In [216]:

```
#import necessary packages

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 bytes
import warnings #Ignore warning
warnings.filterwarnings('ignore')
```

In [217]:

```
#read the data

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

Out [217]:

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	a
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	n
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	...	38	6000	NaN	no	no	no	good	no	n
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31	7500	NaN	no	yes	no	poor	no	y
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	y
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	n

5 rows × 26 columns

In [218]:

```
#Data preparation
#Rename the columns

data.columns
```

Out [218]:

```
Index(['id', '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')
```

In [219]:

```
data.columns=(['Id','Age','Blood_pressure','Specific_gravity','Albumin','Sugar','Red_blood_cells','Pu
data.columns
```

Out [219]:

```
Index(['Id', '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')
```

In [220]:

```
#Handling the missing values
#info will give you a summary of dataset

data.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   Blood_pressure           388 non-null    float64
 3   Specific_gravity         353 non-null    float64
 4   Albumin                  354 non-null    float64
```

```
 5   Sugar                   351 non-null    float64
 6   Red_blood_cells         248 non-null    object
 7   Pus_cell                335 non-null    object
 8   Pus_cell_clumps         396 non-null    object
 9   Bacteria                396 non-null    object
 10  Blood glucose random    356 non-null    float64
 11  Blood_urea              381 non-null    float64
 12  Serum_creatinine        383 non-null    float64
 13  Sodium                  313 non-null    float64
 14  Potassium               312 non-null    float64
 15  Hemoglobin              348 non-null    float64
 16  Packed_cell_volume      330 non-null    object
 17  White_blood_cell_count  295 non-null    object
 18  Red_blood_cell_count    270 non-null    object
 19  Hypertension            398 non-null    object
 20  Diabetesmellitus        398 non-null    object
 21  Coronary_artery_disease 398 non-null    object
 22  Appetite                399 non-null    object
 23  Pedal_edema             399 non-null    object
 24  Anemia                  399 non-null    object
 25  Class                   400 non-null    object
dtypes: float64(11), int64(1), object(14)
memory usage: 81.4+ KB
```

In [221]:

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

Out [221]:

```
Id                          False
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
```

In [222]:

```
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)
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['Packed_cell_volume'].fillna(data['Packed_cell_volume'].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)
```

In [223]:

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

Out [223]:
```
Id                         0
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
```

In [224]:
```
#replaces the unwanted classes

data['Class'] = data['Class'].replace(to_replace = {'ckd\t': 'ckd', 'notckd': 'not ckd'})
```

In [225]:
```
data['Class'] = data['Class'].map({'ckd': 0, 'not ckd': 1})
c(data['Class'])
```

Out [225]:
```
Counter({0: 250, 1: 150})
```

In [226]:
```
#handling categorical columns

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

```
{'Red_blood_cells', 'Anemia', 'Red_blood_cell_count', 'Diabetesmellitus', 'Packed_cell_volume', 'Hypertension', 'Pus_cell', 'Coro
```

In [227]:
```
for i in catcols:
  print("Columns:",i)
  print(c(data[i]))   #using counter for checking the number of classes in the column
  print('*'*120+'\n')
```

```
Columns: Red_blood_cells
Counter({'normal': 353, 'abnormal': 47})
************************************************************************************************************************

Columns: Anemia
Counter({'no': 340, 'yes': 60})
************************************************************************************************************************

Columns: Red_blood_cell_count
Counter({'5.2': 148, '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.
************************************************************************************************************************

Columns: Diabetesmellitus
Counter({'no': 260, 'yes': 134, '\tno': 3, '\tyes': 2, ' yes': 1})
************************************************************************************************************************

Columns: Packed_cell_volume
Counter({'41': 91, '52': 21, '44': 19, '48': 19, '40': 16, '43': 14, '45': 13, '42': 13, '32': 12, '36': 12, '33': 12, '28': 12,
************************************************************************************************************************

Columns: Hypertension
Counter({'no': 253, 'yes': 147})
************************************************************************************************************************

Columns: Pus_cell
Counter({'normal': 324, 'abnormal': 76})
************************************************************************************************************************

Columns: Coronary_artery_disease
Counter({'no': 364, 'yes': 34, '\tno': 2})
************************************************************************************************************************

Columns: Bacteria
Counter({'notpresent': 378, 'present': 22})
************************************************************************************************************************

Columns: White_blood_cell_count
```

```
Counter({'9800': 116, '6700': 10, '9600': 9, '9200': 9, '7200': 9, '6900': 8, '11000': 8, '5800': 8, '7800': 7, '9100': 7, '9400'
****************************************************************************************************

Columns: Pus_cell_clumps
Counter({'notpresent': 358, 'present': 42})
****************************************************************************************************

Columns: Pedal_edema
Counter({'no': 324, 'yes': 76})
****************************************************************************************************

Columns: Appetite
Counter({'good': 318, 'poor': 82})
****************************************************************************************************
```

In [227]:

```python

```

In [228]:

```python
catcols.remove('Red_blood_cell_count')#remove is used for removing particular column
catcols.remove('Packed_cell_volume')
catcols.remove('White_blood_cell_count')
print(catcols)
```

```
{'Red_blood_cells', 'Anemia', 'Diabetesmellitus', 'Hypertension', 'Pus_cell', 'Coronary_artery_disease', 'Bacteria', 'Pus_cell_cl
```

In [229]:

```python
catscols=[ 'Class','Red_blood_cells', 'Appetite', 'Hypertension', 'pus_cell', 'Coronary_artery_disea
```

In [230]:

```python
from sklearn.preprocessing import LabelEncoder #importing 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 LabelEncoding
    print(c(data[i])) #getting the classes values before transformation
    data[i]=LEi.fit_transform(data[i])#transforming our text classes to numerical valules
    print(c(data[i])) #getting the classes values after transformation
    print("*"*100)
```

```
LABEL ENCODING OF: Red_blood_cells
Counter({'normal': 353, 'abnormal': 47})
Counter({1: 353, 0: 47})
****************************************************************************************************
LABEL ENCODING OF: Anemia
Counter({'no': 340, 'yes': 60})
Counter({0: 340, 1: 60})
****************************************************************************************************
LABEL ENCODING OF: Diabetesmellitus
Counter({'no': 260, 'yes': 134, '\tno': 3, '\tyes': 2, ' yes': 1})
Counter({3: 260, 4: 134, 0: 3, 1: 2, 2: 1})
****************************************************************************************************
LABEL ENCODING OF: Hypertension
Counter({'no': 253, 'yes': 147})
Counter({0: 253, 1: 147})
****************************************************************************************************
LABEL ENCODING OF: Pus_cell
Counter({'normal': 324, 'abnormal': 76})
Counter({1: 324, 0: 76})
****************************************************************************************************
LABEL ENCODING OF: Coronary_artery_disease
Counter({'no': 364, 'yes': 34, '\tno': 2})
Counter({1: 364, 2: 34, 0: 2})
****************************************************************************************************
LABEL ENCODING OF: Bacteria
Counter({'notpresent': 378, 'present': 22})
Counter({0: 378, 1: 22})
****************************************************************************************************
LABEL ENCODING OF: Pus_cell_clumps
Counter({'notpresent': 358, 'present': 42})
Counter({0: 358, 1: 42})
****************************************************************************************************
LABEL ENCODING OF: Pedal_edema
Counter({'no': 324, 'yes': 76})
Counter({0: 324, 1: 76})
****************************************************************************************************
LABEL ENCODING OF: Appetite
Counter({'good': 318, 'poor': 82})
Counter({0: 318, 1: 82})
****************************************************************************************************
```

In [231]:

```python
# Handling numerical columns

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

```
{'Age', 'Red_blood_cells', 'Specific_gravity', 'Sodium', 'Pus_cell', 'Coronary_artery_disease', 'Class', 'Serum_creatinine', 'Ped
```

In [232]:

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

```
Continous Columns : Age
Counter({60.0: 28, 65.0: 17, 48.0: 12, 50.0: 12, 55.0: 12, 47.0: 11, 62.0: 10, 45.0: 10, 54.0: 10, 59.0: 10, 56.0: 10, 61.0: 9, 7
************************************************************************************************************************

Continous Columns : Red_blood_cells
Counter({1: 353, 0: 47})
************************************************************************************************************************

Continous Columns : Specific_gravity
Counter({1.02: 153, 1.01: 84, 1.025: 81, 1.015: 75, 1.005: 7})
************************************************************************************************************************

Continous Columns : Sodium
Counter({137.52875399361022: 87, 135.0: 40, 140.0: 25, 141.0: 22, 139.0: 21, 142.0: 20, 138.0: 20, 137.0: 19, 136.0: 17, 150.0: 1
************************************************************************************************************************

Continous Columns : Pus_cell
Counter({1: 324, 0: 76})
************************************************************************************************************************

Continous Columns : Coronary_artery_disease
Counter({1: 364, 2: 34, 0: 2})
************************************************************************************************************************

Continous Columns : Class
Counter({0: 250, 1: 150})
************************************************************************************************************************

Continous Columns : Serum_creatinine
Counter({1.2: 40, 1.1: 24, 1.0: 23, 0.5: 23, 0.7: 22, 0.9: 22, 0.6: 18, 0.8: 17, 3.072454308093995: 17, 2.2: 10, 1.5: 9, 1.7: 9,
************************************************************************************************************************

Continous Columns : Pedal_edema
Counter({0: 324, 1: 76})
************************************************************************************************************************

Continous Columns : Blood_urea
Counter({57.425721784776904: 19, 46.0: 15, 25.0: 13, 19.0: 11, 40.0: 10, 18.0: 9, 50.0: 9, 15.0: 9, 48.0: 9, 26.0: 8, 27.0: 8, 32
************************************************************************************************************************

Continous Columns : Appetite
Counter({0: 318, 1: 82})
************************************************************************************************************************

Continous Columns : Anemia
Counter({0: 340, 1: 60})
************************************************************************************************************************

Continous Columns : Diabetesmellitus
Counter({3: 260, 4: 134, 0: 3, 1: 2, 2: 1})
************************************************************************************************************************

Continous Columns : Bacteria
Counter({0: 378, 1: 22})
************************************************************************************************************************

Continous Columns : Hemoglobin
Counter({12.526436781609195: 52, 15.0: 16, 10.9: 8, 9.8: 7, 11.1: 7, 13.0: 7, 13.6: 7, 11.3: 6, 10.3: 6, 12.0: 6, 13.9: 6, 15.4:
************************************************************************************************************************

Continous Columns : Blood_pressure
Counter({80.0: 116, 70.0: 112, 60.0: 71, 90.0: 53, 100.0: 25, 76.46907216494846: 12, 50.0: 5, 110.0: 3, 140.0: 1, 180.0: 1, 120.0
************************************************************************************************************************

Continous Columns : Albumin
Counter({0.0: 245, 1.0: 44, 2.0: 43, 3.0: 43, 4.0: 24, 5.0: 1})
************************************************************************************************************************

Continous Columns : Sugar
Counter({0.0: 339, 2.0: 18, 3.0: 14, 4.0: 13, 1.0: 13, 5.0: 3})
************************************************************************************************************************

Continous Columns : Blood glucose random
Counter({148.0365168539326: 44, 99.0: 10, 100.0: 9, 93.0: 9, 107.0: 8, 117.0: 6, 140.0: 6, 92.0: 6, 109.0: 6, 131.0: 6, 130.0: 6,
************************************************************************************************************************

Continous Columns : Potassium
Counter({4.62724358974359: 88, 5.0: 30, 3.5: 30, 4.9: 27, 4.7: 17, 4.8: 16, 4.0: 14, 4.2: 14, 4.1: 14, 3.8: 14, 3.9: 14, 4.4: 14,
************************************************************************************************************************

Continous Columns : Hypertension
Counter({0: 253, 1: 147})
************************************************************************************************************************

Continous Columns : Id
Counter({0: 1, 1: 1, 2: 1, 3: 1, 4: 1, 5: 1, 6: 1, 7: 1, 8: 1, 9: 1, 10: 1, 11: 1, 12: 1, 13: 1, 14: 1, 15: 1, 16: 1, 17: 1, 18:
************************************************************************************************************************
```

```
Continous Columns : Pus_cell_clumps
Counter({0: 358, 1: 42})
****************************************************************************************************
```

In [233]:
```
contcols.remove('Specific_gravity')
contcols.remove('Albumin')
contcols.remove('Sugar')
print(contcols)
```

```
{'Age', 'Red_blood_cells', 'Sodium', 'Pus_cell', 'Coronary_artery_disease', 'Class', 'Serum_creatinine', 'Pedal_edema', 'Blood_ur
```

In [234]:
```
contcols.add('Red_blood_cell_count')
contcols.add('Packed_cell_volume')
contcols.add('White_blood_cell_count')
print(contcols)
```

```
{'Age', 'Red_blood_cells', 'Sodium', 'Pus_cell', 'Coronary_artery_disease', 'Class', 'Serum_creatinine', 'Pedal_edema', 'Blood_ur
```

In [235]:
```
catcols.add('Specific_gravity')
catcols.add('Albumin')
catcols.add('Sugar')
print(catcols)
```

```
{'Red_blood_cells', 'Anemia', 'Albumin', 'Diabetesmellitus', 'Sugar', 'Specific_gravity', 'Hypertension', 'Pus_cell', 'Coronary_a
```

In [236]:
```
data['Diabetesmellitus'].replace(to_replace = {'\tno':'no','\tyes':'yes',' yes':'yes'},inplace=True)
c(data['Diabetesmellitus'])
```

Out [236]:
```
Counter({4: 134, 3: 260, 2: 1, 0: 3, 1: 2})
```

In [237]:
```
data['Coronary_artery_disease'] = data['Coronary_artery_disease'].replace(to_replace = '\tno', value=
c(data['Coronary_artery_disease'])
```

Out [237]:
```
Counter({1: 364, 2: 34, 0: 2})
```

In [238]:
```
data['Class'] = data['Class'].replace(to_replace = {'ckd\t': 'ckd', 'notckd': 'not ckd'})
c(data['Class'])
```

Out [238]:
```
Counter({0: 250, 1: 150})
```

In [239]:
```
def clean_dataset(df):
    assert isinstance(data, pd.DataFrame), "df needs to be a pd.DataFrame"
    data.dropna(inplace=True)
    indices_to_keep = ~data.isin([np.nan, np.inf, -np.inf]).any(axis=1)
    return data[indices_to_keep].astype(np.float64)
```

In [240]:
```
data.replace([np.inf, -np.inf], np.nan, inplace=True)
```

In [241]:
```
#Exploratory Data Analysis
 #Descriptive statistical Analysis




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

Out [241]:

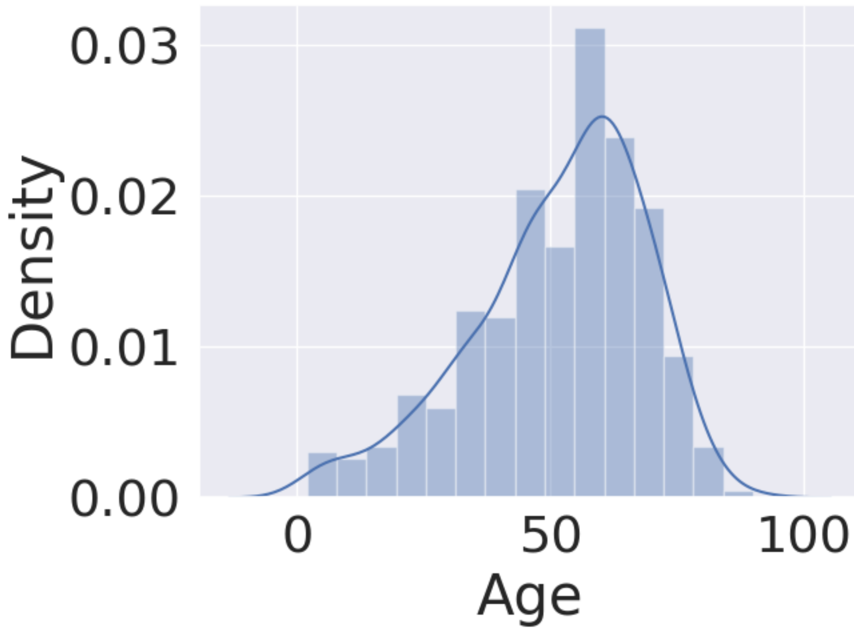
	Id	Age	Blood_pressure	Specific_gravity	Albumin	Sugar	Red_blood_cells	Pus_cell	Pus_cell_
count	400.000000	400.000000	400.000000	400.000000	400.00000	400.000000	400.000000	400.000000	400.0000
mean	199.500000	51.675000	76.469072	1.017712	0.90000	0.395000	0.882500	0.810000	0.10500C
std	115.614301	17.022008	13.476298	0.005434	1.31313	1.040038	0.322418	0.392792	0.306937
min	0.000000	2.000000	50.000000	1.005000	0.00000	0.000000	0.000000	0.000000	0.00000C
25%	99.750000	42.000000	70.000000	1.015000	0.00000	0.000000	1.000000	1.000000	0.00000C
50%	199.500000	55.000000	78.234536	1.020000	0.00000	0.000000	1.000000	1.000000	0.00000C
75%	299.250000	64.000000	80.000000	1.020000	2.00000	0.000000	1.000000	1.000000	0.00000C

	Id	Age	Blood_pressure	Specific_gravity	Albumin	Sugar	Red_blood_cells	Pus_cell	Pus_cell_
max	399.000000	90.000000	180.000000	1.025000	5.00000	5.000000	1.000000	1.000000	1.000000

8 rows × 23 columns

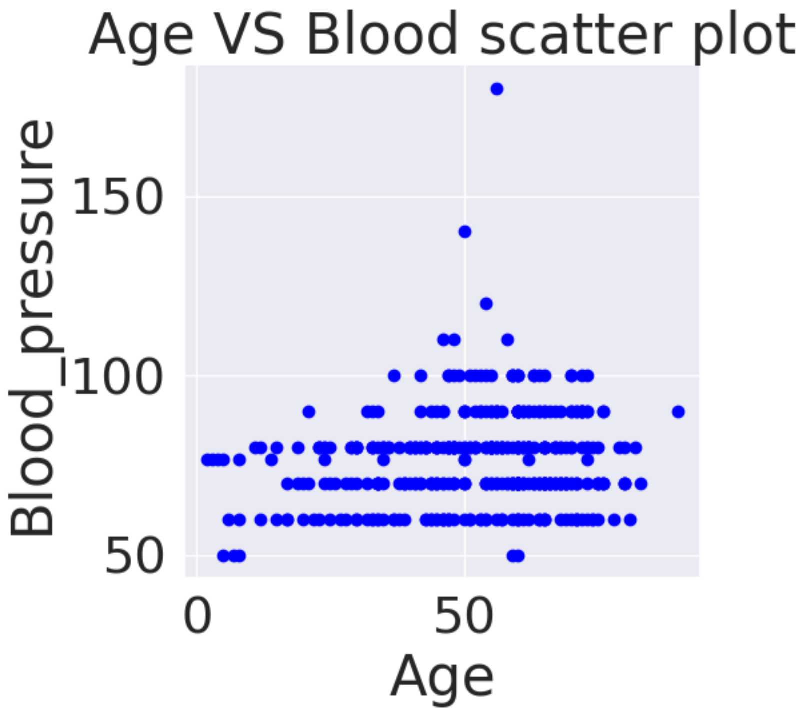
In [242]:
```
#visual analysis
# 1.Univariate analysis
#AGE DISTRIBUTION


sns.distplot(data.Age)
```

Out [242]: <Axes: xlabel='Age', ylabel='Density'>



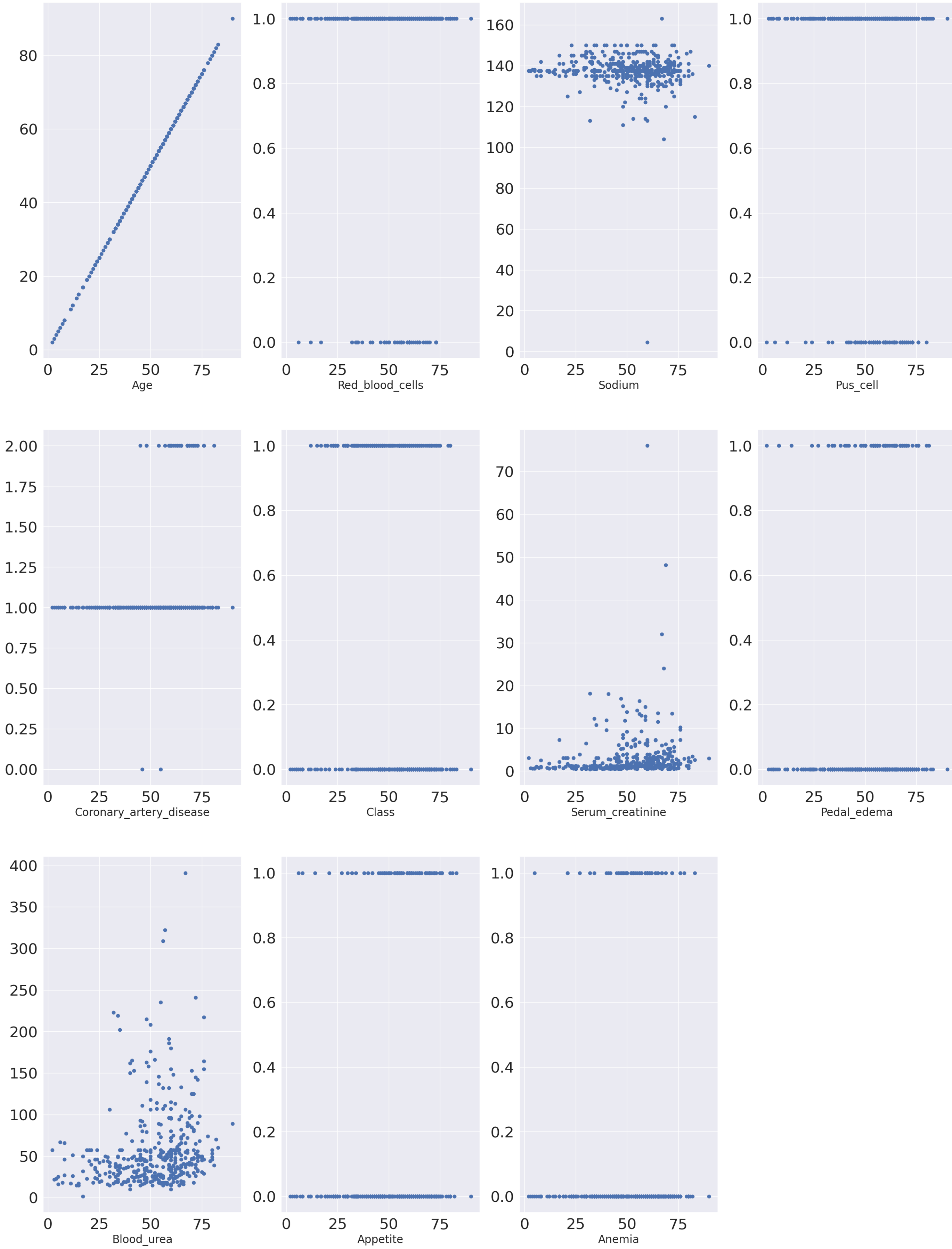
In [243]:
```
#2.Bivariate analysis
#Age vs Pressure


import matplotlib.pyplot as plt
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 labelfor y axis
plt.title("Age VS Blood scatter plot") #Set a title for the axes
```

Out [243]: Text(0.5, 1.0, 'Age VS Blood scatter plot')



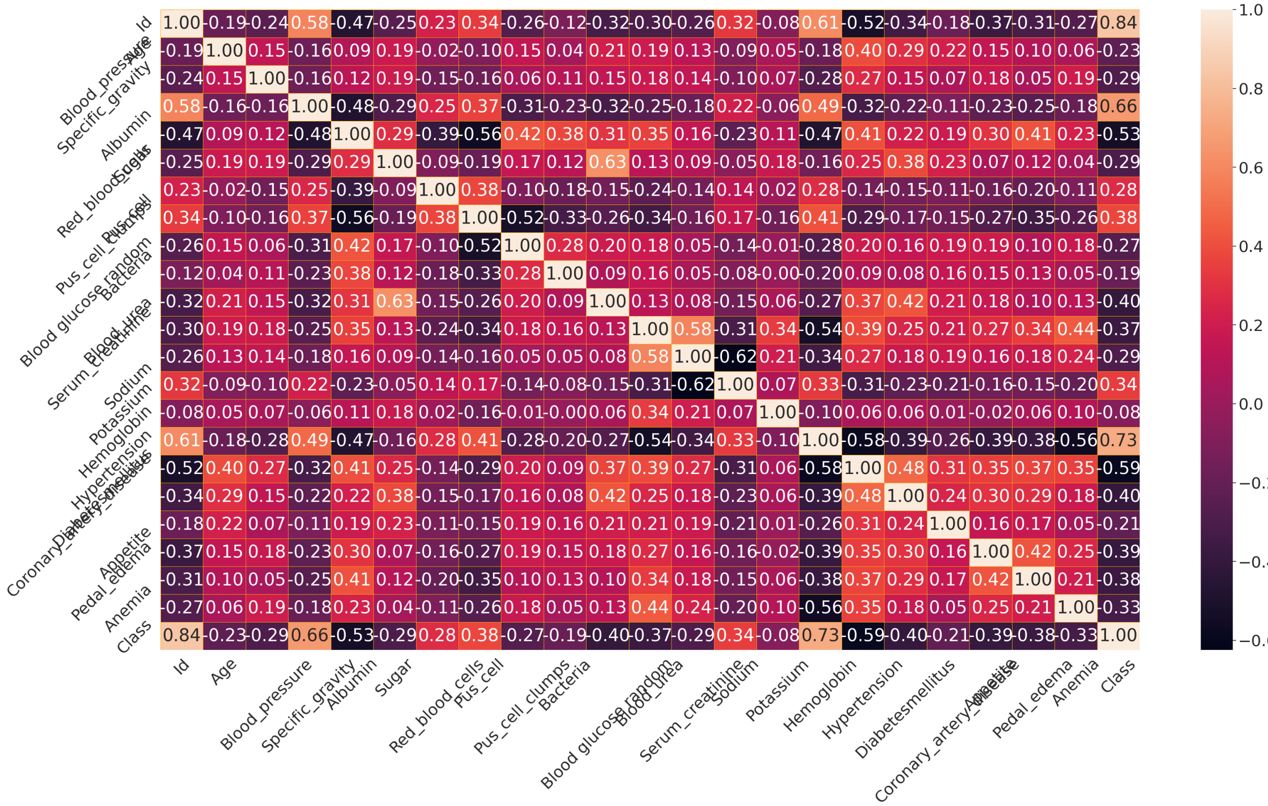
In [244]:

```
#Multivariate analysis
#Age Vs all continuous columns


plt.figure(figsize=(30,40),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)
  plotnumber+=1
plt.show()
```



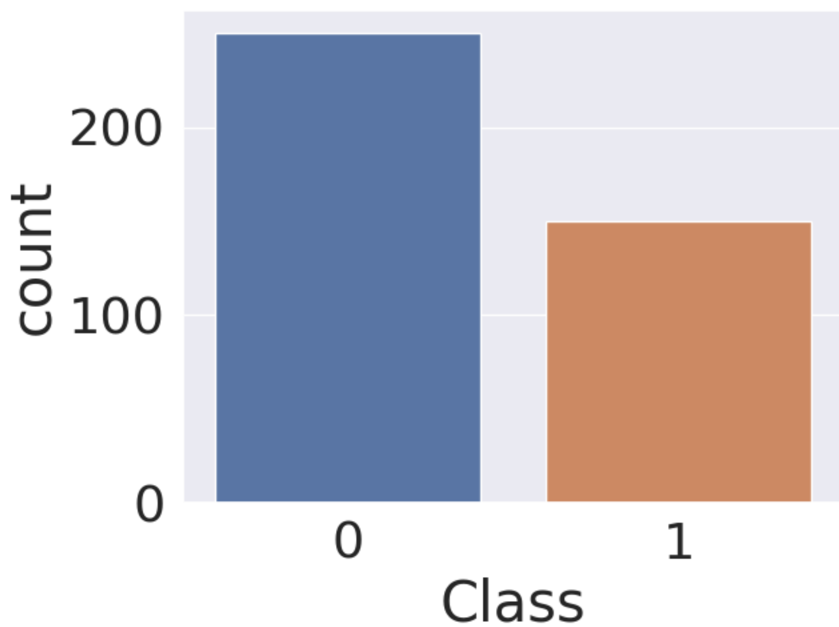
In [245]:

```
#Finding correlation between the independent columns
#HEAT MAP #correlation of parameters
f,ax=plt.subplots(figsize=(38,20))
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()
```



In [246]:
```
sns.countplot(x=data['Class'])
```

Out [246]: <Axes: xlabel='Class', ylabel='count'>



In [247]:
```
#creating Independent and Dependent


selcols=['Red_blood_cells','Pus_cell','Blood glucose random','Blood_urea','Pedal_edema','Anemia','Di
x=pd.DataFrame(data,columns=selcols)
y=pd.DataFrame(data,columns=['Class'])
```

In [248]:
```

#scaling the data
#performing feature scaling operation using standard scaller on X part of the database because
```

```
#there different type of values in the columns

from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_bal=sc.fit_transform(x)
```

In [249]:
```
print(x.shape)
print(y.shape)
```

```
(400, 8)
(400, 1)
```

In [250]:
```
#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=0)
```

In [251]:
```
#model  building
#Train the model in multiple algorithm
#ANN algorithm
#importint the keras libraries and packages


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

In [252]:
```
#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'))
```

In [253]:
```

#compiling the ANN model
classification.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

In [254]:
```
#training the model


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

```
Epoch 1/100
26/26 [==============================] - 2s 17ms/step - loss: 0.6282 - accuracy: 0.6289 - val_loss: 0.5147 - val_accuracy: 0.6094
Epoch 2/100
26/26 [==============================] - 0s 5ms/step - loss: 0.5456 - accuracy: 0.6719 - val_loss: 0.5124 - val_accuracy: 0.6250
Epoch 3/100
26/26 [==============================] - 0s 6ms/step - loss: 0.5349 - accuracy: 0.6484 - val_loss: 0.5479 - val_accuracy: 0.5938
Epoch 4/100
26/26 [==============================] - 0s 6ms/step - loss: 0.5410 - accuracy: 0.6484 - val_loss: 0.5194 - val_accuracy: 0.7344
Epoch 5/100
26/26 [==============================] - 0s 6ms/step - loss: 0.5357 - accuracy: 0.6250 - val_loss: 0.4842 - val_accuracy: 0.6875
Epoch 6/100
26/26 [==============================] - 0s 5ms/step - loss: 0.5643 - accuracy: 0.6367 - val_loss: 0.5185 - val_accuracy: 0.5938
Epoch 7/100
26/26 [==============================] - 0s 6ms/step - loss: 0.5328 - accuracy: 0.6602 - val_loss: 0.4783 - val_accuracy: 0.6250
Epoch 8/100
26/26 [==============================] - 0s 6ms/step - loss: 0.5281 - accuracy: 0.6523 - val_loss: 0.4894 - val_accuracy: 0.8125
Epoch 9/100
26/26 [==============================] - 0s 5ms/step - loss: 0.5409 - accuracy: 0.6523 - val_loss: 0.4767 - val_accuracy: 0.7500
Epoch 10/100
26/26 [==============================] - 0s 5ms/step - loss: 0.5132 - accuracy: 0.6562 - val_loss: 0.4785 - val_accuracy: 0.6094
Epoch 11/100
26/26 [==============================] - 0s 6ms/step - loss: 0.5101 - accuracy: 0.6602 - val_loss: 0.5861 - val_accuracy: 0.6250
Epoch 12/100
```

```
26/26 [==============================] - 0s 6ms/step - loss: 0.5241 - accuracy: 0.6562 - val_loss: 0.5472 - val_accuracy: 0.5938
Epoch 13/100
26/26 [==============================] - 0s 6ms/step - loss: 0.5881 - accuracy: 0.6211 - val_loss: 0.4931 - val_accuracy: 0.5938
Epoch 14/100
26/26 [==============================] - 0s 5ms/step - loss: 0.5055 - accuracy: 0.7148 - val_loss: 0.5036 - val_accuracy: 0.5938
Epoch 15/100
26/26 [==============================] - 0s 6ms/step - loss: 0.5036 - accuracy: 0.6641 - val_loss: 0.4795 - val_accuracy: 0.7500
Epoch 16/100
26/26 [==============================] - 0s 7ms/step - loss: 0.4928 - accuracy: 0.6641 - val_loss: 0.4677 - val_accuracy: 0.8281
Epoch 17/100
26/26 [==============================] - 0s 6ms/step - loss: 0.5272 - accuracy: 0.6758 - val_loss: 0.4821 - val_accuracy: 0.8438
Epoch 18/100
26/26 [==============================] - 0s 7ms/step - loss: 0.4860 - accuracy: 0.7227 - val_loss: 0.4800 - val_accuracy: 0.7812
Epoch 19/100
26/26 [==============================] - 0s 6ms/step - loss: 0.4894 - accuracy: 0.6953 - val_loss: 0.4676 - val_accuracy: 0.8281
Epoch 20/100
26/26 [==============================] - 0s 5ms/step - loss: 0.4687 - accuracy: 0.7617 - val_loss: 0.4484 - val_accuracy: 0.8594
Epoch 21/100
26/26 [==============================] - 0s 4ms/step - loss: 0.4875 - accuracy: 0.6602 - val_loss: 0.4572 - val_accuracy: 0.8594
Epoch 22/100
26/26 [==============================] - 0s 4ms/step - loss: 0.4459 - accuracy: 0.7500 - val_loss: 0.4147 - val_accuracy: 0.8281
Epoch 23/100
26/26 [==============================] - 0s 4ms/step - loss: 0.4460 - accuracy: 0.7930 - val_loss: 0.4437 - val_accuracy: 0.8438
Epoch 24/100
26/26 [==============================] - 0s 4ms/step - loss: 0.4434 - accuracy: 0.7500 - val_loss: 0.4219 - val_accuracy: 0.7969
Epoch 25/100
26/26 [==============================] - 0s 4ms/step - loss: 0.4355 - accuracy: 0.7734 - val_loss: 0.4141 - val_accuracy: 0.8438
Epoch 26/100
26/26 [==============================] - 0s 4ms/step - loss: 0.4137 - accuracy: 0.8125 - val_loss: 0.4080 - val_accuracy: 0.8750
Epoch 27/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3956 - accuracy: 0.7891 - val_loss: 0.4887 - val_accuracy: 0.7344
Epoch 28/100
26/26 [==============================] - 0s 4ms/step - loss: 0.4634 - accuracy: 0.7578 - val_loss: 0.4143 - val_accuracy: 0.7969
Epoch 29/100
26/26 [==============================] - 0s 4ms/step - loss: 0.4417 - accuracy: 0.7461 - val_loss: 0.4604 - val_accuracy: 0.7500
Epoch 30/100
26/26 [==============================] - 0s 4ms/step - loss: 0.4146 - accuracy: 0.7969 - val_loss: 0.3926 - val_accuracy: 0.8281
Epoch 31/100
26/26 [==============================] - 0s 4ms/step - loss: 0.4003 - accuracy: 0.7695 - val_loss: 0.3408 - val_accuracy: 0.8750
Epoch 32/100
26/26 [==============================] - 0s 5ms/step - loss: 0.3593 - accuracy: 0.8320 - val_loss: 0.3120 - val_accuracy: 0.8750
Epoch 33/100
26/26 [==============================] - 0s 5ms/step - loss: 0.3689 - accuracy: 0.8516 - val_loss: 0.3092 - val_accuracy: 0.8750
Epoch 34/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3440 - accuracy: 0.8594 - val_loss: 0.2926 - val_accuracy: 0.8906
Epoch 35/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3587 - accuracy: 0.8281 - val_loss: 0.2926 - val_accuracy: 0.8750
Epoch 36/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3287 - accuracy: 0.8672 - val_loss: 0.3057 - val_accuracy: 0.8281
Epoch 37/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3302 - accuracy: 0.8555 - val_loss: 0.3071 - val_accuracy: 0.8594
Epoch 38/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3256 - accuracy: 0.8594 - val_loss: 0.3304 - val_accuracy: 0.8594
Epoch 39/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3211 - accuracy: 0.8750 - val_loss: 0.4623 - val_accuracy: 0.7812
Epoch 40/100
26/26 [==============================] - 0s 5ms/step - loss: 0.4125 - accuracy: 0.7891 - val_loss: 0.3452 - val_accuracy: 0.8750
Epoch 41/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3312 - accuracy: 0.8555 - val_loss: 0.3143 - val_accuracy: 0.8594
Epoch 42/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3160 - accuracy: 0.8438 - val_loss: 0.2820 - val_accuracy: 0.8750
Epoch 43/100
26/26 [==============================] - 0s 5ms/step - loss: 0.3034 - accuracy: 0.8750 - val_loss: 0.3446 - val_accuracy: 0.8281
Epoch 44/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3809 - accuracy: 0.8047 - val_loss: 0.3193 - val_accuracy: 0.8281
Epoch 45/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3342 - accuracy: 0.8359 - val_loss: 0.3015 - val_accuracy: 0.8438
Epoch 46/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3118 - accuracy: 0.8594 - val_loss: 0.2946 - val_accuracy: 0.8750
Epoch 47/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2888 - accuracy: 0.8750 - val_loss: 0.3116 - val_accuracy: 0.8594
Epoch 48/100
26/26 [==============================] - 0s 5ms/step - loss: 0.2902 - accuracy: 0.8789 - val_loss: 0.2842 - val_accuracy: 0.8594
Epoch 49/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3188 - accuracy: 0.8633 - val_loss: 0.3377 - val_accuracy: 0.8438
Epoch 50/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2954 - accuracy: 0.8633 - val_loss: 0.3417 - val_accuracy: 0.8281
Epoch 51/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3253 - accuracy: 0.8281 - val_loss: 0.3771 - val_accuracy: 0.8281
Epoch 52/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2916 - accuracy: 0.8594 - val_loss: 0.2705 - val_accuracy: 0.8750
Epoch 53/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2945 - accuracy: 0.8516 - val_loss: 0.2867 - val_accuracy: 0.8750
Epoch 54/100
26/26 [==============================] - 0s 5ms/step - loss: 0.2710 - accuracy: 0.9062 - val_loss: 0.2729 - val_accuracy: 0.8750
Epoch 55/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2700 - accuracy: 0.8867 - val_loss: 0.2677 - val_accuracy: 0.8750
Epoch 56/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2670 - accuracy: 0.8867 - val_loss: 0.2656 - val_accuracy: 0.8750
Epoch 57/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2853 - accuracy: 0.8789 - val_loss: 0.3414 - val_accuracy: 0.8281
Epoch 58/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2647 - accuracy: 0.8906 - val_loss: 0.2611 - val_accuracy: 0.8906
Epoch 59/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2558 - accuracy: 0.8945 - val_loss: 0.2729 - val_accuracy: 0.8750
Epoch 60/100
26/26 [==============================] - 0s 5ms/step - loss: 0.2558 - accuracy: 0.8945 - val_loss: 0.3153 - val_accuracy: 0.8438
Epoch 61/100
```

```
26/26 [==============================] - 0s 4ms/step - loss: 0.2647 - accuracy: 0.8906 - val_loss: 0.2655 - val_accuracy: 0.8750
Epoch 62/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2537 - accuracy: 0.8867 - val_loss: 0.3043 - val_accuracy: 0.8594
Epoch 63/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2864 - accuracy: 0.8672 - val_loss: 0.2957 - val_accuracy: 0.8438
Epoch 64/100
26/26 [==============================] - 0s 6ms/step - loss: 0.2762 - accuracy: 0.8867 - val_loss: 0.3112 - val_accuracy: 0.8594
Epoch 65/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2813 - accuracy: 0.8711 - val_loss: 0.2921 - val_accuracy: 0.8594
Epoch 66/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3058 - accuracy: 0.8828 - val_loss: 0.3298 - val_accuracy: 0.8281
Epoch 67/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2898 - accuracy: 0.8711 - val_loss: 0.3016 - val_accuracy: 0.8438
Epoch 68/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2487 - accuracy: 0.8906 - val_loss: 0.2743 - val_accuracy: 0.8750
Epoch 69/100
26/26 [==============================] - 0s 5ms/step - loss: 0.2435 - accuracy: 0.9062 - val_loss: 0.2814 - val_accuracy: 0.8594
Epoch 70/100
26/26 [==============================] - 0s 5ms/step - loss: 0.2831 - accuracy: 0.8789 - val_loss: 0.3245 - val_accuracy: 0.8281
Epoch 71/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2970 - accuracy: 0.8828 - val_loss: 0.2566 - val_accuracy: 0.8906
Epoch 72/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2517 - accuracy: 0.8867 - val_loss: 0.2648 - val_accuracy: 0.8750
Epoch 73/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2550 - accuracy: 0.8945 - val_loss: 0.3167 - val_accuracy: 0.8750
Epoch 74/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2679 - accuracy: 0.8750 - val_loss: 0.2980 - val_accuracy: 0.8594
Epoch 75/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2489 - accuracy: 0.8984 - val_loss: 0.2644 - val_accuracy: 0.8750
Epoch 76/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2443 - accuracy: 0.8906 - val_loss: 0.2533 - val_accuracy: 0.9062
Epoch 77/100
26/26 [==============================] - 0s 5ms/step - loss: 0.2639 - accuracy: 0.8906 - val_loss: 0.3004 - val_accuracy: 0.8594
Epoch 78/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2650 - accuracy: 0.8789 - val_loss: 0.3031 - val_accuracy: 0.8438
Epoch 79/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2638 - accuracy: 0.8789 - val_loss: 0.2919 - val_accuracy: 0.8750
Epoch 80/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2350 - accuracy: 0.9062 - val_loss: 0.2627 - val_accuracy: 0.8750
Epoch 81/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2523 - accuracy: 0.8984 - val_loss: 0.2894 - val_accuracy: 0.8594
Epoch 82/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2994 - accuracy: 0.8672 - val_loss: 0.2399 - val_accuracy: 0.9062
Epoch 83/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3763 - accuracy: 0.8086 - val_loss: 0.3202 - val_accuracy: 0.8438
Epoch 84/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2847 - accuracy: 0.8906 - val_loss: 0.2968 - val_accuracy: 0.8750
Epoch 85/100
26/26 [==============================] - 0s 5ms/step - loss: 0.2441 - accuracy: 0.9102 - val_loss: 0.2505 - val_accuracy: 0.8906
Epoch 86/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2425 - accuracy: 0.8984 - val_loss: 0.4668 - val_accuracy: 0.7969
Epoch 87/100
26/26 [==============================] - 0s 6ms/step - loss: 0.3072 - accuracy: 0.8672 - val_loss: 0.3620 - val_accuracy: 0.8281
Epoch 88/100
26/26 [==============================] - 0s 11ms/step - loss: 0.2743 - accuracy: 0.8906 - val_loss: 0.2568 - val_accuracy: 0.8906
Epoch 89/100
26/26 [==============================] - 0s 5ms/step - loss: 0.2549 - accuracy: 0.9023 - val_loss: 0.3222 - val_accuracy: 0.8438
Epoch 90/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2422 - accuracy: 0.8984 - val_loss: 0.4742 - val_accuracy: 0.7500
Epoch 91/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2491 - accuracy: 0.9062 - val_loss: 0.2784 - val_accuracy: 0.8594
Epoch 92/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2369 - accuracy: 0.8984 - val_loss: 0.3183 - val_accuracy: 0.8438
Epoch 93/100
26/26 [==============================] - 0s 5ms/step - loss: 0.2362 - accuracy: 0.9023 - val_loss: 0.2985 - val_accuracy: 0.8438
Epoch 94/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3035 - accuracy: 0.8594 - val_loss: 0.2599 - val_accuracy: 0.8750
Epoch 95/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2399 - accuracy: 0.8984 - val_loss: 0.2988 - val_accuracy: 0.8594
Epoch 96/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2330 - accuracy: 0.9102 - val_loss: 0.2662 - val_accuracy: 0.8750
Epoch 97/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2506 - accuracy: 0.9023 - val_loss: 0.2378 - val_accuracy: 0.9062
Epoch 98/100
26/26 [==============================] - 0s 5ms/step - loss: 0.2688 - accuracy: 0.8906 - val_loss: 0.3041 - val_accuracy: 0.8438
Epoch 99/100
26/26 [==============================] - 0s 5ms/step - loss: 0.2562 - accuracy: 0.8906 - val_loss: 0.2662 - val_accuracy: 0.8750
Epoch 100/100
26/26 [==============================] - 0s 4ms/step - loss: 0.2209 - accuracy: 0.9141 - val_loss: 0.3824 - val_accuracy: 0.8281
```

Out [254]:
```
<keras.callbacks.History at 0x7f547babe520>
```

In [255]:
```python
#DecisionTree Classifier

from sklearn.tree import DecisionTreeClassifier
dtc=DecisionTreeClassifier(max_depth=4,splitter='best',criterion='entropy')
dtc.fit(x_train,y_train)
```

Out [255]:
```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

In [256]:
```
y_predict=dtc.predict(x_test)
y_predict
```

Out [256]:
```
array([0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1,
       0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0,
       1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1,
       0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1])
```

In [257]:
```
y_predict_train=dtc.predict(x_train)
```

In [258]:
```
#Random forest model

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

In [259]:
```
rfc.fit(x_train,y_train)
```

Out [259]:
```
RandomForestClassifier
RandomForestClassifier(criterion='entropy', n_estimators=10)
```

In [260]:
```
y_predict=rfc.predict(x_test)
```

In [261]:
```
y_predict_train=rfc.predict(x_train)
```

In [262]:
```
#Logistic regression

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

Out [262]:
```
LogisticRegression
LogisticRegression()
```

In [263]:
```
from sklearn.metrics import accuracy_score,classification_report
y_predict=lgr.predict(x_test)
```

In [264]:
```
#testing the model
#Logistic regression

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

```
[1]
```

In [265]:
```
#DecisionTree Classifier

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

```
[1]
```

In [266]:
```
#random forest classifier

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

print(y_pred)
(y_pred)
```

```
[1]
```

Out [266]:
```
array([1])
```

In [267]:
```
classification.save("ckd.h5")   #save the model to test the input
```

In [268]:
```
y_pred=classification.predict(x_test)
```

```
3/3 [==============================] - 0s 4ms/step
```

In [269]:
```
y_pred
```

Out [269]:
```
array([[3.2799813e-26],
       [8.6611879e-01],
       [8.9339346e-01],
       [4.2735275e-35],
       [0.0000000e+00],
       [6.0134321e-03],
       [8.4661645e-01],
       [8.8894916e-01],
       [9.9474786e-12],
       [9.2334367e-02],
       [4.4956473e-03],
       [8.9615196e-01],
       [3.3230895e-01],
       [8.6796945e-01],
       [9.9701312e-05],
       [7.0081555e-08],
       [8.7239134e-01],
       [6.4365529e-03],
       [2.1209590e-13],
       [3.7103656e-01],
       [3.0358048e-02],
       [9.0400612e-01],
       [3.0260345e-17],
       [8.8600874e-01],
       [7.6465434e-03],
       [8.8811946e-01],
       [1.2354474e-08],
       [0.0000000e+00],
       [8.7161559e-01],
       [8.9591187e-01],
       [1.8242787e-08],
       [8.8397735e-01],
       [8.7522346e-01],
       [1.4841768e-13],
       [5.6544667e-01],
       [3.5753965e-27],
       [7.0966971e-03],
       [8.2054991e-01],
       [4.4287202e-01],
       [8.2326061e-01],
       [2.4894029e-05],
       [9.0314025e-01],
       [8.8148874e-01],
       [8.0231309e-01],
       [8.8141328e-01],
       [9.2595261e-01],
       [9.0789807e-01],
       [8.8067216e-01],
       [1.8201415e-07],
       [8.7412709e-01],
       [3.5960027e-05],
       [3.4717080e-01],
       [8.9945745e-01],
       [1.0781690e-03],
       [8.9717191e-01],
       [8.8471144e-01],
       [2.3643085e-01],
       [4.9429693e-13],
       [8.9951378e-01],
       [4.5329035e-10],
       [3.3378345e-15],
       [8.1019086e-01],
       [8.8412100e-01],
       [8.5706699e-01],
       [1.0676155e-19],
       [9.1492683e-01],
       [0.0000000e+00],
       [7.3467463e-01],
       [4.0371278e-17],
       [8.8657379e-01],
       [2.3724805e-10],
       [8.9453310e-01],
       [9.0870289e-03],
       [1.3485392e-24],
       [3.0666674e-04],
       [9.1281188e-10],
       [9.0141940e-01],
       [8.9592367e-01],
       [8.7083334e-01],
       [8.8645744e-01]], dtype=float32)
```

In [270]:
```
y_pred=(y_pred>0.5)
y_pred
```

Out [270]:
```
array([[False],
       [ True],
       [ True],
```

```
       [False],
       [False],
       [False],
       [ True],
       [ True],
       [False],
       [False],
       [False],
       [ True],
       [False],
       [ True],
       [False],
       [False],
       [ True],
       [False],
       [False],
       [False],
       [False],
       [ True],
       [False],
       [ True],
       [False],
       [ True],
       [False],
       [False],
       [ True],
       [ True],
       [False],
       [ True],
       [ True],
       [False],
       [ True],
       [False],
       [False],
       [ True],
       [False],
       [ True],
       [False],
       [ True],
       [ True],
       [ True],
       [ True],
       [ True],
       [ True],
       [ True],
       [False],
       [ True],
       [False],
       [False],
       [ True],
       [False],
       [ True],
       [ True],
       [False],
       [False],
       [ True],
       [False],
       [False],
       [ True],
       [ True],
       [ True],
       [False],
       [ True],
       [False],
       [ True],
       [False],
       [ True],
       [False],
       [ True],
       [False],
       [False],
       [False],
       [False],
       [ True],
       [ True],
       [ True],
       [ True]])
```

In [271]:

```
def predict_exit(sample_value):
  sample_value=np.array(sample_value)
  sample_value=sample_value(1,-1)
  sample_value=sc.transform(sample_value)
  return classifier.predict(sample_value)
```

In [272]:

```
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')
```

```
1/1 [==============================] - 0s 132ms/step
Prediction:Low chance of ckd
```

In [273]:

```
#compare model
from sklearn.model_selection import train_test_split #splits data in random train and test
from sklearn.model_selection import train_test_split


from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn import model_selection
dfs=[]
models = [('LogReg',LogisticRegression()),
        ('RF',RandomForestClassifier()),
        ('DecisionTree',DecisionTreeClassifier())]
df_mean=pd.DataFrame(models)
df_mean['models'] = df_mean.index
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)
print(final)
```

```
LogReg
              precision    recall  f1-score   support

      NO CKD       0.98      0.87      0.92        52
         CKD       0.79      0.96      0.87        28

    accuracy                           0.90        80
   macro avg       0.89      0.91      0.89        80
weighted avg       0.91      0.90      0.90        80

RF
              precision    recall  f1-score   support

      NO CKD       0.96      0.88      0.92        52
         CKD       0.81      0.93      0.87        28

    accuracy                           0.90        80
   macro avg       0.89      0.91      0.89        80
weighted avg       0.91      0.90      0.90        80

DecisionTree
              precision    recall  f1-score   support

      NO CKD       0.89      0.90      0.90        52
         CKD       0.81      0.79      0.80        28

    accuracy                           0.86        80
   macro avg       0.85      0.84      0.85        80
weighted avg       0.86      0.86      0.86        80

   fit_time  score_time  test_accuracy  test_precision_weighted  \
0  0.045275    0.044287       0.921875                 0.936343
1  0.033240    0.053008       0.906250                 0.914062
2  0.028699    0.041299       0.906250                 0.908775
```

```
3   0.026642    0.042326       0.843750                0.880456
4   0.026608    0.041404       0.828125                0.827214
5   0.303171    0.086288       0.953125                0.954103
6   0.261782    0.070513       0.937500                0.945312
7   0.265278    0.075149       0.921875                0.922681
8   0.257758    0.070575       0.921875                0.937500
9   0.272198    0.078529       0.906250                0.906250
10  0.005434    0.050206       0.937500                0.938068
11  0.005088    0.039249       0.859375                0.860283
12  0.004822    0.040810       0.859375                0.859027
13  0.005381    0.056969       0.859375                0.861642
14  0.015522    0.079689       0.859375                0.858724

    test_recall_weighted  test_f1_weighted  test_roc_auc         model
0               0.921875          0.923389      0.968615        LogReg
1               0.906250          0.906618      0.931548        LogReg
2               0.906250          0.906622      0.944945        LogReg
3               0.843750          0.848958      0.929545        LogReg
4               0.828125          0.827459      0.916410        LogReg
5               0.953125          0.953363      0.994589            RF
6               0.937500          0.937745      0.949901            RF
7               0.921875          0.922050      0.958458            RF
8               0.921875          0.923862      0.925568            RF
9               0.906250          0.906250      0.960513            RF
10              0.937500          0.936739      0.919913  DecisionTree
11              0.859375          0.859618      0.895833  DecisionTree
12              0.859375          0.858986      0.860360  DecisionTree
13              0.859375          0.860282      0.843182  DecisionTree
14              0.859375          0.858830      0.848718  DecisionTree
```

In [274]:

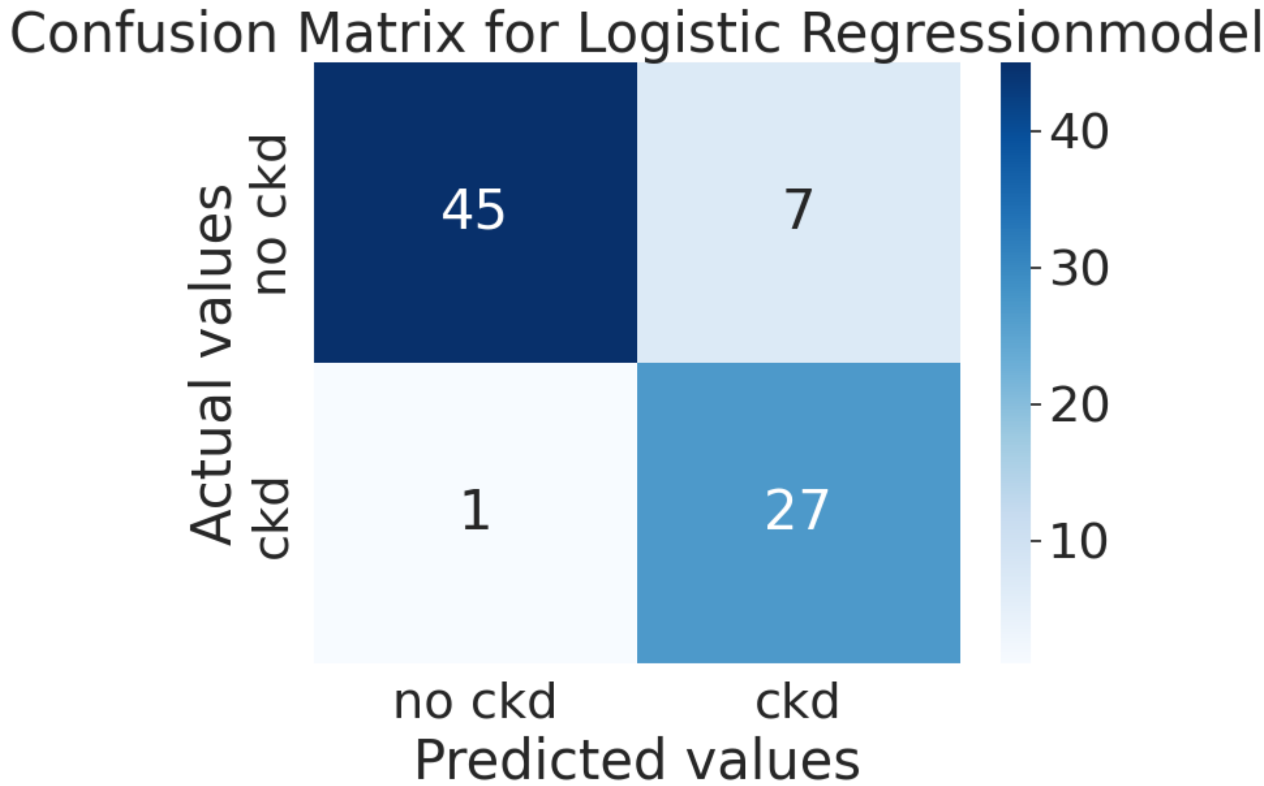
```
#Making the confusion Matrix

from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_predict)
cm
```

Out [274]:

```
array([[45,  7],
       [ 1, 27]])
```

In [275]:

```
#ploting 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 Regressionmodel')
plt.show()
```



In [276]:

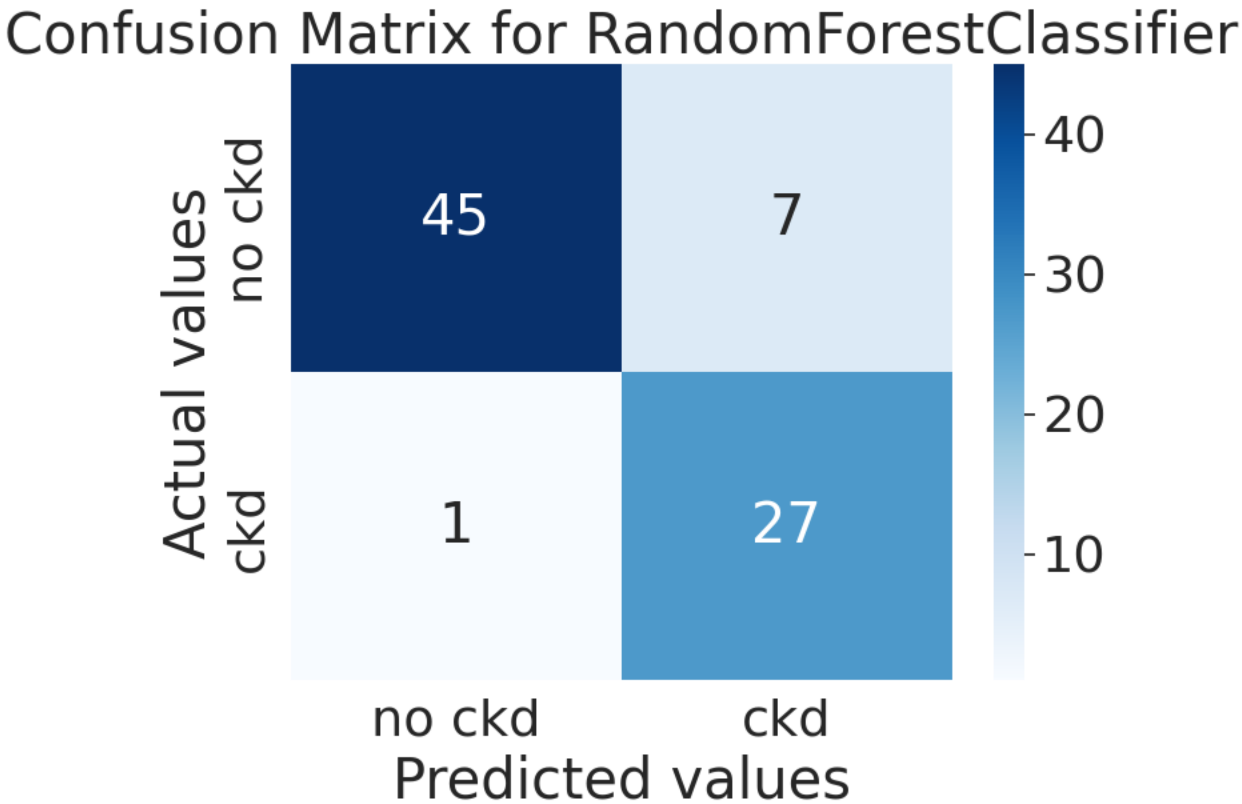
```
#Making the confusion Matrix

from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_predict)
cm
```

Out [276]:

```
array([[45,  7],
       [ 1, 27]])
```

In [277]:

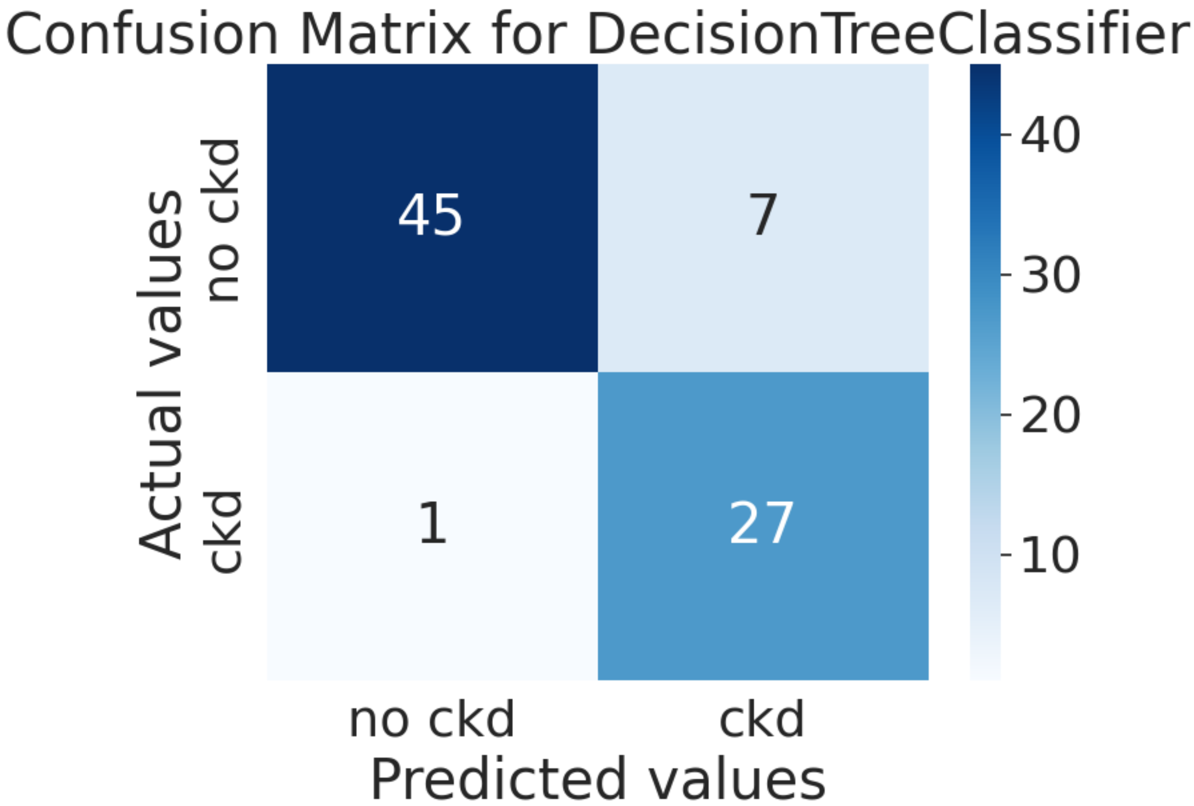
```
#ploting confusion matrix

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()
```



In [278]:
```
#Making the confusion Matrix

from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_predict)
cm
```

Out [278]:
```
array([[45,  7],
       [ 1, 27]])
```

In [279]:
```
#ploting 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()
```



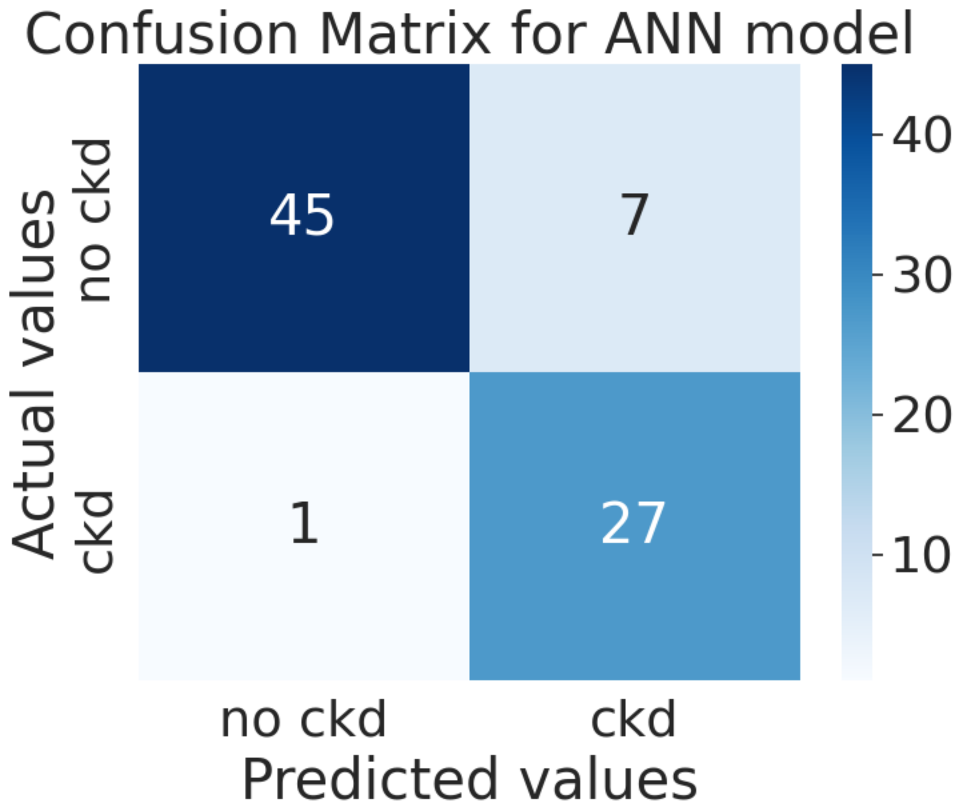
In [280]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.90      0.90        52
           1       0.81      0.79      0.80        28

    accuracy                           0.86        80
   macro avg       0.85      0.84      0.85        80
weighted avg       0.86      0.86      0.86        80
```

In [281]:
```
#Making the confusion Matrix

from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_predict)
cm
```

Out [281]:
```
array([[45,  7],
       [ 1, 27]])
```

In [282]:
```
#ploting 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()
```



In [282]:

In [283]:

```
#Evaluate the results
import pandas as pd

bootstraps=[]
bootstraps= pd.DataFrame(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,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 fi
results_long_nofit=results_long_nofit.sort_values(by='values')

#TIME METRICS

results_long_fit=results_long.loc[results_long['metrics'].isin(time_metrics)]   #get df with fit data
results_long_fit=results_long_fit.sort_values(by='values')
```

In [284]:

```
import pickle

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

In [ ]:

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(20,12))
sns.set(font_scale=2.5)
df_mean=pd.DataFrame(models)
```

```
df_mean['models'] = df_mean.index
g=sns.boxplot(x=models,y="values",hue="metrics",data=results_long_nofit,palette="Set3")
plt.legend(bbox_to_anchor=(1.05,1),loc=2,borderaxespad=0.)
plt.title("Comparision of Model by classification Metric")
plt.savefig('.\benchmark_models_performance.png',dpi=300)
```

In [ ]: