform(X_train)
X_test_scaled=sc.transform(X_test)
```

Milestone 4: Model Building:

Activity 1: Training the model in multiple algorithms:

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

Activity 1.1:ANN model:

- First Sequential and Dense are imported from tensorflow.keras.models then Sequential is initialised and input layer,output layer and hidden layers are added with activation functions,then training data is passed to the model with the .fit() function.
- After saving the model test data is predicted with .predict() function and saved in a new variable.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```
classification=Sequential()
classification.add(Dense(30,activation='relu'))
classification.add(Dense(128,activation='relu'))
classification.add(Dense(64,activation='relu'))
classification.add(Dense(32,activation='relu'))
classification.add(Dense(1,activation='sigmoid'))
```

```
classification.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

```
classification.fit(X_train,y_train,batch_size=10,validation_split=0.2,epochs=100)
```

```
classification.save('ckd.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via
legacy. We recommend using instead the native Keras format
keras')`.
```

```
y_pred=classification.predict(X_test)
print(y_pred)
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
[[3.12661141e-01]
 [9.00769413e-01]
 [1.26431976e-03]
 [9.78700593e-11]
 [1.97617100e-09]
```

Activity 1.2: RandomForestClassifier model:

First Random Forest Model is imported from sklearn Library then RandomForestClassifier 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.class_weight is passed balanced to avoid unbalanced data.

```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier(n_estimators=10,criterion='entropy',random_state=42,class_weight='balanced')
```

```
rfc.fit(X_train,y_train)
```

```
RandomForestClassifier(class_weight='balanced', criterion='entropy',
                       n_estimators=10, random_state=42)
```

```
y_pred=rfc.predict(X_test)
y_pred_train=rfc.predict(X_train)
```

Activity 1.3: Logistic Regression model:

Logistic Regression Model is imported from sklearn Library then Logistic Regression algorithm is initialised and training data is passed to the model with .fit() function.and predictions are made. class_weight is passed balanced to avoid unbalanced data.

```
from sklearn.linear_model import LogisticRegression
```

```
lgr=LogisticRegression(random_state=42,solver='liblinear',class_weight='balanced')
```

```
lgr.fit(X_train,y_train)
```

```
LogisticRegression(class_weight='balanced', random_state=42, solver='liblinear')
```

```
y_pred=lgr.predict(X_test)
y_pred_train=lgr.predict(X_train)
```

Activity 1.3: DecisionTreeClassifier model:

DecisionTreeClassifier Model is imported from sklearn Library then DecisionTreeClassifier algorithm is initialised and training data is passed to the model with .fit() function. class_weight is passed balanced to avoid unbalanced data.

```
from sklearn.tree import DecisionTreeClassifier

dtc=DecisionTreeClassifier(random_state=42,class_weight='balanced')

dtc.fit(X_train,y_train)

DecisionTreeClassifier
DecisionTreeClassifier(class_weight='balanced', random_state=42)

y_pred=dtc.predict(X_test)
y_pred
```

Activity 2: Testing the model:

Here we have tested with Decision Tree algorithm. You can test with all algorithm.
With the help of predict() function.

```
y_pred=dtc.predict(X_test)
y_pred

array([1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0,
       1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0])
```

```
y_pred=lgr.predict([[1,1,121.000000,36.0,0,0,1,0]])
print(y_pred)

[0]
C:\Users\JAYNIL\python\panda\.venv\Lib\site-packages\sklearn\utils\validation.
LogisticRegression was fitted with feature names
  warnings.warn(

y_pred=dtc.predict([[1,1,121.000000,36.0,0,0,1,0]])
print(y_pred)

[0]
C:\Users\JAYNIL\python\panda\.venv\Lib\site-packages\sklearn\utils\validation.
DecisionTreeClassifier was fitted with feature names
  warnings.warn(

y_pred=rfc.predict([[1,1,121.000000,36.0,0,0,1,0]])
print(y_pred)

[0]
```

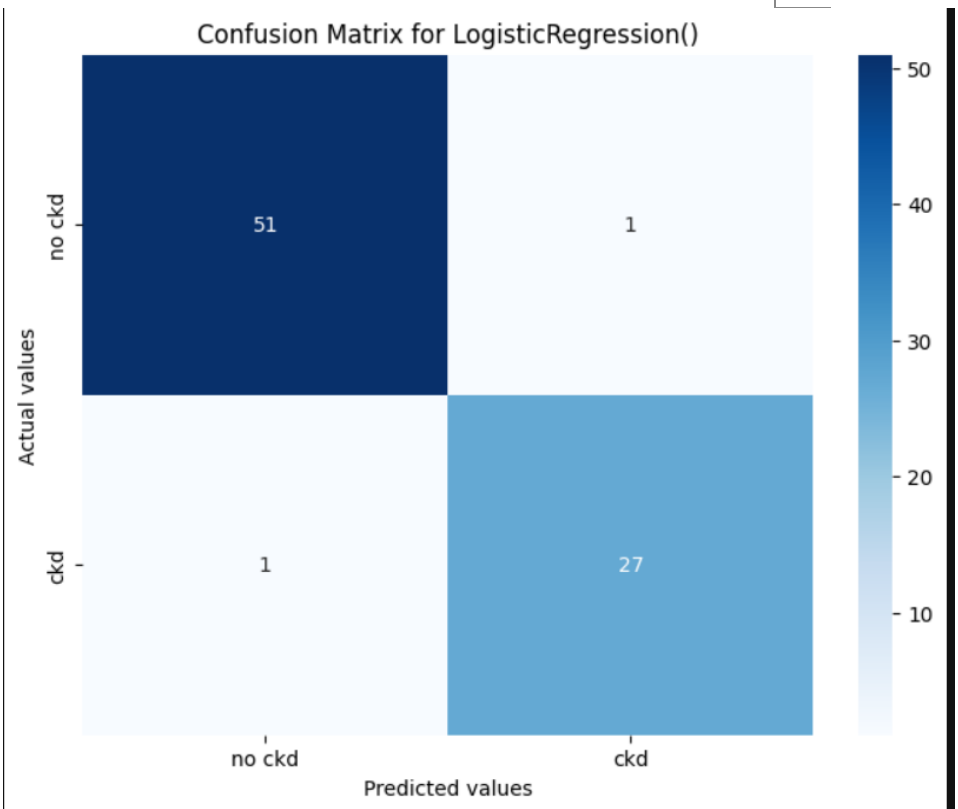
Milestone 5: Performance Testing & Evaluate the results:

Activity 1: Testing model with multiple evaluation metrics:

Multiple evaluation metrics means evaluating the model's performance on a test set using different performance measures. This can provide a more comprehensive understanding of the model's strengths and weaknesses. We are using evaluation metrics for classification tasks including accuracy, precision, recall, support and F1-score.

```
def compare():
    dfs=[]
    models=[('LogReg',LogisticRegression()),('RF',RandomForestClassifier()),('DecisionTree',Decisio
    results=[]
    names=[]
    scoring=['accuracy','precision_weighted','recall_weighted','f1_weighted','roc_auc']
    target_names=['NO CKD','CKD']

    for name,model in models:
        kfold=model_selection.KFold(n_splits=5,shuffle=True,random_state=42)
        cv_results=model_selection.cross_validate(model,X_train,y_train,cv=kfold,scoring=scoring)
        clf=model.fit(X_train,y_train)
        y_pred=clf.predict(X_test)
        print(name)
        print(classification_report(y_test,y_pred,target_names=target_names))
        results.append(cv_results)
        names.append(name)
        this_df=pd.DataFrame(cv_results)
        this_df['model']=name
        dfs.append(this_df)

        cm=confusion_matrix(y_test,y_pred)
        print(cm)
        plt.figure(figsize=(8,6))
        sns.heatmap(cm,cmap='Blues',annot=True,xticklabels=['no ckd','ckd'],yticklabels=['no ckd','
        plt.xlabel('Predicted values')
        plt.ylabel('Actual values')
        plt.title(f'Confusion Matrix for {model}')
        plt.show()
    final=pd.concat(dfs,ignore_index=True)
    return final
final=compare()
```

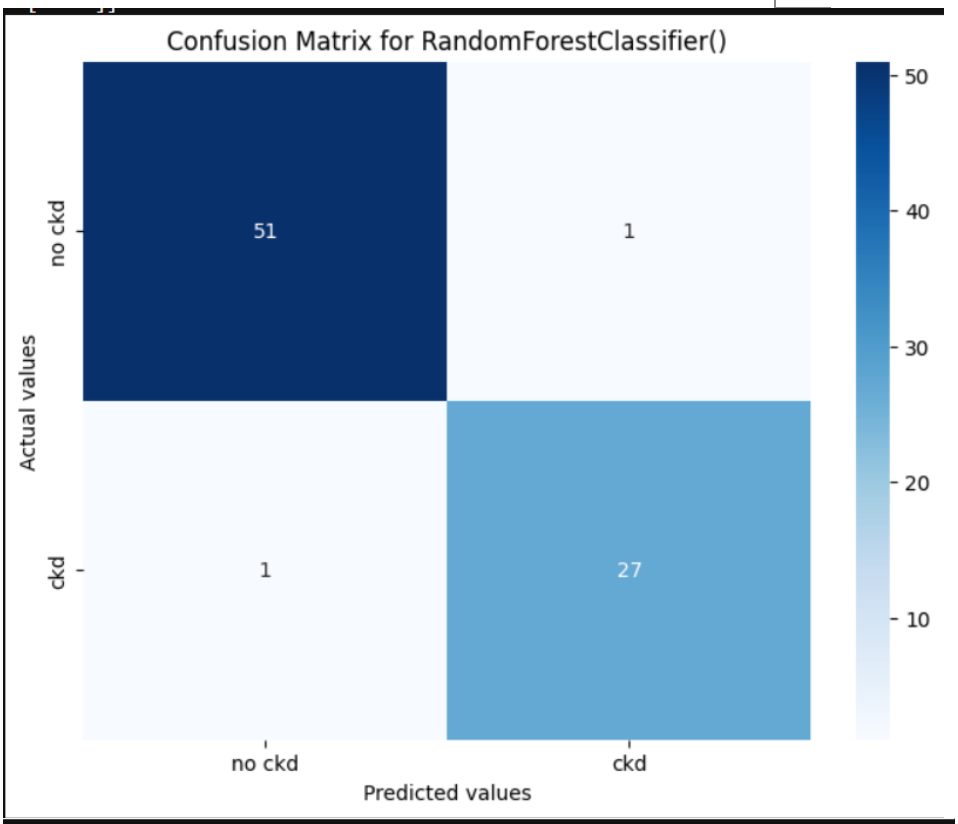
```
LogReg
              precision    recall  f1-score   support

      NO CKD       0.98      0.98      0.98        52
         CKD       0.96      0.96      0.96        28

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

[[51  1]
 [ 1 27]]
```



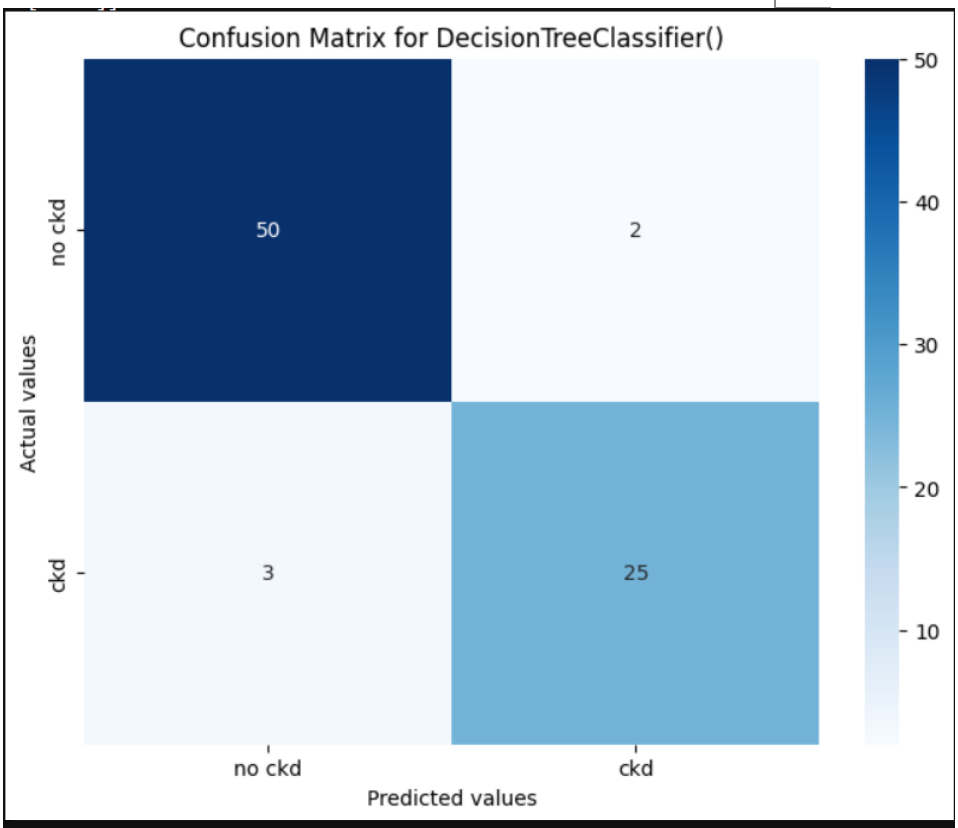
RF

	precision	recall	f1-score	support
NO CKD	0.98	0.98	0.98	52
CKD	0.96	0.96	0.96	28
accuracy			0.97	80
macro avg	0.97	0.97	0.97	80
weighted avg	0.97	0.97	0.97	80

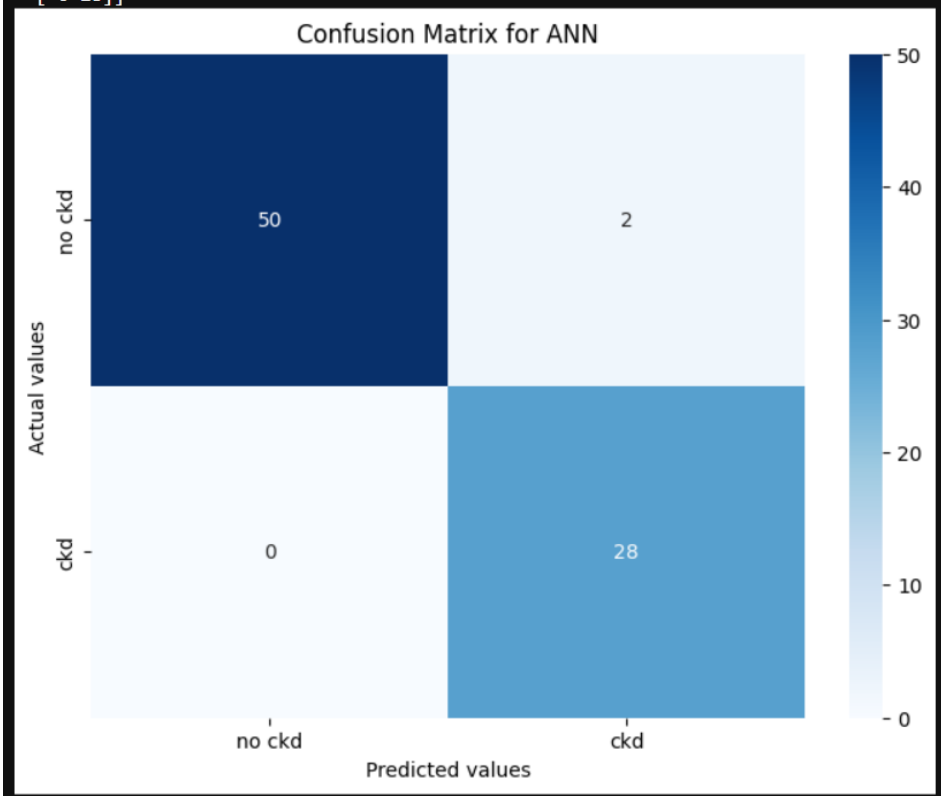
```
[[51  1]
 [ 1 27]]
```




```
DecisionTree
              precision    recall  f1-score   support

      NO CKD       0.94      0.96      0.95        52
         CKD       0.93      0.89      0.91        28

    accuracy                           0.94        80
   macro avg       0.93      0.93      0.93        80
weighted avg       0.94      0.94      0.94        80

[[50  2]
 [ 3 25]]
```


Confusion Matrix for DecisionTreeClassifier()
Actual values
no ckd
ckd
50
2
3
25
no ckd
ckd
Predicted values
50
40
30
20
10

```
[[50  2]
 [ 0 28]]
```


Confusion Matrix for ANN
Actual values
no ckd
ckd
50
2
0
28
no ckd
ckd
Predicted values
50
40
30
20
10
0

Activity 2: Evaluate the results

```
bootstraps=[]
for model in list(set(final.model.values)):
    model_df=final.loc[final.model == model]
    bootstrap=model_df.sample(n=80,replace=True)
    bootstraps.append(bootstrap)

bootstrap_df=pd.concat(bootstraps,ignore_index=True)
results_long=pd.melt(bootstrap_df,id_vars=['model'],var_name='metrics',value_name='values')
time_metrics=['fit_time','score_time']

results_long_nofit=results_long.loc[~results_long['metrics'].isin(time_metrics)]
results_long_nofit=results_long_nofit.sort_values(by='values')

results_long_fit=results_long.loc[results_long['metrics'].isin(time_metrics)]
results_long_fit=results_long_fit.sort_values(by='values')
```

```
plt.figure(figsize=(20,12))
sns.set(font_scale=2.5)
g=sns.boxplot(x="model",y="values",hue="metrics",data=results_long_nofit,palette="Set3")
plt.legend(bbox_to_anchor=(1.02,1),loc=2,borderaxespad=0.)
plt.title('Comparison of Model by Classification Metric')
plt.savefig('./benchmark_model_performance.png',dpi=300)
```

since Logistic Regression and Random Forest have the same high performance scores, we should choose Logistic Regression for its simplicity, interpretability, and efficiency.

Milestone 6: Model Deployment:

Activity 1: Save the best model:

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance.This can be useful in avoiding the need to retrain the model every time it is needed and also to be able to use it in the future.

```
import pickle

pickle.dump(lgr,open('CKD_m2.pkl','wb'))
```

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

●indexnew.html
●about.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
```

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.

```
app=Flask(__name__)
model=pickle.load(open('CKD.pkl','rb'))
```

Render HTML page:

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

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

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

Retrieves the value from UI:

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.

Main Function:

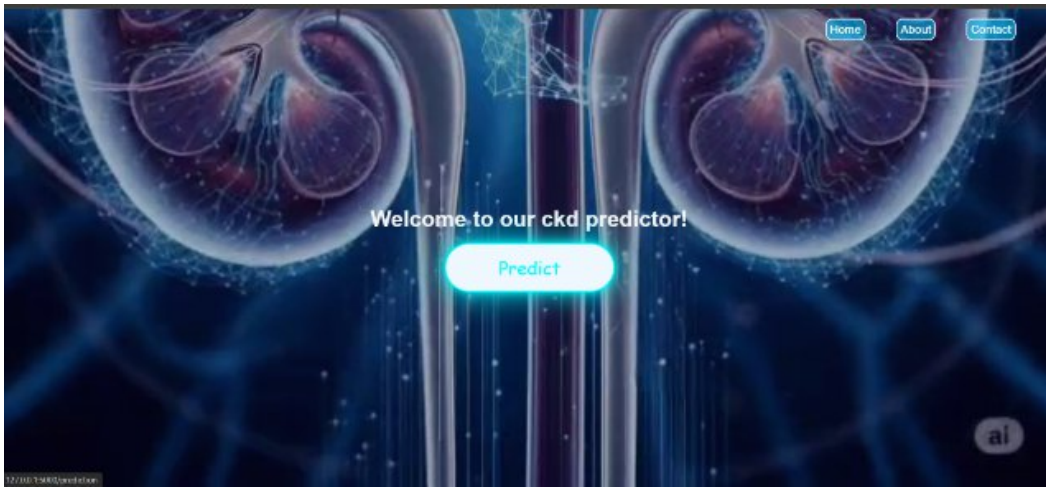
```
if __name__=='__main__':
    app.run(debug=True)
```

Activity 2.3: Run the web application

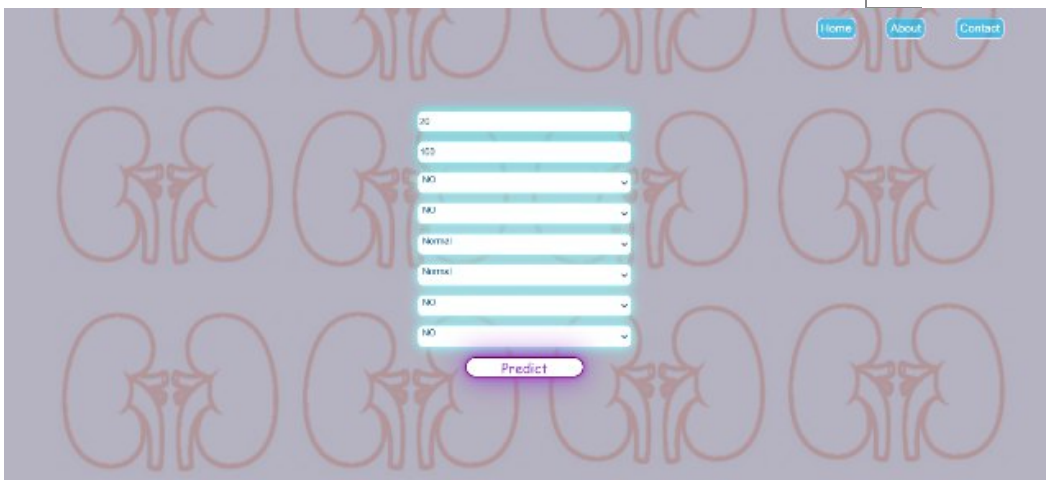
- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top left corner, enter the inputs, click on the submit button, and see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

```
(.venv) PS C:\Users\JAYNIL\python\CKD_prediction> python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server inst
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 291-267-001
```

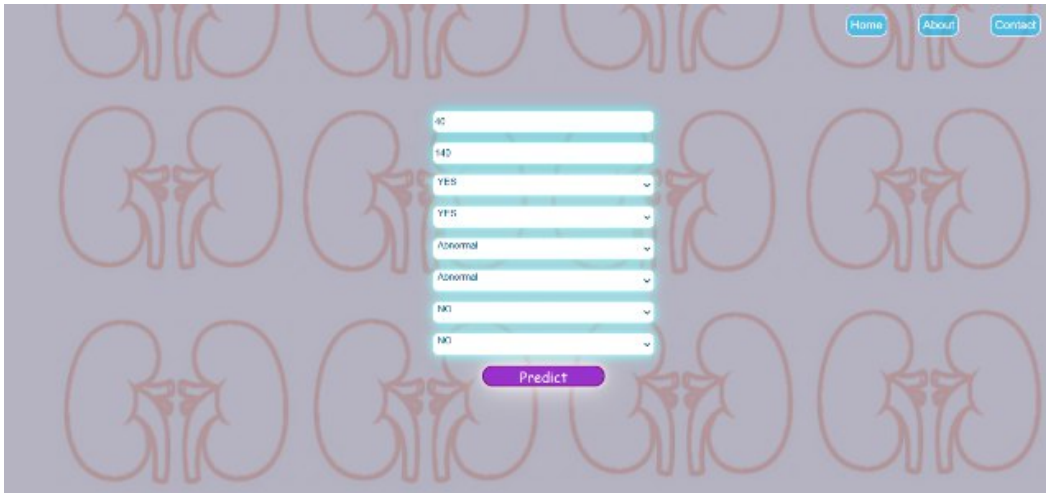
Now , Go the web browser and write the localhost url (http://127.0.0.1:5000) to get the below result.

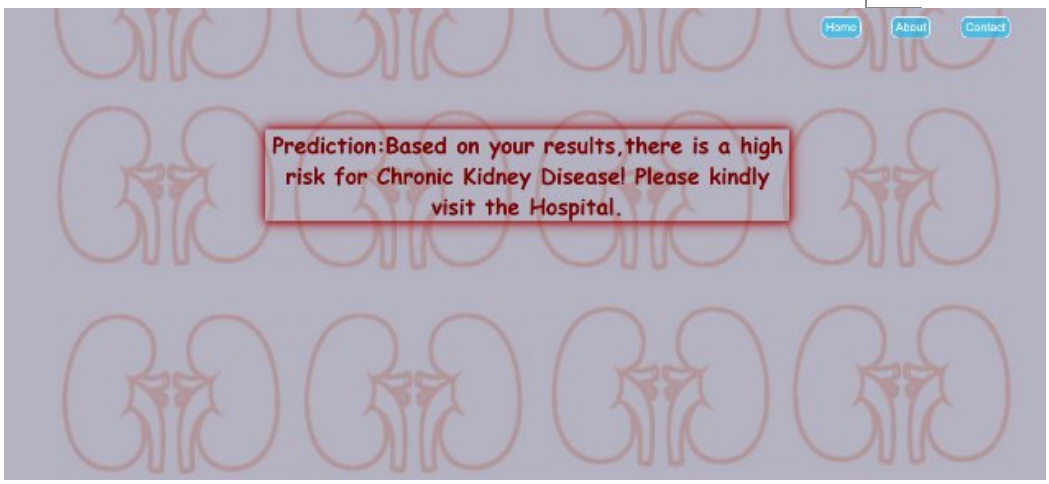


Input - Now, the user will give inputs to get the predicted result after clicking onto the submit button.


Home
About
Contact
40
140
NO
NO
Normal
Normal
NO
NO
Predict


Home
About
Contact
Prediction:Great! You Don't
have Chronic Kidney Disease.


Home
About
Contact
40
140
YES
YES
Abnormal
Abnormal
NO
NO
Predict



Milestone 7: Project Demonstration & Documentation

Below mentioned deliverables to be submitted along with other deliverables

Activity 1:- Record explanation Video for project end to end solution

Activity 2:- Project Documentation-Step by step project development procedure Create document as per the template provided