Early Prediction for Chronic Kidney Disease Detection:

A Progressive Approach to Health Management

INTRODUCTION:

OVERVEW:

Chronic Kidney Disease (CKD) is a major medical problem and can be cured if treated in the early stages. Usually, people are not aware that medical tests we take for different purposes could contain valuable information concerning kidney diseases. Consequently, attributes of various medical tests are investigated to distinguish which attributes may contain helpful information about the disease. The information says that it helps us to measure the severity of the problem, the predicted survival of the patient after the illness, the pattern of the disease and work for curing the disease.

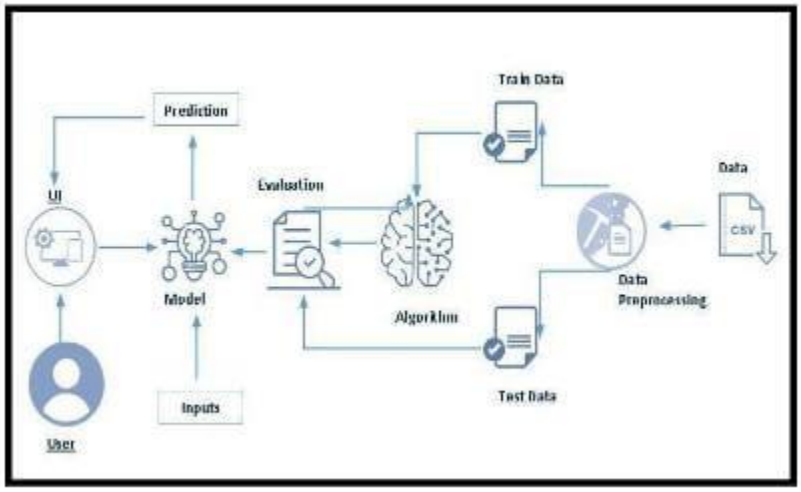
In todays world as we know most of the people are facing so many disease and as this can be cured if we treat people in early stages this project can use a pretrained model to predict the Chronic Kidney Disease which can help in treatments of peoples who are suffer from this disease

PURPOSE:

Your kidneys filter extra water and wastes out of your blood and make urine. Kidney disease means your kidneys are damaged and can't filter blood the way they should. You are at greater risk for kidney disease if you have diabetes or high blood pressure.

PROBLEM DEFINITION & DESIGN THINKING:

EMPATHY MAP IIIIIIIII



IDEATION &BRAINTORIMING MAP:

RESULT:

```
data=pd.read_csv("chronickidneydisease.csv") #loading the csv data
data.head() #return you the first 5 rows values
```

	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

Importing Libaries

```
import pandas as pd #used for data manipulation
import numpy as np #used for numerical analysis
from collections import Counter as c # return counts of number of classess
import matplotlib.pyplot as plt #used for data Visualization
import seaborn as sns #data visualization library
import missingno as msno #finding missing values
from sklearn.metrics import accuracy_score, confusion_matrix#model performance
from sklearn.model_selection import train_test_split #splits data in random train and test array
from sklearn.preprocessing import LabelEncoder #encoding the levels of categorical features
from sklearn.linear_model import LogisticRegression #Classification ML algorithm
import pickle #Python object hierarchy is converted into a byte stream,
```

```
data.columns #return all the column names
```

```
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')
```

```
data.columns=['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','diabetesmellitus','coronary_artery_disease','appetite',
              'pedal_edema','anemia','class'] # manually giving the name  of the columns
data.columns
```

```
Index(['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',
       'diabetesmellitus', 'coronary_artery_disease', 'appetite',
       'pedal_edema', 'anemia', 'class'],
      dtype='object')
```

```
data.info() #info will give you a summary of dataset
```

```
<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   specific_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 
 20  coronary_artery_disease  398 non-null    object 
 21  appetite                 399 non-null    object 
 22  pedal_edema              399 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
```

```
data.isnull().any() #it will return true if any columns is having null values
```

```
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
```

```
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['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['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)
```

```python
for i in catcols:
    print("Columns :",i)
    print(c(data[i])) #using counter for checking the number of classess in the column
    print('*'*120+'\n')
```

```
Columns : hypertension
Counter({'no': 251, 'yes': 147, nan: 2})
************************************************************************************************************************

Columns : packed_cell_volume
Counter({nan: 70, '52': 21, '41': 21, '44': 19, '48': 19, '40': 16, '43': 14, '45': 13, '42': 13, '32': 12, '36': 12, '33': 12, '28': 12,
'50': 12, '37': 11, '34': 11, '35': 9, '29': 9, '30': 9, '46': 9, '31': 8, '39': 7, '24': 7, '26': 6, '38': 5, '47': 4, '49': 4, '53': 4,
'51': 4, '54': 4, '27': 3, '22': 3, '25': 3, '23': 2, '19': 2, '16': 1, '\t?': 1, '14': 1, '18': 1, '17': 1, '15': 1, '21': 1, '20': 1,
'\t43': 1, '9': 1})
************************************************************************************************************************

Columns : class
Counter({'ckd': 250, 'notckd': 150})
************************************************************************************************************************

Columns : coronary_artery_disease
Counter({'no': 362, 'yes': 34, '\tno': 2, nan: 2})
************************************************************************************************************************

Columns : anemia
Counter({'no': 339, 'yes': 60, nan: 1})
************************************************************************************************************************

Columns : red_blood_cell_count
Counter({nan: 130, '5.2': 18, '4.5': 16, '4.9': 14, '4.7': 11, '3.9': 10, '4.8': 10, '4.6': 9, '3.4': 9, '3.7': 8, '5.0': 8, '6.1': 8, '5.
5': 8, '5.9': 8, '3.8': 7, '5.4': 7, '5.8': 7, '5.3': 7, '4.3': 6, '4.2': 6, '5.6': 6, '4.4': 5, '3.2': 5, '4.1': 5, '6.2': 5, '5.1': 5,
'6.4': 5, '5.7': 5, '6.5': 5, '3.6': 4, '6.0': 4, '6.3': 4, '4.0': 3, '4': 3, '3.5': 3, '3.3': 3, '5': 2, '2.6': 2, '2.8': 2, '2.5': 2,
'3.1': 2, '2.1': 2, '2.9': 2, '2.7': 2, '3.0': 2, '2.3': 1, '8.0': 1, '3': 1, '2.4': 1, '\t?': 1})
************************************************************************************************************************
```

Labeling Encoding of Categorical Column

```python
#'specific_gravity','albumin', 'sugar'(as these columns are  numerical it is removed)
catcols=['anemia','pedal_edema','appetite','bacteria','class','coronary_artery_disease','diabetesmellit
 'hypertension','pus_cell','pus_cell_clumps','red_blood_cells'] #only considered the text class columns
```

```python
from sklearn.preprocessing import LabelEncoder #imorting the LabelEncoding from sklearn
for i in catcols: #looping through all the categorical columns
    print("LABEL ENCODING OF:",i)
    LEi = LabelEncoder() # creating an object of LabelEncoder
    print(c(data[i])) #getting the classes values before transformation
    data[i] = LEi.fit_transform(data[i])# trannsforming our text classes to numerical values
    print(c(data[i])) #getting the classes values after transformation
    print("*"*100)
```

```
Columns : red_blood_cells
Counter({'normal': 201, nan: 152, 'abnormal': 47})
**********************************************************************************************************************

Columns : bacteria
Counter({'notpresent': 374, 'present': 22, nan: 4})
**********************************************************************************************************************

Columns : pedal_edema
Counter({'no': 323, 'yes': 76, nan: 1})
**********************************************************************************************************************

Columns : appetite
Counter({'good': 317, 'poor': 82, nan: 1})
**********************************************************************************************************************

Columns : pus_cell
Counter({'normal': 259, 'abnormal': 76, nan: 65})
**********************************************************************************************************************

Columns : diabetesmellitus
Counter({'no': 258, 'yes': 134, '\tno': 3, '\tyes': 2, nan: 2, ' yes': 1})
**********************************************************************************************************************

Columns : pus_cell_clumps
Counter({'notpresent': 354, 'present': 42, nan: 4})
**********************************************************************************************************************

Columns : white_blood_cell_count
Counter({nan: 105, '9800': 11, '6700': 10, '9600': 9, '9200': 9, '7200': 9, '6900': 8, '11000': 8, '5800': 8, '7800': 7, '9100': 7, '940
0': 7, '7000': 7, '4300': 6, '6300': 6, '10700': 6, '10500': 6, '7500': 5, '8300': 5, '7900': 5, '8600': 5, '5600': 5, '10200': 5, '5000':
5, '8100': 5, '9500': 5, '6000': 4, '6200': 4, '10300': 4, '7700': 4, '5500': 4, '10400': 4, '6800': 4, '6500': 4, '4700': 4, '7300': 3,
'4500': 3, '8400': 3, '6400': 3, '4200': 3, '7400': 3, '8000': 3, '5400': 3, '3800': 2, '11400': 2, '5300': 2, '8500': 2, '14600': 2, '710
0': 2, '13200': 2, '9000': 2, '8200': 2, '15200': 2, '12400': 2, '12800': 2, '8800': 2, '5700': 2, '9300': 2, '6600': 2, '12100': 1, '1220
0': 1, '18900': 1, '21600': 1, '11300': 1, '\t6200': 1, '11800': 1, '12500': 1, '11900': 1, '12700': 1, '13600': 1, '14900': 1, '16300':
1, '\t8400': 1, '10900': 1, '2200': 1, '11200': 1, '19100': 1, '\t?': 1, '12300': 1, '16700': 1, '2600': 1, '26400': 1, '4900': 1, '1200
0': 1, '15700': 1, '4100': 1, '11500': 1, '10800': 1, '9900': 1, '5200': 1, '5900': 1, '9700': 1, '5100': 1})
**********************************************************************************************************************
```

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

Labeling Encoding of Categorical Column

```
#'specific_gravity','albumin', 'sugar'(as these columns are  numerical it is removed)
catcols=['anemia','pedal_edema','appetite','bacteria','class','coronary_artery_disease','diabetesmellit
 'hypertension','pus_cell','pus_cell_clumps','red_blood_cells'] #only considered the text class columns
```

```
from sklearn.preprocessing import LabelEncoder #imorting the LabelEncoding from sklearn
for i in catcols: #looping through all the categorical columns
    print("LABEL ENCODING OF:",i)
    LEi = LabelEncoder() # creating an object of LabelEncoder
    print(c(data[i])) #getting the classes values before transformation
    data[i] = LEi.fit_transform(data[i])# trannsforming our text classes to numerical values
    print(c(data[i])) #getting the classes values after transformation
    print("*"*100)
```

```
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'}
```

```
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})
```

```
data.describe() # computes summary values for continous column data
```

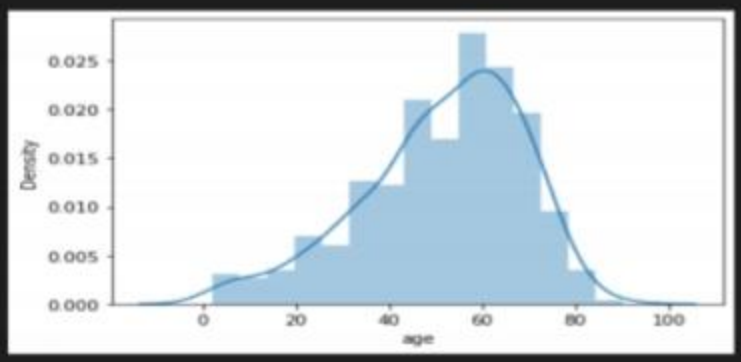
	age	blood_pressure	specific_gravity	albumin	sugar	blood glucose random	blood_urea	serum_creatinine	sodium
count	391.000000	388.000000	353.000000	354.000000	351.000000	356.000000	381.000000	383.000000	313.000000
mean	51.483376	76.469072	1.017408	1.016949	0.450142	148.036517	57.425722	3.072454	137.528754
std	17.169714	13.683637	0.005717	1.352679	1.099191	79.281714	50.503006	5.741126	10.408752
min	2.000000	50.000000	1.005000	0.000000	0.000000	22.000000	1.500000	0.400000	4.500000
25%	42.000000	70.000000	1.010000	0.000000	0.000000	99.000000	27.000000	0.900000	135.000000
50%	55.000000	80.000000	1.020000	0.000000	0.000000	121.000000	42.000000	1.300000	138.000000
75%	64.500000	80.000000	1.020000	2.000000	0.000000	163.000000	66.000000	2.800000	142.000000
max	90.000000	180.000000	1.025000	5.000000	5.000000	490.000000	391.000000	76.000000	163.000000

We can find the distribution of the feature:

Age distribution

```
[236] sns.distplot(data.age)
```

```
C:\Users\Saumya\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWa
your code to use either `displot` (a figure-level function with similar flexibility
  warnings.warn(msg, FutureWarning)

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




```
data.describe() # computes summary values for continous column data
```

	age	blood_pressure	specific_gravity	albumin	sugar	blood glucose random	blood_urea	serum_creatinine	sodium
count	391.000000	388.000000	353.000000	354.000000	351.000000	356.000000	381.000000	383.000000	313.000000
mean	51.483376	76.469072	1.017408	1.016949	0.450142	148.036517	57.425722	3.072454	137.528754
std	17.169714	13.683637	0.005717	1.352679	1.099191	79.281714	50.503006	5.741126	10.408752
min	2.000000	50.000000	1.005000	0.000000	0.000000	22.000000	1.500000	0.400000	4.500000
25%	42.000000	70.000000	1.010000	0.000000	0.000000	99.000000	27.000000	0.900000	135.000000
50%	55.000000	80.000000	1.020000	0.000000	0.000000	121.000000	42.000000	1.300000	138.000000
75%	64.500000	80.000000	1.020000	2.000000	0.000000	163.000000	66.000000	2.800000	142.000000
max	90.000000	180.000000	1.025000	5.000000	5.000000	490.000000	391.000000	76.000000	163.000000

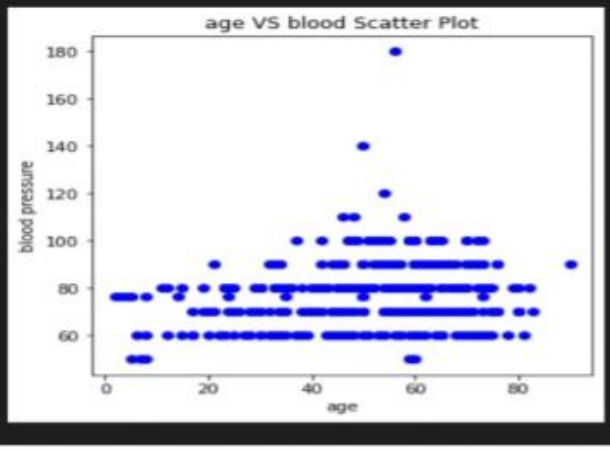
Age vs all continous columns

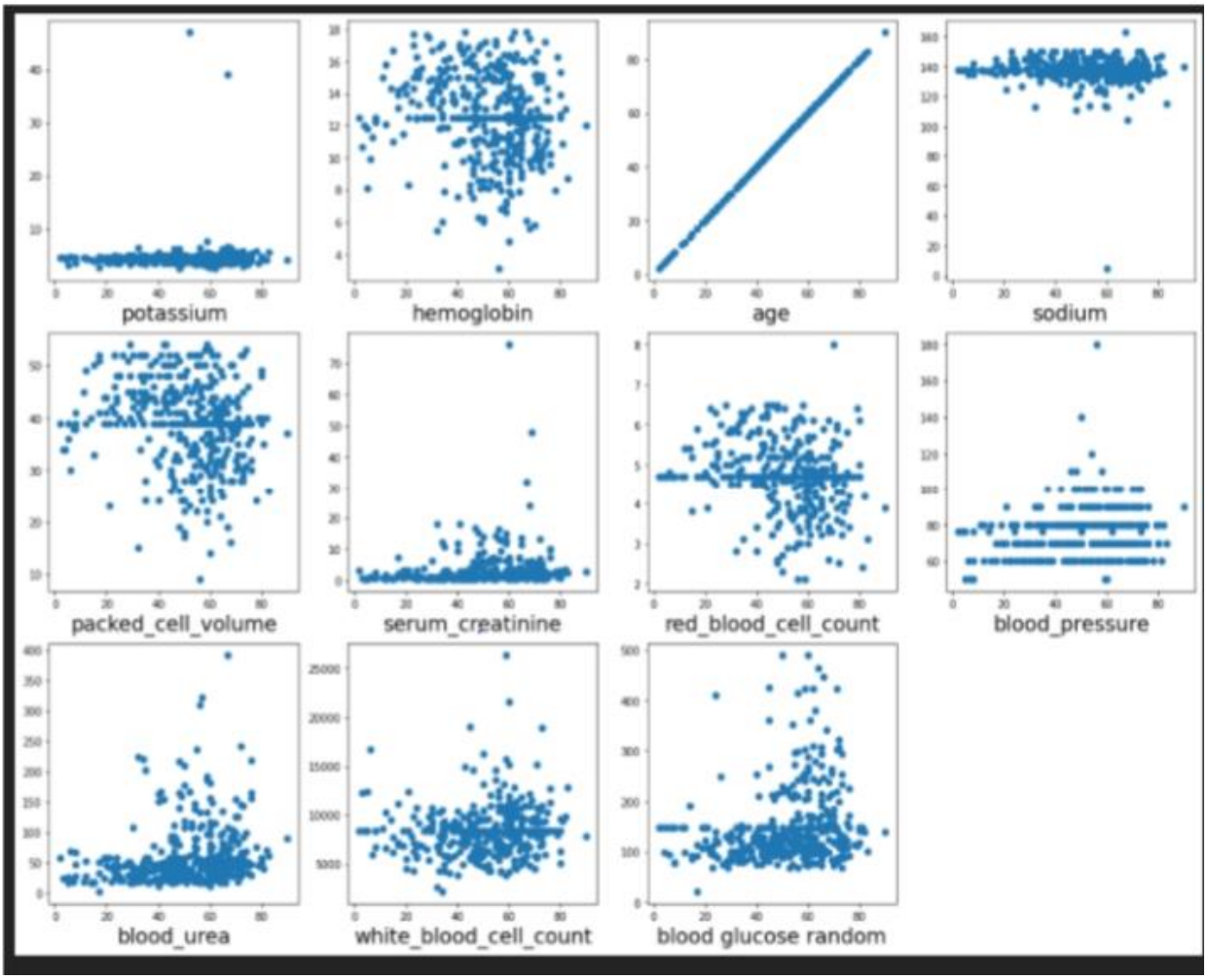
```
plt.figure(figsize=(20,15), facecolor='white')
plotnumber = 1

for column in contcols:
    if plotnumber<=11 :     # as there are 11 continous columns in the data
        ax = plt.subplot(3,4,plotnumber) # 3,4 is refer to 3X4 matrix
        plt.scatter(data['age'],data[column]) #plotting scatter plot
        plt.xlabel(column,fontsize=20)
        #plt.ylabel('Salary',fontsize=20)
    plotnumber+=1
plt.show()
```

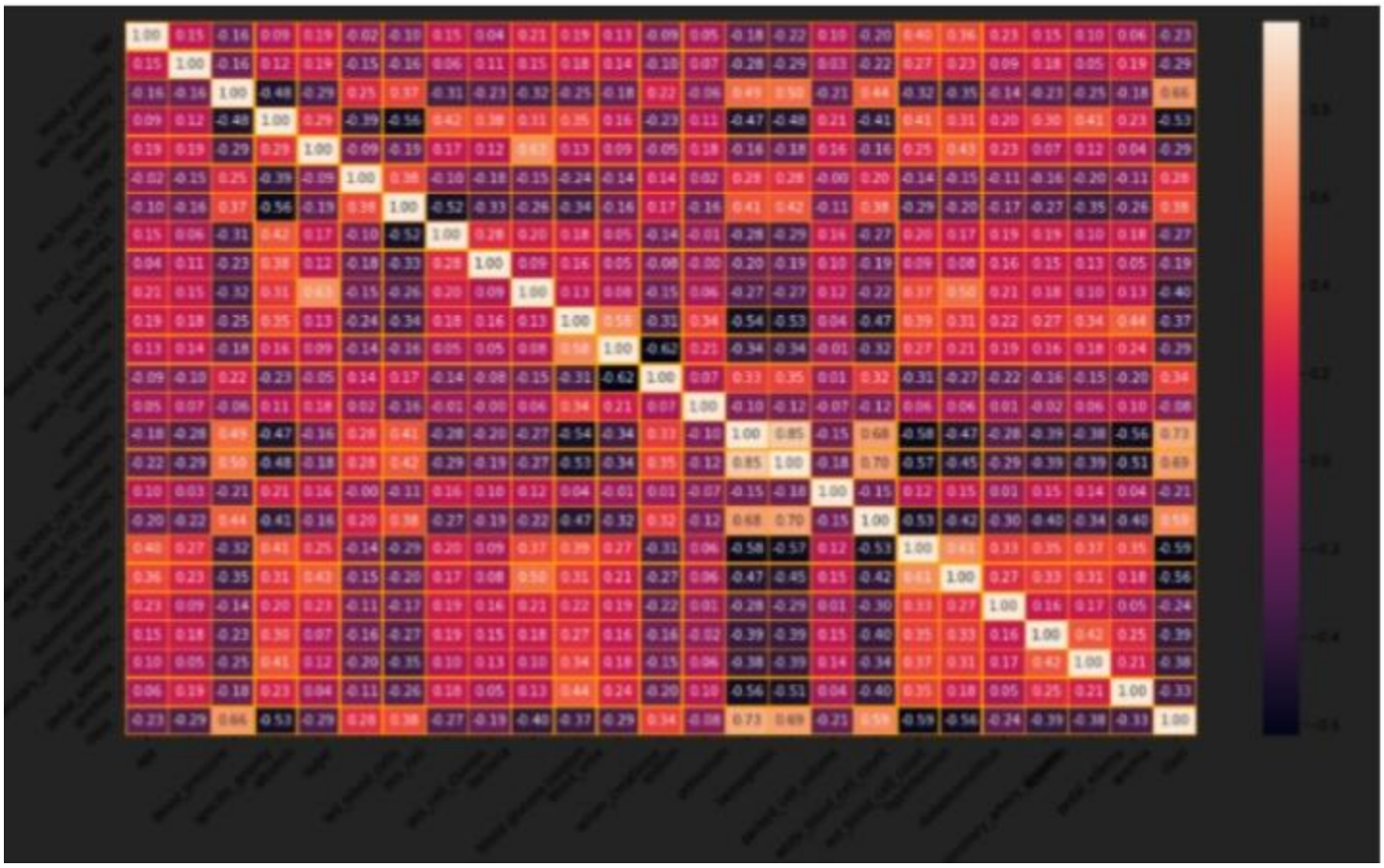
Age vs Blood Pressure

```
import matplotlib.pyplot as plt # import the matplotlib libaray
fig=plt.figure(figsize=(5,5)) #plot size
plt.scatter(data['age'],data['blood_pressure'],color='blue')
plt.xlabel('age') #set the label for x-axis
plt.ylabel('blood pressure') #set the label for y-axis
plt.title("age VS blood Scatter Plot") #set a title for the axes
```

```
Text(0.5, 1.0, 'age VS blood Scatter Plot')
```




potassium
hemoglobin
age
sodium
packed_cell_volume
serum_creatinine
red_blood_cell_count
blood_pressure
blood_urea
white_blood_cell_count
blood glucose random

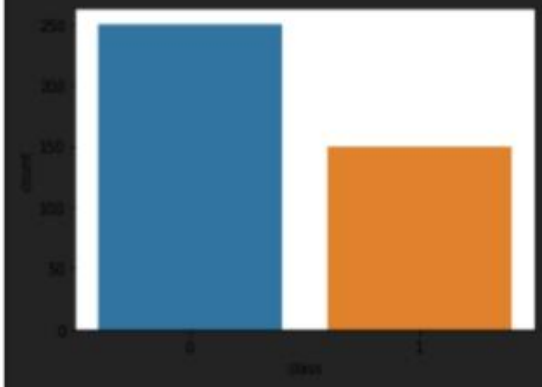


Finding correlation between the independent Columns

```
#HEAT MAP #correlation of parameters
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()
```

```
sns.countplot(data['class'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x20c1d390d30>
```




```
# perfroming feature Scaling op[eration using standard scaller on X part of the dataset because
# there different type of values in the columns
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_bal=sc.fit_transform(x)
```

```
# Importing the Keras libraries and packages
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```
# Creating ANN skleton view

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'))
```

Splitting the data into train and test

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=2)#train test split
```

```
[==============================] - 0s 4ms/step - loss: 0.1139 - accuracy: 0.9531 - val_loss: 0.2799 - val_accuracy: 0.8906 Ep
...
Epoch 99/100 26/26 [==============================] - 0s 3ms/step - loss: 0.1074 - accuracy: 0.9570 - val_loss: 0.2439 - va
[==============================] - 0s 4ms/step - loss: 0.1062 - accuracy: 0.9570 - val_loss: 0.2572 - val_accuracy: 0.9062

<tensorflow.python.keras.callbacks.History at 0x1fdf3ca7b20>
```

```
# Compiling the ANN model

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

```
# Training the model

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

```
Output exceeds the size limit. Open the full output data in a text editor
Epoch 1/100
26/26 [==============================] - 0s 6ms/step - loss: 0.1151 - accuracy: 0.9531 - val_loss: 0.2476 - val_accuracy: 0.9062
Epoch 2/100
26/26 [==============================] - 0s 4ms/step - loss: 0.1171 - accuracy: 0.9570 - val_loss: 0.2498 - val_accuracy: 0.9062
Epoch 3/100
26/26 [==============================] - 0s 4ms/step - loss: 0.1146 - accuracy: 0.9531 - val_loss: 0.2317 - val_accuracy: 0.9219
Epoch 4/100
26/26 [==============================] - 0s 4ms/step - loss: 0.1305 - accuracy: 0.9531 - val_loss: 0.2855 - val_accuracy: 0.8906
Epoch 5/100
26/26 [==============================] - 0s 4ms/step - loss: 0.1387 - accuracy: 0.9492 - val_loss: 0.2068 - val_accuracy: 0.9219
Epoch 6/100
26/26 [==============================] - 0s 4ms/step - loss: 0.1230 - accuracy: 0.9492 - val_loss: 0.2576 - val_accuracy: 0.9062
Epoch 7/100
26/26 [==============================] - 0s 4ms/step - loss: 0.1241 - accuracy: 0.9531 - val_loss: 0.2688 - val_accuracy: 0.8906
Epoch 8/100
26/26 [==============================] - 0s 4ms/step - loss: 0.1128 - accuracy: 0.9570 - val_loss: 0.2334 - val_accuracy: 0.9219
Epoch 9/100
26/26 [==============================] - 0s 4ms/step - loss: 0.1180 - accuracy: 0.9531 - val_loss: 0.2435 - val_accuracy: 0.9062
Epoch 10/100
```

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

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

```
<ipython-input-255-b87bb2ba9825>:1: DataConversionWarning: A column-vector y wa
(n_samples,), for example using ravel().
  rfc.fit(x_train,y_train)

RandomForestClassifier(criterion='entropy', n_estimators=10)
```

```
y_predict = rfc.predict(x_test)
```

+ Code

```
y_predict_train = rfc.predict(x_train)
```

```
from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier(max_depth=4,splitter='best',criterion='entropy')
```

```
dtc.fit(x_train,y_train)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

```
y_predict= dtc.predict(x_test)
y_predict
```

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

```
y_predict_train = dtc.predict(x_train)
```

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

```
C:\Users\Saumya\Anaconda3\lib\site-packages\sklearn\utils\validation.py:72: DataConversionWar
Please change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

LogisticRegression()
```

Predicting our output with the model which we build

```
from sklearn.metrics import accuracy_score,classification_report

y_predict = lgr.predict(x_test)
```

```
# logistic Regression

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

print(y_pred)
(y_pred)
```

```
[0]

array([0])
```

```
# DecisionTree classifier

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

print(y_pred)
(y_pred)
```

```
[0]

array([0])
```

```
# Random Forest Classifier
y_pred = rfc.predict([[1,1,121.000000,36.0,0,0,1,0]])

print(y_pred)
(y_pred)
```

```
[0]

array([0])
```

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

```
# Testing the model

y_pred = classification.predict(x_test)
```

```
y_pred
```

```
Output exceeds the size limit. Open the full output data in a text editor
array([[2.07892948e-12],
       [7.16007332e-13],
       [0.00000000e+00],
       [6.47086192e-23],
       [9.99349952e-01],
       [1.47531908e-22],
       [0.00000000e+00]
```

```
y_pred = (y_pred > 0.5)
y_pred
```

```
Output exceeds the size limit. Open the full output data in a te
array([[False],
       [False],
       [False],
       [False],
       [ True],
       [False],
       [False],
```

```
def predict_exit(sample_value):

  # Convert list to numpy array
    sample_value = np.array(sample_value)

  # Reshape because sample_value contains only 1 record
    sample_value = sample_value.reshape(1, -1)

  # Feature Scaling
    sample_value = sc.transform(sample_value)

    return classifier.predict(sample_value)
```

```
test=classification.predict([[1,1,121.000000,36.0,0,0,1,0]])
if test==1:
    print('Prediction: High chance of CKD!')
else:
    print('Prediction: Low chance of CKD.')
```

```
Prediction: Low chance of CKD.
```

LogReg

	precision	recall	f1-score	support
NO CKD	1.00	0.87	0.93	54
CKD	0.79	1.00	0.88	26
accuracy			0.91	80
macro avg	0.89	0.94	0.91	80
weighted avg	0.93	0.91	0.91	80

Compare the model

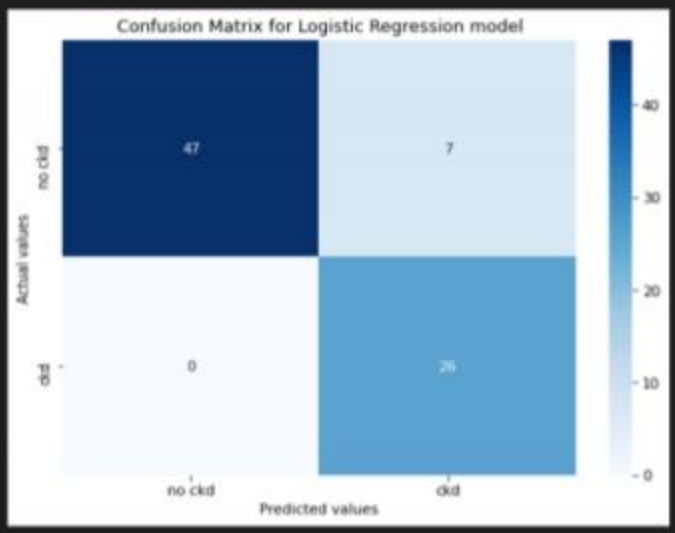
```
from sklearn import model_selection
```

```
    dfs = []
models = [
          ('LogReg', LogisticRegression()),
          ('RF', RandomForestClassifier()),
          ('DecisionTree',DecisionTreeClassifier()),

        ]
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=90210)
        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)
final = pd.concat(dfs, ignore_index=True)
return final
```

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_predict)
cm
```

```
array([[47,  7],
       [ 0, 26]], dtype=int64)
```

```
# Plotting confusion matrix
plt.figure(figsize=(8,6))
sns.heatmap(cm, cmap='Blues', annot=True, xticklabels=['no ckd', 'ckd'], yticklabels=['no ckd', 'ckd'])
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.title('Confusion Matrix for Logistic Regression model')
plt.show()
```



LogReg

	precision	recall	f1-score	support
NO CKD	1.00	0.87	0.93	54
CKD	0.79	1.00	0.88	26
accuracy			0.91	80
macro avg	0.89	0.94	0.91	80
weighted avg	0.93	0.91	0.91	80

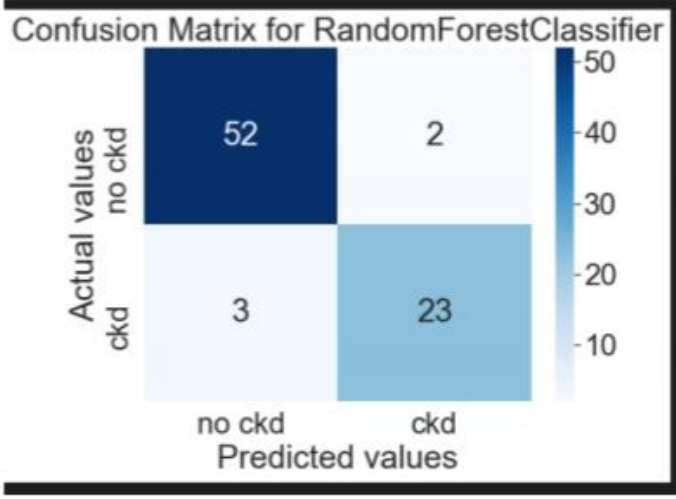
DecisionTree

	precision	recall	f1-score	support
NO CKD	0.93	0.94	0.94	54
CKD	0.88	0.85	0.86	26
accuracy			0.91	80
macro avg	0.90	0.90	0.90	80
weighted avg	0.91	0.91	0.91	80

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_predict)
cm
```

```
array([[52,  2],
       [ 3, 23]], dtype=int64)
```

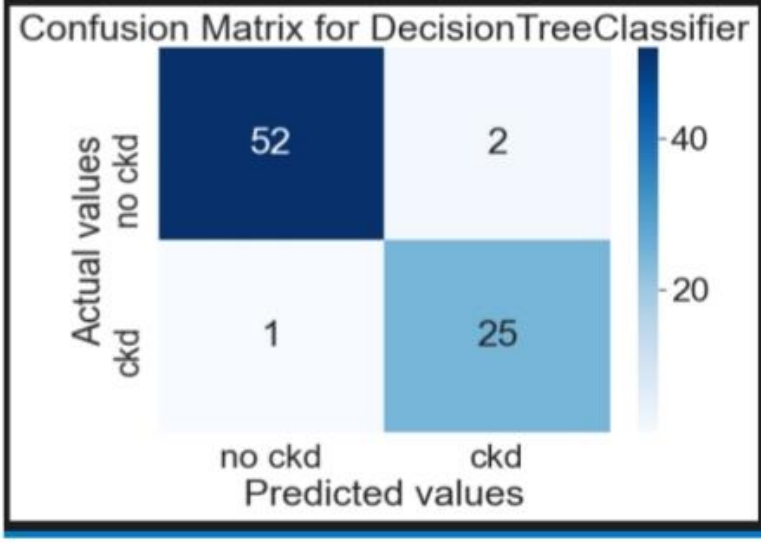
```
# Plotting confusion matrix
plt.figure(figsize=(8,6))
sns.heatmap(cm, cmap='Blues', annot=True, xticklabels=['no ckd', 'ckd'], yticklabels=['no ckd', 'ckd'])
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.title('Confusion Matrix for RandomForestClassifier')
plt.show()
```



```python
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_predict)
cm
```

```
array([[52,  2],
       [ 1, 25]], dtype=int64)
```

```python
# Plotting confusion matrix
plt.figure(figsize=(8,6))
sns.heatmap(cm, cmap='Blues', annot=True, xticklabels=['no ckd', 'ckd'], yticklabels=['no ckd', 'ckd'])
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.title('Confusion Matrix for DecisionTreeClassifier')
plt.show()
```




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

[201]

```
                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
```

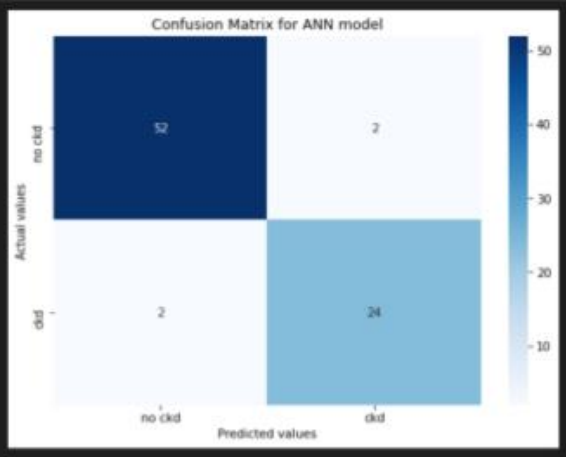
```
print (classification_report(y_test, y_pred))
```
[201]

```
...                  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
```

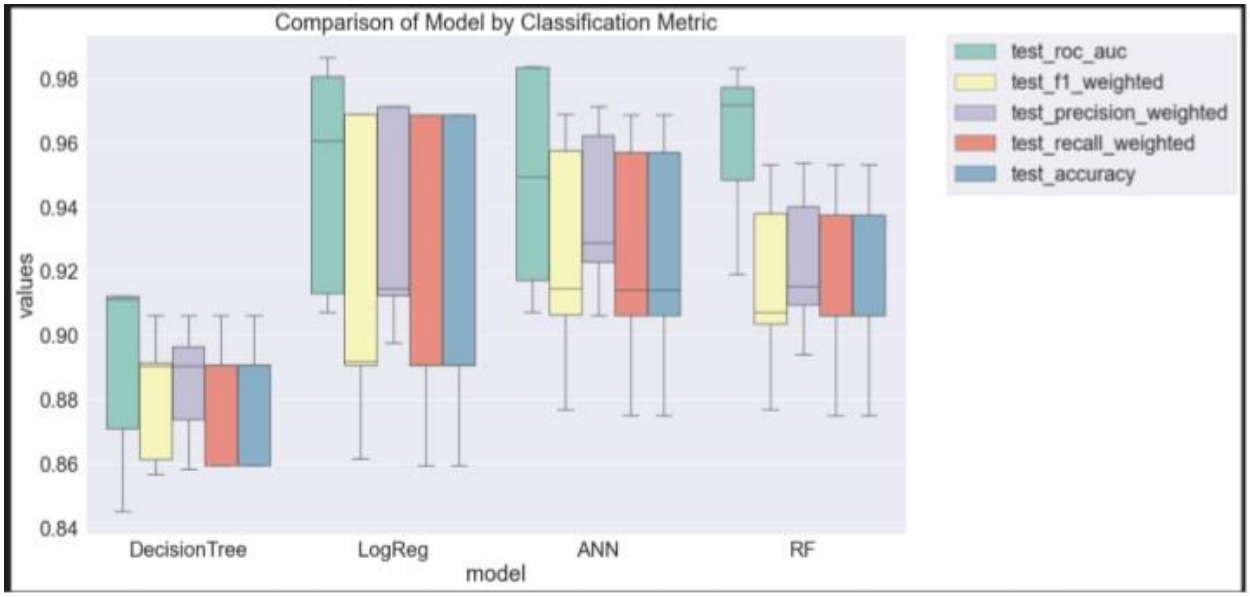
```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

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

```
# Plotting confusion matrix
plt.figure(figsize=(8,6))
sns.heatmap(cm, cmap='Blues', annot=True, xticklabels=['no ckd', 'ckd'], yticklabels=['no ckd', 'ckd'])
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.title('Confusion Matrix for ANN model')
plt.show()
```



```
import matplotlib.pyplot as plt
import seaborn as sns
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=(1.05, 1), loc=2, borderaxespad=0.)
plt.title('Comparison of Model by Classification Metric')
plt.savefig('./benchmark_models_performance.png',dpi=300)
```




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

```
from flask import Flask, render_template, request
import numpy as np
import pickle
```

```
app = Flask(__name__) # initializing a flask app
model = pickle.load(open('CKD.pkl', 'rb')) #loading the model
```

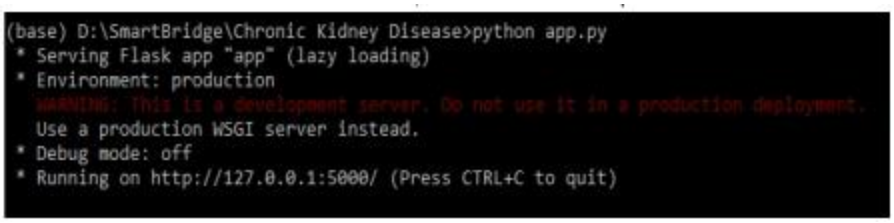
```python
@app.route('/')# route to display the home page
def home():
    return render_template('home.html') #rendering the home page
```

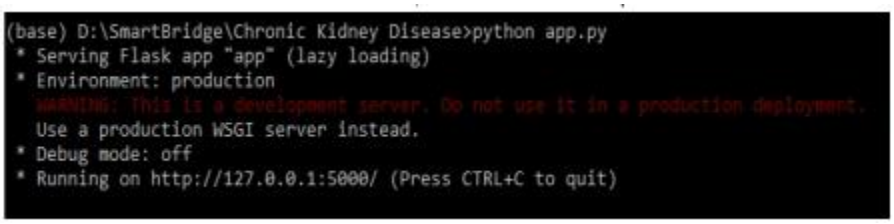
```python
@app.route('/Prediction',methods=['POST','GET'])

def prediction():
    return render_template('indexnew.html')
@app.route('/Home',methods=['POST','GET'])
def my_home():
    return render_template('home.html')

@app.route('/predict',methods=['POST'])# route to show the predictions in a web UI
def predict():

    #reading the inputs given by the user
    input_features = [float(x) for x in request.form.values()]
    features_value = [np.array(input_features)]

    features_name = ['blood_urea', 'blood glucose random', 'anemia',
       'coronary_artery_disease', 'pus_cell', 'red_blood_cells',
       'diabetesmellitus', 'pedal_edema']

    df = pd.DataFrame(features_value, columns=features_name)

    # predictions using the loaded model file
    output = model.predict(df)
```

```python
# showing the prediction results in a UI# showing the prediction results in a UI
return render_template('result.html', prediction_text=output)
```

```python
if __name__ == '__main__':
    # running the app
    app.run(debug=True)
```

```
(base) D:\SmartBridge\Chronic Kidney Disease>python app.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```





ADVANTAGES & DISADVANTAGES:

ADVANTAGES:

Dialysis is a treatment for people who have renal failure. Renal failure causes your kidneys to stop filtering blood. As a result, wastes and poisons build up in your bloodstream. Dialysis aids your kidneys in their function by removing waste and excess fluid from your blood.

DISADVANTAGES:

Low blood pressure (hypotension) is one of the most common side effects of haemodialysis. It can be caused by the drop in fluid levels during dialysis. Low blood pressure can cause nausea and dizziness. The best way to minimise these symptoms of low blood pressure is to keep to your daily fluid intake recommendations.

The most common side effects of hemodialysis include low blood pressure, access site infection, muscle cramps, itchy skin, and blood clots. The most common side effects of peritoneal dialysis include peritonitis, hernia, blood sugar changes, potassium imbalances, and weight gain.

APPLICATIONS:

Most people know that a major function of the kidneys is to remove waste products and excess fluid from the body. These waste products and excess fluid are removed through the urine. The production of urine involves highly complex steps of excretion and re-absorption.

CONCLUSION:

Chronic kidney disease develops indolently, with many patients diagnosed late and a specific cause never established in a significant number of patients. It has various multi-system complications, significantly impairing the quality of life and shortening the life span of victims.

FUTURE SCOPE:

The inceasing prevalence of chronic kidney disease is well known, as it is a fact that recorded data in all countries show continuing growth in the number of patients that need substitutive treatment for their renal function. The consequences from the social and economic viewpoint are very

significant and we cannot be happy with morbidity and mortality rates in terminal stage renal patients that continue to be unacceptably high.

APPENDIX:

Importing Libaries

```
import pandas as pd #used for data manipulation
import numpy as np #used for numerical analysis
from collections import Counter as c # return counts of number of classes
import matplotlib.pyplot as plt #used for data Visualization
import seaborn as sns #data visualization library
import missingno as msno #finding missing values
from sklearn.metrics import accuracy_score, confusion_matrix#model performance
from sklearn.model_selection import train_test_split #splits data in random train and test array
from sklearn.preprocessing import LabelEncoder #encoding the levels of categorical features
from sklearn.linear_model import LogisticRegression #Classification ML algorithm
import pickle #Python object hierarchy is converted into a byte stream,
```

```
data.columns #return all the column names
```

```
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')
```

```
data.columns=['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','diabetesmellitus','coronary_artery_disease','appetite',
              'pedal_edema','anemia','class'] # manually giving the name  of the columns
data.columns
```

```
Index(['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',
       'diabetesmellitus', 'coronary_artery_disease', 'appetite',
       'pedal_edema', 'anemia', 'class'],
      dtype='object')
```

```
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)
```

```
catcols=set(data.dtypes[data.dtypes=='O'].index.values) # only fetch the object type columns
print(catcols)
```

```
for i in catcols:
    print("Columns :",i)
    print(c(data[i])) #using counter for checking the number of classess in the column
    print('*'*120+'\n')
```

```
#'specific_gravity','albumin', 'sugar'(as these columns are  numerical it is removed)
catcols=['anemia','pedal_edema','appetite','bacteria','class','coronary_artery_disease','diabetesmellit
 'hypertension','pus_cell','pus_cell_clumps','red_blood_cells'] #only considered the text class columns
```

```
from sklearn.preprocessing import LabelEncoder #imorting the LabelEncoding from sklearn
for i in catcols: #looping through all the categorical columns
    print("LABEL ENCODING OF:",i)
    LEi = LabelEncoder() # creating an object of LabelEncoder
    print(c(data[i])) #getting the classes values before transformation
    data[i] = LEi.fit_transform(data[i])# trannsforming our text classes to numerical values
    print(c(data[i])) #getting the classes values after transformation
    print("*"*100)
```

```
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'}
```

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

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

```
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'}
```

```
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})
```

```
import matplotlib.pyplot as plt # import the matplotlib libaray
fig=plt.figure(figsize=(5,5)) #plot size
plt.scatter(data['age'],data['blood_pressure'],color='blue')
plt.xlabel('age') #set the label for x-axis
plt.ylabel('blood pressure') #set the label for y-axis
plt.title("age VS blood Scatter Plot") #set a title for the axes
```

```
import matplotlib.pyplot as plt # import the matplotlib libaray
fig=plt.figure(figsize=(5,5)) #plot size
plt.scatter(data['age'],data['blood_pressure'],color='blue')
plt.xlabel('age') #set the label for x-axis
plt.ylabel('blood pressure') #set the label for y-axis
plt.title("age VS blood Scatter Plot") #set a title for the axes
```

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

for column in contcols:
    if plotnumber<=11 :     # as there are 11 continous columns in the data
        ax = plt.subplot(3,4,plotnumber) # 3,4 is refer to 3X4 matrix
        plt.scatter(data['age'],data[column]) #plotting scatter plot
        plt.xlabel(column,fontsize=20)
        #plt.ylabel('Salary',fontsize=20)
    plotnumber+=1
plt.show()
```

```
#HEAT MAP #correlation of parameters
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()
```

```
def predict exit(sample value):

  # Convert list to numpy array
    sample_value = np.array(sample_value)

  # Reshape because sample_value contains only 1 record
    sample_value = sample_value.reshape(1, -1)

  # Feature Scaling
    sample_value = sc.transform(sample_value)

    return classifier.predict(sample_value)
```

```
test=classification.predict([[1,1,121.000000,36.0,0,0,1,0]])
if test==1:
    print('Prediction: High chance of CKD!')
else:
    print('Prediction: Low chance of CKD.')
```

```
Prediction: Low chance of CKD.
```

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_predict)
cm
```

```
array([[52,  2],
       [ 3, 23]], dtype=int64)
```

```
# Plotting confusion matrix
plt.figure(figsize=(8,6))
sns.heatmap(cm, cmap='Blues', annot=True, xticklabels=['no ckd', 'ckd'], yticklabels=['no ckd', 'ckd'])
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.title('Confusion Matrix for RandomForestClassifier')
plt.show()
```

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_predict)
cm
```

```
array([[52,  2],
       [ 1, 25]], dtype=int64)
```

```
# Plotting confusion matrix
plt.figure(figsize=(8,6))
sns.heatmap(cm, cmap='Blues', annot=True, xticklabels=['no ckd', 'ckd'], yticklabels=['no ckd', 'ckd'])
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.title('Confusion Matrix for DecisionTreeClassifier')
plt.show()
```

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

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

```
# Plotting confusion matrix
plt.figure(figsize=(8,6))
sns.heatmap(cm, cmap='Blues', annot=True, xticklabels=['no ckd', 'ckd'], yticklabels=['no ckd', 'ckd'])
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.title('Confusion Matrix for ANN model')
plt.show()
```

```
bootstraps = []
for model in list(set(final.model.values)):
    model_df = final.loc[final.model == model]
    bootstrap = model_df.sample(n=30, 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'] # fit time metrics
## PERFORMANCE METRICS
results_long_nofit = results_long.loc[~results_long['metrics'].isin(time_metrics)] # get df without fit data
results_long_nofit = results_long_nofit.sort_values(by='values')
## TIME METRICS
results_long_fit = results_long.loc[results_long['metrics'].isin(time_metrics)] # df with fit data
results_long_fit = results_long_fit.sort_values(by='values')
```

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

```
from flask import Flask, render_template, request
import numpy as np
import pickle
```

```
app = Flask(__name__) # initializing a flask app
model = pickle.load(open('CKD.pkl', 'rb')) #loading the model
```

```
@app.route('/')# route to display the home page
def home():
    return render_template('home.html') #rendering the home page
```

```
@app.route('/Prediction',methods=['POST','GET'])

def prediction():
    return render_template('indexnew.html')
@app.route('/Home',methods=['POST','GET'])
def my_home():
    return render_template('home.html')

@app.route('/predict',methods=['POST'])# route to show the predictions in a web UI
def predict():

    #reading the inputs given by the user
    input_features = [float(x) for x in request.form.values()]
    features_value = [np.array(input_features)]

    features_name = ['blood_urea', 'blood glucose random', 'anemia',
       'coronary_artery_disease', 'pus_cell', 'red_blood_cells',
       'diabetesmellitus', 'pedal_edema']

    df = pd.DataFrame(features_value, columns=features_name)

    # predictions using the loaded model file
    output = model.predict(df)
```

```
# showing the prediction results in a UI# showing the prediction results in a UI
return render_template('result.html', prediction_text=output)
```

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