TASK 1 - CHRONIC KIDNEY DATA ANALYSIS

B PRIYANKAA

s224207694

priyayj2016@gmail.com

TASK:

The task is to create a **ML model** based on a decision tree and submit the code along with the screen shot that I got it working.

Also, needed to provide the explanation of the selected dataset, the developed code line by line and discuss the output and the performance of the built model using the selected dataset. Need to **zip the model** at the end.

TARGET GRADE : PASS(P)

1. What is the selected dataset and what is the related problem for this dataset? You need to provide details of datasets, dataset description, what are the features, output (class label) and discuss the problem that needs to be solved by machinelearning model.

This dataset has information of patients of a nearby hospital collected for a 2 months period. This dataset is used to detect the early stage of chronic kidney disease. This dataset has 400 entries having 24 feature columns and one 'id' column and one 'class' column.

The dataset is downloaded from the following link:
https://archive.ics.uci.edu/dataset/336/chronic+kidney+disease

This dataset is zipped twice and then it is extracted twice, then the file with name **chronic_kidney_disease_full.arff** is converted into csv using the below link:

https://pulipulichen.github.io/jieba-js/weka/arff2csv/

DATASET EXPLANATION:

1. The final csv file is saved with the name "ckd_data_df.csv". This dataset has 400 entries and 24 feature columns, 1 id column and 1 class column.

2. The 26 features are:

id(id), age(age), blood pressure(bp), specific gravity(sg), albumin(al), sugar(su), red blood cells(rbc), pus cell(pc), pus cell clumps(pcc), bacteria(ba), blood glucose random(bgr), blood urea(bu), serum creatinine(sc), sodium(sod), hemoglobin(hemo), packed cell volume(pcv), white blood cells count(wbcc), red blood cells count(rbcc), hypertension(htn), diabetes mellitus(dm), coronary artery disease (cad), appetite(appet), pedal edema(pe), anemia(ane), class(class)

1. The class has features like ckd, notckd that ie ckd - chronic kidney disease, notckd - not chronic kidney disease.

2. The class label with values (ckd and notckd) is the output column.

3. There are some numerical columns and categorical columns that has **?** as a value in it. This dataset needs cleaning.

4. The dataset then has to be divided into train and test and train data is set into dtree classifier. The model is tested with test data.

PROBLEM TO BE SOLVED BY THE ML MODEL:

Many patients suffer with chronic kidney disease so this dataset has features of the patients that helps us to identify the chronic kidney disease in an early stage. There will be DecisionTree model developed based on the training data and tested using the test data thats already divided in the whole dataset.

2. You need to provide the screenshot of the built ML pipeline (Data ingestion, Data preparation, model training and evaluating the model). You need to provide a cell by cell explanation of the code.

ML PIPELINE:

1. Data Ingestion ----> Data Preparation ----> Model Training ----> Model Evaluation

1. DATA INGESTION:

1. The neccessary libraries are imported.

2. Data Ingestion is the process of importing the dataset.

3. This dataset is zipped twice and then it is extracted twice, then the file with name chronic_kidney_disease_full.arff is converted into csv using the below link: https://pulipulichen.github.io/jieba-js/weka/arff2csv/

4. The csv file **ckd_data_df.csv** is stored onto the variable df.

2. DATA PREPARATION:

1. The 'id' column is dropped from the entire dataset, df.

2. The ? values are replaced by NAN values in the numerical columns of the whole dataset, df.

3. The numerical columns were of 'object' datatype and now it is converted into float64 datatype using apply(pd.to_numeric) function.

4. Imputing the NAN values with Mean values in the following columns: **'age', 'bp', 'bgr','bu', 'sc', 'sod', 'pot', 'hemo', 'pcv', 'wbcc', 'rbcc'.**

5. Imputing the NAN values with Mode values in the following Categorical columns: **'sg', 'al', 'su', 'rbc', 'pc', 'pcc','ba','htn', 'dm', 'cad', 'appet', 'pe', 'ane', 'class'.**

6. The describe() function gives the min, max, median, std and mean values of all the numerical columns.

7. Here, **Label Encoding** is done on the categorical columns in order to convert it into numerical forms. This helps us to use the data for further analysis and the dataset can be given as input to the ML models.

Correlation in the dataset:

1. df.corr() is the correlattion measure of the output label, class with respect to all the other remaining variables.

2. sns.heatmap is used to visually see the correlattion measure of the output label, class with respect to all the other remaining variables.

3. MODEL TRAINING:

1. Here X is the independent feature columns(inputs) , where dataset df without target column 'class' is set into it.

2. Here Y is the dependent column, Output label - Target column.

3. Splitting the dataset for training and testing purpose with a test_size 0.30.

4. Here the DecisionTree Classfiier is set into a variable dtree.

5. Fitting the model dtree onto the training dataset (X_train, y_train).

6. model.predict() is used to predict the class samples.

4. MODEL EVALUATION:

1. The performance metrics: accuracy and confusion matrix are used to evaluated.

Importing the necessary libraries in order to complete the task:

In [1]:
```
#Importing the necessary libraries in order to complete the task:

import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix


from sklearn.preprocessing import LabelEncoder

import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objs as go
%matplotlib inline

from sklearn.metrics import mean_squared_error

import warnings
warnings.filterwarnings("ignore")

import pickle
import zipfile
```

The column names are then splitted with a comma(,). Then the data is fed into pandas dataframe.

In [2]:
```
data = []
with open('ckd_data_df.csv', "r") as f:
    for line in f:
        line = line.replace('\n', '')
        data.append(line.split(','))


names = ['id','age', 'bp', 'sg', 'al', 'su', 'rbc', 'pc', 'pcc', 'ba',
         'bgr', 'bu',  'sc', 'sod', 'pot', 'hemo', 'pcv', 'wbcc',
         'rbcc', 'htn', 'dm', 'cad', 'appet', 'pe', 'ane',
         'class']

df = pd.DataFrame(data[1:], columns=names)
```

DATA EXPLORATION:

In [3]:
```
df.head()
```

Out[3]:

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wbcc	rbcc	htn	dn
0	1	48	80	1.02	1	0	?	normal	notpresent	notpresent	...	44	7800	5.2	yes	ye
1	2	7	50	1.02	4	0	?	normal	notpresent	notpresent	...	38	6000	?	no	no
2	3	62	80	1.01	2	3	normal	normal	notpresent	notpresent	...	31	7500	?	no	ye
3	4	48	70	1.005	4	0	normal	abnormal	present	notpresent	...	32	6700	3.9	yes	no
4	5	51	80	1.01	2	0	normal	normal	notpresent	notpresent	...	35	7300	4.6	no	no

5 rows × 26 columns

The top 5 columns are printed, there are ? values present in most of the columns.

In [4]:

```
print('The values present in rbc column are:', df['rbc'].value_counts())
print('The values present in pc column are:', df['pc'].value_counts())
print('The values present in pcc column are:', df['pcc'].value_counts())
print('The values present in ba column are:', df['ba'].value_counts())
print('The values present in htn column are:', df['htn'].value_counts())
print('The values present in dm column are:', df['dm'].value_counts())
print('The values present in cad column are:', df['cad'].value_counts())
print('The values present in class column are:', df['class'].value_counts())
print('The values present in appet column are:', df['appet'].value_counts())
print('The values present in pe column are:', df['pe'].value_counts())
print('The values present in ane column are:', df['ane'].value_counts())
print('The values present in class (output) column are:', df['class'].value_counts())
```

```
The values present in rbc column are: rbc
normal      201
?           152
abnormal     47
Name: count, dtype: int64
The values present in pc column are: pc
normal      259
abnormal     76
?            65
Name: count, dtype: int64
The values present in pcc column are: pcc
notpresent    354
present        42
?               4
Name: count, dtype: int64
The values present in ba column are: ba
notpresent    374
present        22
?               4
Name: count, dtype: int64
The values present in htn column are: htn
no     251
yes    147
?        2
Name: count, dtype: int64
The values present in dm column are: dm
no     260
yes    137
?        2
         1
Name: count, dtype: int64
The values present in cad column are: cad
no     364
yes     34
?        2
Name: count, dtype: int64
The values present in class column are: class
ckd       250
notckd    149
no          1
Name: count, dtype: int64
The values present in appet column are: appet
good    316
poor     82
?         1
no        1
Name: count, dtype: int64
The values present in pe column are: pe
no      322
yes      76
?         1
good      1
Name: count, dtype: int64
The values present in ane column are: ane
no     339
yes     60
?        1
Name: count, dtype: int64
The values present in class (output) column are: class
ckd       250
```

```
notckd    149
no          1
Name: count, dtype: int64
```

In the above columns, the values present in each categorical columna are printed.

In [5]:
```
df.drop('id', axis=1, inplace=True)
```

In [6]:
```
df.head()
```

Out[6]:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wbcc	rbcc	htn	c
0	48	80	1.02	1	0	?	normal	notpresent	notpresent	121	...	44	7800	5.2	yes	y
1	7	50	1.02	4	0	?	normal	notpresent	notpresent	?	...	38	6000	?	no	
2	62	80	1.01	2	3	normal	normal	notpresent	notpresent	423	...	31	7500	?	no	y
3	48	70	1.005	4	0	normal	abnormal	present	notpresent	117	...	32	6700	3.9	yes	
4	51	80	1.01	2	0	normal	normal	notpresent	notpresent	106	...	35	7300	4.6	no	

5 rows × 25 columns

Removing the not important 'id' column.

In [7]:
```
print('The shape of this dataset is:', df.shape)
```

```
The shape of this dataset is: (400, 25)
```

In [8]:
```
print('Columns present in the dataset are:', df.columns)
```

```
Columns present in the dataset are: Index(['age', 'bp', 'sg', 'al', 'su', 'rbc', 'p
c', 'pcc', 'ba', 'bgr', 'bu',
       'sc', 'sod', 'pot', 'hemo', 'pcv', 'wbcc', 'rbcc', 'htn', 'dm', 'cad',
       'appet', 'pe', 'ane', 'class'],
      dtype='object')
```

In [9]:
```
df.dtypes # This gives the datatype of each of the columns present.
```

Out[9]:
```
age       object
bp        object
sg        object
al        object
su        object
rbc       object
pc        object
pcc       object
ba        object
bgr       object
bu        object
sc        object
sod       object
pot       object
hemo      object
pcv       object
wbcc      object
rbcc      object
htn       object
dm        object
cad       object
appet     object
pe        object
ane       object
class     object
dtype: object
```

2.1 DATA CLEANING:

1. The **?** values are replaced by NAN values in the whole dataset, df.

In [10]:
```
df.replace('?', np.nan, inplace=True)
```

In [11]:
```
columns_to_convert = ['age', 'bp', 'bgr','bu', 'sc', 'sod', 'pot', 'hemo', 'pcv', 'wbc
df[columns_to_convert] = df[columns_to_convert].apply(pd.to_numeric)
```

The numerical columns were of 'object' datatype and now it is converted into **float64** datatype using apply(pd.to_numeric) function.

In [12]:
```
df.head()
```

Out[12]:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wbcc	rbcc	ht
0	48.0	80.0	1.02	1	0	NaN	normal	notpresent	notpresent	121.0	...	44.0	7800.0	5.2	y
1	7.0	50.0	1.02	4	0	NaN	normal	notpresent	notpresent	NaN	...	38.0	6000.0	NaN	r
2	62.0	80.0	1.01	2	3	normal	normal	notpresent	notpresent	423.0	...	31.0	7500.0	NaN	r
3	48.0	70.0	1.005	4	0	normal	abnormal	present	notpresent	117.0	...	32.0	6700.0	3.9	y
4	51.0	80.0	1.01	2	0	normal	normal	notpresent	notpresent	106.0	...	35.0	7300.0	4.6	r

5 rows × 25 columns

In [13]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 25 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   age     391 non-null    float64
 1   bp      388 non-null    float64
 2   sg      353 non-null    object 
 3   al      354 non-null    object 
 4   su      351 non-null    object 
 5   rbc     248 non-null    object 
 6   pc      335 non-null    object 
 7   pcc     396 non-null    object 
 8   ba      396 non-null    object 
 9   bgr     356 non-null    float64
 10  bu      381 non-null    float64
 11  sc      383 non-null    float64
 12  sod     313 non-null    float64
 13  pot     312 non-null    float64
 14  hemo    348 non-null    float64
 15  pcv     329 non-null    float64
 16  wbcc    294 non-null    float64
 17  rbcc    269 non-null    float64
 18  htn     398 non-null    object 
 19  dm      398 non-null    object 
 20  cad     398 non-null    object 
 21  appet   399 non-null    object 
 22  pe      399 non-null    object 
 23  ane     399 non-null    object 
 24  class   400 non-null    object 
dtypes: float64(11), object(14)
memory usage: 78.3+ KB
```

The describe() function gives the min, max, median, std and mean values of all the numerical columns.

In [14]:
```
df.describe()
```

Out[14]:

	age	bp	bgr	bu	sc	sod	pot	hemo
count	391.000000	388.000000	356.000000	381.000000	383.000000	313.000000	312.000000	348.000000
mean	51.483376	76.469072	148.036517	57.425722	3.072454	137.528754	4.627244	12.526437
std	17.169714	13.683637	79.281714	50.503006	5.741126	10.408752	3.193904	2.912587
min	2.000000	50.000000	22.000000	1.500000	0.400000	4.500000	2.500000	3.100000
25%	42.000000	70.000000	99.000000	27.000000	0.900000	135.000000	3.800000	10.300000
50%	55.000000	80.000000	121.000000	42.000000	1.300000	138.000000	4.400000	12.650000
75%	64.500000	80.000000	163.000000	66.000000	2.800000	142.000000	4.900000	15.000000
max	90.000000	180.000000	490.000000	391.000000	76.000000	163.000000	47.000000	17.800000

Inference made:

1. The average age of the patients is 51.
2. The minimum bp is 50 and maximum bp value is 180.
3. The median value for the blood glucose random(bgr) is 121.
4. The average value of blood urea among the patients as per the csv dataset is 57.4 .

isna().sum() is used to find the percentage(%) of NAN values present in all the columns of the dataset, df.

In [15]:
```
df.isna().sum() * 100 / len(df)
```

Out[15]:
```
age       2.25
bp        3.00
sg       11.75
al       11.50
su       12.25
rbc      38.00
pc       16.25
pcc       1.00
ba        1.00
bgr      11.00
bu        4.75
sc        4.25
sod      21.75
pot      22.00
hemo     13.00
pcv      17.75
wbcc     26.50
rbcc     32.75
htn       0.50
dm        0.50
cad       0.50
appet     0.25
pe        0.25
ane       0.25
class     0.00
dtype: float64
```

Imputing the NAN values with Mean values in the following columns:

'age', 'bp', 'bgr','bu', 'sc', 'sod', 'pot', 'hemo', 'pcv', 'wbcc', 'rbcc'

In [16]:
```
numeric_columns = ['age', 'bp', 'bgr','bu', 'sc', 'sod', 'pot', 'hemo', 'pcv', 'wbcc',


for col in numeric_columns:
    col_mean = df[col].mean()
    df[col].fillna(col_mean, inplace=True)
```

In [17]:
```
df.columns
```

Out[17]:
```
Index(['age', 'bp', 'sg', 'al', 'su', 'rbc', 'pc', 'pcc', 'ba', 'bgr', 'bu',
       'sc', 'sod', 'pot', 'hemo', 'pcv', 'wbcc', 'rbcc', 'htn', 'dm', 'cad',
       'appet', 'pe', 'ane', 'class'],
      dtype='object')
```

In [18]:
```
df.isna().sum() * 100 / len(df)
```

Out[18]:
```
age       0.00
bp        0.00
sg       11.75
al       11.50
su       12.25
rbc      38.00
pc       16.25
pcc       1.00
ba        1.00
bgr       0.00
bu        0.00
sc        0.00
sod       0.00
pot       0.00
hemo      0.00
pcv       0.00
wbcc      0.00
rbcc      0.00
htn       0.50
dm        0.50
cad       0.50
appet     0.25
pe        0.25
ane       0.25
class     0.00
dtype: float64
```

The nan percentage(%) in the numerical columns are reduced.

Imputing the NAN values with Mode values in the following Categorical columns:

'sg', 'al', 'su', 'rbc', 'pc', 'pcc','ba','htn', 'dm', 'cad', 'appet', 'pe', 'ane', 'class'.

In [19]:
```
categorical_columns = ['sg', 'al', 'su', 'rbc', 'pc', 'pcc','ba','htn', 'dm',
       'cad', 'appet', 'pe', 'ane', 'class']


# Replace NaN values with mean for selected columns
for col in categorical_columns:
    col_mode = df[col].mode()[0]
    df[col].fillna(col_mode, inplace=True)
```

In [20]:
```
df.isna().sum() * 100 / len(df)
```

Out[20]:
```
age      0.0
bp       0.0
sg       0.0
al       0.0
su       0.0
rbc      0.0
pc       0.0
pcc      0.0
ba       0.0
bgr      0.0
bu       0.0
sc       0.0
sod      0.0
pot      0.0
hemo     0.0
pcv      0.0
wbcc     0.0
rbcc     0.0
htn      0.0
dm       0.0
cad      0.0
appet    0.0
pe       0.0
ane      0.0
class    0.0
dtype: float64
```

In [21]:
```
print('age \n',df.age.value_counts())
print('\n')
print('bp \n',df.bp.value_counts())
print('\n')
print('bgr \n',df.bgr.value_counts())
print('\n')
print('bu \n',df.bu.value_counts())
print('\n')
print('sc \n',df.sc.value_counts())
print('\n')
print('sod \n',df.sod.value_counts())
print('\n')
print('pot \n',df.pot.value_counts())
print('\n')
print('hemo \n',df.hemo.value_counts())
print('\n')
print('pcv \n',df.pcv.value_counts())
print('\n')
print('wbcc \n',df.wbcc.value_counts())
print('\n')
print('rbcc \n',df.rbcc.value_counts())
```

```
age
 age
60.0    19
65.0    17
48.0    12
50.0    12
55.0    12
        ..
90.0     1
27.0     1
83.0     1
4.0      1
79.0     1
Name: count, Length: 77, dtype: int64


bp
 bp
80.000000     116
70.000000     112
60.000000      71
90.000000      53
100.000000     25
76.469072      12
50.000000       5
110.000000      3
140.000000      1
180.000000      1
120.000000      1
Name: count, dtype: int64


bgr
 bgr
148.036517    44
99.000000     10
93.000000      9
100.000000     9
107.000000     8
              ..
380.000000     1
288.000000     1
84.000000      1
256.000000     1
309.000000     1
Name: count, Length: 147, dtype: int64


bu
 bu
57.425722     19
46.000000     15
25.000000     13
19.000000     11
40.000000     10
              ..
322.000000     1
162.000000     1
235.000000     1
85.000000      1
```

```
165.000000     1
Name: count, Length: 119, dtype: int64


sc
 sc
1.2     40
1.1     24
0.5     23
1.0     23
0.9     22
        ..
9.2      1
13.8     1
9.7      1
24.0     1
0.4      1
Name: count, Length: 85, dtype: int64


sod
 sod
137.528754    87
135.000000    40
140.000000    25
141.000000    22
139.000000    21
142.000000    20
138.000000    20
137.000000    19
150.000000    17
136.000000    17
147.000000    13
145.000000    11
146.000000    10
132.000000    10
144.000000     9
131.000000     9
133.000000     8
130.000000     7
134.000000     6
143.000000     4
127.000000     3
124.000000     3
114.000000     2
125.000000     2
120.000000     2
113.000000     2
128.000000     2
122.000000     2
104.000000     1
129.000000     1
115.000000     1
4.500000       1
163.000000     1
111.000000     1
126.000000     1
Name: count, dtype: int64
```

```
pot
 pot
4.627244     88
3.500000     30
5.000000     30
4.900000     27
4.700000     17
4.800000     16
4.000000     14
4.100000     14
4.400000     14
3.900000     14
3.800000     14
4.200000     14
4.500000     13
4.300000     12
3.700000     12
3.600000      8
4.600000      7
3.400000      5
5.200000      5
5.700000      4
5.300000      4
6.300000      3
5.400000      3
2.900000      3
3.300000      3
5.500000      3
3.200000      3
2.500000      2
5.900000      2
5.800000      2
5.600000      2
3.000000      2
6.500000      2
7.600000      1
39.000000     1
6.400000      1
47.000000     1
5.100000      1
2.800000      1
2.700000      1
6.600000      1
Name: count, dtype: int64


hemo
 hemo
12.526437    52
15.000000    16
10.900000     8
13.600000     7
13.000000     7
             ..
6.800000      1
8.500000      1
7.300000      1
12.800000     1
17.600000     1
Name: count, Length: 116, dtype: int64
```

```
pcv
 pcv
38.884498    71
41.000000    21
52.000000    21
44.000000    19
48.000000    19
40.000000    16
43.000000    15
45.000000    13
42.000000    13
36.000000    12
33.000000    12
28.000000    12
32.000000    12
50.000000    12
37.000000    11
34.000000    11
46.000000     9
30.000000     9
29.000000     9
35.000000     9
31.000000     8
24.000000     7
39.000000     7
26.000000     6
38.000000     5
53.000000     4
51.000000     4
49.000000     4
47.000000     4
54.000000     4
25.000000     3
22.000000     3
27.000000     3
19.000000     2
23.000000     2
15.000000     1
21.000000     1
20.000000     1
17.000000     1
9.000000      1
18.000000     1
16.000000     1
14.000000     1
Name: count, dtype: int64


wbcc
 wbcc
8406.122449     106
9800.000000      11
6700.000000      10
9600.000000       9
7200.000000       9
               ...
19100.000000      1
12300.000000      1
```

```
16700.000000      1
14900.000000      1
2600.000000       1
Name: count, Length: 90, dtype: int64


rbcc
 rbcc
4.707435    131
5.200000     18
4.500000     16
4.900000     14
4.700000     11
3.900000     10
5.000000     10
4.800000     10
4.600000      9
3.400000      9
5.900000      8
3.700000      8
6.100000      8
5.500000      8
5.400000      7
5.300000      7
5.800000      7
3.800000      7
4.200000      6
4.300000      6
4.000000      6
5.600000      6
5.100000      5
6.200000      5
6.400000      5
5.700000      5
6.500000      5
4.100000      5
4.400000      5
3.200000      5
6.000000      4
3.600000      4
6.300000      4
3.300000      3
3.000000      3
3.500000      3
2.600000      2
2.800000      2
2.900000      2
2.500000      2
2.700000      2
2.100000      2
3.100000      2
2.300000      1
2.400000      1
8.000000      1
Name: count, dtype: int64
```

The values present in each of the numerical columns are printed above.

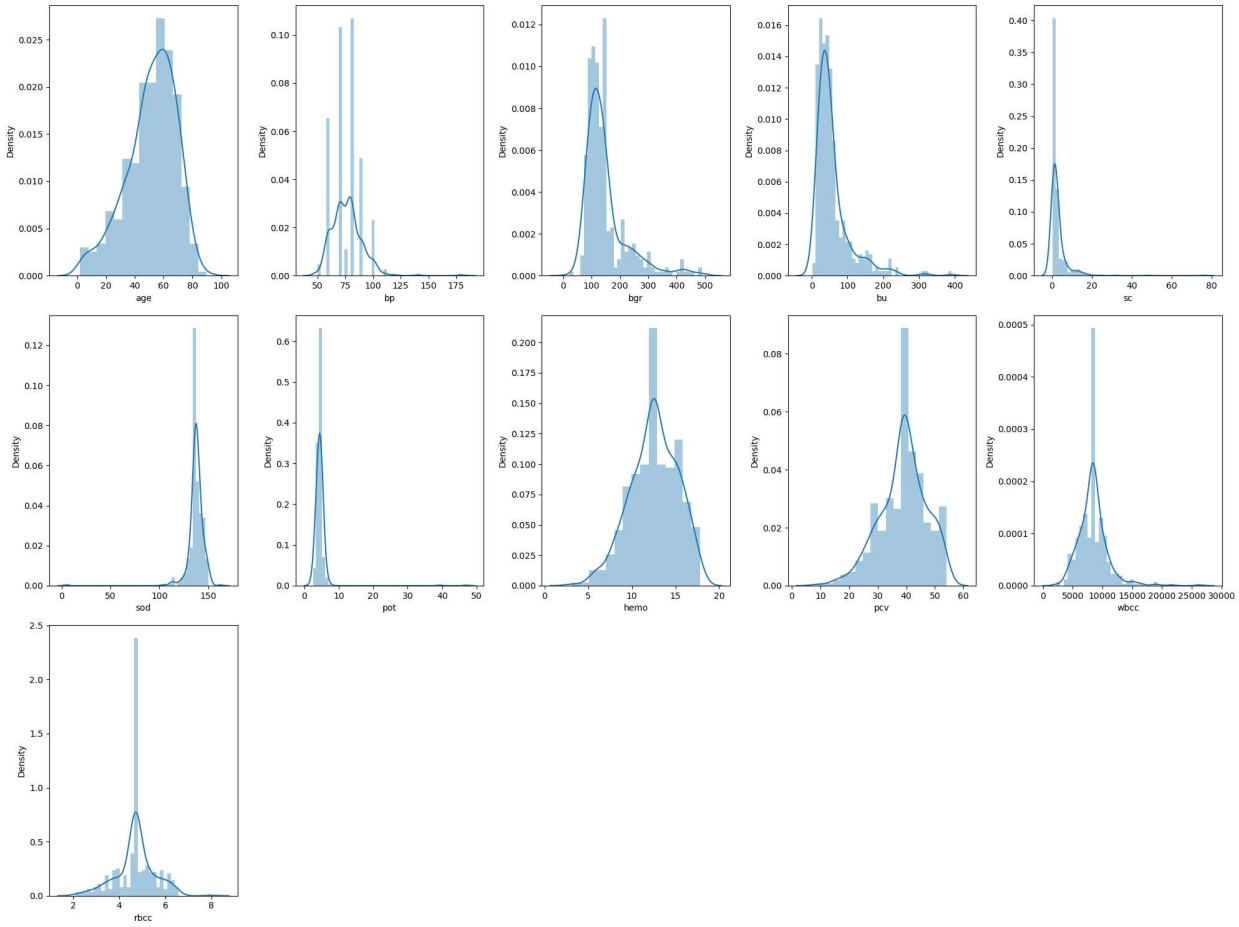
2.2 DATA VISUALISATION

In [22]:
```
plt.figure(figsize = (20, 15))
plotnumber = 1

for column in numeric_columns:
    if plotnumber <= 14:
        ax = plt.subplot(3, 5, plotnumber)
        sns.distplot(df[column])
        plt.xlabel(column)

    plotnumber += 1

plt.tight_layout()
plt.show()
```



This displot() of each of the columns depicts the variation present in the data distribution. This distribution gives better idea of how the continuous data variables are distributed along the curve.

2.3 DATA ENCODING:

Here, Label Encoding is done on the categorical columns in order to convert it into numerical forms. This helps us to use the data for further analysis and the dataset can be given as input to the ML models.

In [23]:
```
# Cleaning data and encoding
```

```
le = LabelEncoder()

def clean_data(data):
    df['sg'] = le.fit_transform(df['sg'].values)
    df['al'] = le.fit_transform(df['al'].values)
    df['su'] = le.fit_transform(df['su'].values)
    df['rbc'] = le.fit_transform(df['rbc'].values)
    df['pc'] = le.fit_transform(df['pc'].values)
    df['pcc'] = le.fit_transform(df['pcc'].values)
    df['ba'] = le.fit_transform(df['ba'].values)
    df['htn'] = le.fit_transform(df['htn'].values)
    df['dm'] = le.fit_transform(df['dm'].values)
    df['cad'] = le.fit_transform(df['cad'].values)
    df['appet'] = le.fit_transform(df['appet'].values)
    df['pe'] = le.fit_transform(df['pe'].values)
    df['ane'] = le.fit_transform(df['ane'].values)

    df['class'].replace({'ckd': 1, 'ckd\t': 1, 'notckd': 0}, inplace=True)

    df.ffill(inplace=True)

    return df
```

In [24]:
```
data = clean_data(df)
```

In [25]:
```
df.head(40)
```

Out[25]:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wbcc	
0	48.000000	80.000000	3	1	0	1	1	0	0	121.000000	...	44.000000	7800.000000	5.2
1	7.000000	50.000000	3	4	0	1	1	0	0	148.036517	...	38.000000	6000.000000	4.7
2	62.000000	80.000000	1	2	3	1	1	0	0	423.000000	...	31.000000	7500.000000	4.7
3	48.000000	70.000000	0	4	0	1	0	1	0	117.000000	...	32.000000	6700.000000	3.9
4	51.000000	80.000000	1	2	0	1	1	0	0	106.000000	...	35.000000	7300.000000	4.6
5	60.000000	90.000000	2	3	0	1	1	0	0	74.000000	...	39.000000	7800.000000	4.4
6	68.000000	70.000000	1	0	0	1	1	0	0	100.000000	...	36.000000	8406.122449	4.7
7	24.000000	76.469072	2	2	4	1	0	0	0	410.000000	...	44.000000	6900.000000	5.0
8	52.000000	100.000000	2	3	0	1	0	1	0	138.000000	...	33.000000	9600.000000	4.0
9	53.000000	90.000000	3	2	0	0	0	1	0	70.000000	...	29.000000	12100.000000	3.7
10	50.000000	60.000000	1	2	4	1	0	1	0	490.000000	...	28.000000	8406.122449	4.7
11	63.000000	70.000000	1	3	0	0	0	1	0	380.000000	...	32.000000	4500.000000	3.8
12	68.000000	70.000000	2	3	1	1	1	1	0	208.000000	...	28.000000	12200.000000	3.4
13	68.000000	70.000000	3	0	0	1	1	0	0	98.000000	...	38.884498	8406.122449	4.7
14	68.000000	80.000000	1	3	2	1	0	1	1	157.000000	...	16.000000	11000.000000	2.6
15	40.000000	80.000000	2	3	0	1	1	0	0	76.000000	...	24.000000	3800.000000	2.8
16	47.000000	70.000000	2	2	0	1	1	0	0	99.000000	...	38.884498	8406.122449	4.7
17	47.000000	80.000000	3	0	0	1	1	0	0	114.000000	...	38.884498	8406.122449	4.7
18	60.000000	100.000000	4	0	3	1	1	0	0	263.000000	...	37.000000	11400.000000	4.3
19	62.000000	60.000000	2	1	0	1	0	1	0	100.000000	...	30.000000	5300.000000	3.7
20	61.000000	80.000000	2	2	0	0	0	0	0	173.000000	...	24.000000	9200.000000	3.2
21	60.000000	90.000000	3	0	0	1	1	0	0	148.036517	...	32.000000	6200.000000	3.6
22	48.000000	80.000000	4	4	0	1	0	0	0	95.000000	...	32.000000	6900.000000	3.4
23	21.000000	70.000000	1	0	0	1	1	0	0	148.036517	...	38.884498	8406.122449	4.7
24	42.000000	100.000000	2	4	0	1	0	0	1	148.036517	...	39.000000	8300.000000	4.6
25	61.000000	60.000000	4	0	0	1	1	0	0	108.000000	...	29.000000	8400.000000	3.7
26	75.000000	80.000000	2	0	0	1	1	0	0	156.000000	...	35.000000	10300.000000	4.0
27	69.000000	70.000000	1	3	4	1	0	0	0	264.000000	...	37.000000	9600.000000	4.1
28	75.000000	70.000000	3	1	3	1	1	0	0	123.000000	...	38.884498	8406.122449	4.7
29	68.000000	70.000000	0	1	0	0	0	1	0	148.036517	...	38.000000	8406.122449	4.7
30	51.483376	70.000000	3	0	0	1	1	0	0	93.000000	...	38.884498	8406.122449	4.7
31	73.000000	90.000000	2	3	0	1	0	1	0	107.000000	...	30.000000	7800.000000	4.0
32	61.000000	90.000000	1	1	1	1	1	0	0	159.000000	...	34.000000	9600.000000	4.0
33	60.000000	100.000000	3	2	0	0	0	0	0	140.000000	...	29.000000	8406.122449	4.7

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wbcc	
34	70.000000	70.000000	1	1	0	1	1	1	1	171.000000	...	38.884498	8406.122449	4.7(
35	65.000000	90.000000	3	2	1	0	1	0	0	270.000000	...	36.000000	9800.000000	4.9(
36	76.000000	70.000000	2	1	0	1	1	0	0	92.000000	...	32.000000	8406.122449	4.7(
37	72.000000	80.000000	3	0	0	1	1	0	0	137.000000	...	28.000000	6900.000000	2.5(
38	69.000000	80.000000	3	3	0	0	1	0	0	148.036517	...	38.884498	8406.122449	4.7(
39	82.000000	80.000000	1	2	2	1	1	0	0	140.000000	...	40.000000	9800.000000	4.2(

40 rows × 25 columns

In [26]:
```
df['class'].value_counts()
```

Out[26]:
```
class
1     250
0     149
no      1
Name: count, dtype: int64
```

In [27]:
```
df.replace('no', 0, inplace=True)
print(df['class'].value_counts())
```

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

The class column had one more value 'no' in it, This 'no' is replaced by 0 for analysis purpose.

In [28]:
```
# Finding correlation of our target column

df.corrwith(df['class']).abs().sort_values(ascending=False)
```

Out[28]:
```
class    1.000000
hemo     0.729628
pcv      0.690060
sg       0.659504
rbcc     0.590913
htn      0.590438
dm       0.560649
al       0.531562
bgr      0.401374
appet    0.389211
pe       0.379163
pc       0.375154
bu       0.372033
sod      0.342288
ane      0.325396
su       0.294555
sc       0.294079
bp       0.290600
rbc      0.282642
pcc      0.265313
cad      0.236088
age      0.225405
wbcc     0.205274
ba       0.186871
pot      0.076921
dtype: float64
```

Hemoglobin, pcv, sg are in positive correlation with the class.

wbcc, ba, pot are in negative correlation with the class column.

In [29]:
```
corr_matrix = df.corr().abs()
corr_matrix.style.background_gradient(cmap='Pastel1_r')
```

Out[29]:

	age	bp	sg	al	su	rbc	pc	pcc	ba	
age	1.000000	0.148004	0.159073	0.084416	0.187615	0.011783	0.101951	0.159074	0.043573	0.214
bp	0.148004	1.000000	0.164422	0.122541	0.190218	0.151369	0.156856	0.059560	0.112173	0.149
sg	0.159073	0.164422	1.000000	0.479962	0.292053	0.253894	0.365353	0.306426	0.231704	0.317
al	0.084416	0.122541	0.479962	1.000000	0.287751	0.394844	0.561713	0.417868	0.377935	0.310
su	0.187615	0.190218	0.292053	0.287751	1.000000	0.092940	0.190062	0.168091	0.119399	0.629
rbc	0.011783	0.151369	0.253894	0.394844	0.092940	1.000000	0.377394	0.102948	0.184402	0.153
pc	0.101951	0.156856	0.365353	0.561713	0.190062	0.377394	1.000000	0.520118	0.330401	0.262
pcc	0.159074	0.059560	0.306426	0.417868	0.168091	0.102948	0.520118	1.000000	0.275082	0.197
ba	0.043573	0.112173	0.231704	0.377935	0.119399	0.184402	0.330401	0.275082	1.000000	0.085
bgr	0.214410	0.149100	0.317893	0.310481	0.629809	0.153076	0.262259	0.197593	0.085940	1.000
bu	0.187544	0.183970	0.249370	0.346935	0.126043	0.236322	0.344048	0.184415	0.158444	0.127
sc	0.127316	0.144359	0.176146	0.160252	0.094565	0.138394	0.157896	0.049940	0.050830	0.082
sod	0.085949	0.103220	0.217473	0.228076	0.053452	0.140572	0.173323	0.142135	0.081733	0.154
pot	0.050148	0.066648	0.063324	0.111614	0.180067	0.018192	0.158750	0.006316	0.002688	0.056
hemo	0.175380	0.279535	0.492143	0.474211	0.156876	0.280990	0.411500	0.275763	0.204954	0.269
pcv	0.211805	0.292714	0.501064	0.475165	0.181518	0.280958	0.418580	0.294242	0.189822	0.267
wbcc	0.100061	0.026067	0.206884	0.207303	0.159034	0.002207	0.107886	0.163456	0.103546	0.121
rbcc	0.201051	0.220822	0.443741	0.411122	0.163871	0.202455	0.383240	0.267982	0.192222	0.222
htn	0.393440	0.270447	0.323643	0.406057	0.254268	0.140538	0.291719	0.195623	0.089046	0.369
dm	0.352887	0.228657	0.349983	0.308845	0.428534	0.146209	0.201708	0.165502	0.080591	0.498
cad	0.232951	0.086618	0.135814	0.200957	0.229301	0.111493	0.172295	0.188029	0.162395	0.212
appet	0.161980	0.176053	0.229592	0.300918	0.068015	0.159684	0.273392	0.188563	0.148329	0.174
pe	0.084652	0.056517	0.253667	0.410882	0.117558	0.199391	0.349426	0.105367	0.134770	0.103
ane	0.050567	0.194962	0.184155	0.229556	0.042464	0.107625	0.260566	0.175861	0.052208	0.126
class	0.225405	0.290600	0.659504	0.531562	0.294555	0.282642	0.375154	0.265313	0.186871	0.401

MODEL TRAINING:

1. From the sklearn, DecisionTreeClassifier model is trained based on the training data - 70%.

2. There are various parameters with respect to DecisionTreeClassifier, they are:

-- criterion = gini (default)

-- splitter = best (default)

-- max_depth = None which means nodes are split until all leaves are pure.

In [30]:
```
# Here X is the independent feature columns, where dataset df without target column 'c

X = df.drop(['class'], axis=1) # Independent feature columns - Input
y = df['class'] # Output label - Target column.
```

In [31]:
```
# Split the dataset for training and testing purpose:

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state
```

Here 70% dataset is used for training purpose and remaining 30% is used for testing purpose.

In [32]:
```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier

## Necessary library to import the DecisionTree Classifier is imported.
```

In [33]:
```
dtree = DecisionTreeClassifier(random_state=1)
```

The DecisionTreeClassifier model is set with random_state = 1 into the dtree variable.

Fitting the model dtree onto the training dataset (X_train, y_train).

In [34]:
```
dtree.fit(X_train, y_train)
```

Out[34]:
```
DecisionTreeClassifier(random_state=1)
```

3. What is the performance of the build model/ models (Based on your target grade)? You need to provide discussion and justification of how the model is performing (discuss different metrics like accuracy, confusion matrix, etc.) based on the selected dataset.

1. The performance metrics are :

Accuracy Confusion Matrix Classification report - precision,recall,f1_score

1. Accuracy on Training Data is 100% where accuracy on testing data is 95.8%

2. The classification report gives the precision value, f1_score and recall value:

Precision: The class ckd - 1 is predicted correctly for about 97%. The class notckd - 0 is predicted correctly for about 94%.

There are total of 69 positives (TP+FP) being predicted, out of which only TP(True positives) are taken into account - **Precision**

Recall - It is the value of how our model predicted correctly. It must be high for a model. Here it is 96%.

f1 - score - It combines both the Precision and Recall value.

1. The confusion matrix is a matrix of actual and predicted labels 0 and 1 respectively. There are 4 values that are calculated, they are:

True Positives: The predicted value is 1 and the actual value is also 1. Here it is 67.

False Positives: The predicted value is 1 and the actual value is also 0. Here it is 2.

True Negatives: The predicted value is 0 and the actual value is also 0. Here it is 48.

False Negatives: The predicted value is 0 and the actual value is also 1. Here it is 3.

In [42]:
```
print('Accuracy on Training Data',dtree.score(X_train,y_train))
print('\n')
print('Accuracy on Testing Data',dtree.score(X_test,y_test))
```

```
Accuracy on Training Data 1.0


Accuracy on Testing Data 0.9583333333333334
```

In [35]:
```
predictions = dtree.predict(X_test) # This is used to predict the class samples

print(predictions)
```

```
[0 1 1 0 1 0 1 0 1 0 1 0 1 1 0 1 0 0 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 1 1 1
 1 0 1 0 1 1 1 0 1 0 0 0 0 1 1 1 1 0 0 0 1 0 0 1 1 1 1 0 1 1 0 0 0 1 0 0 0
 1 0 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 0 0 0 1 0 0 0 0 1 1 0 1 1 1 0 1 1 0 0 1
 0 1 1 1 1 0 1 0 0]
```

Based on the training done on the X_train, y_train dataset: The output of the X_test is printed above.

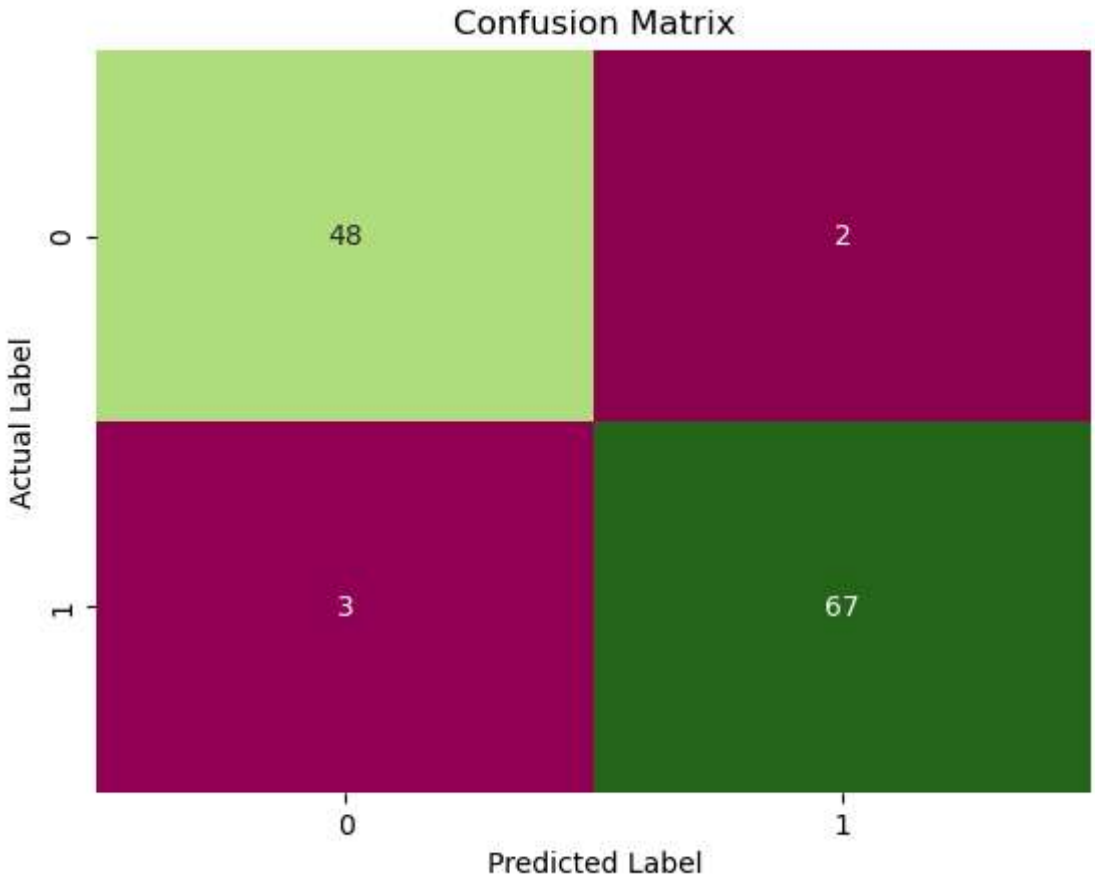
In [36]:
```
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.94      0.96      0.95        50
           1       0.97      0.96      0.96        70

    accuracy                           0.96       120
   macro avg       0.96      0.96      0.96       120
weighted avg       0.96      0.96      0.96       120
```

In [37]:
```
sns.heatmap(confusion_matrix(y_test,predictions),annot=True, fmt='d', cbar=False,cmap=
plt.xlabel('Predicted Label')
plt.ylabel('Actual Label')
plt.title('Confusion Matrix')
plt.show()
```



In [39]:
```
# Exporting the model

f = open('model.pkl', 'wb')
pickle.dump(dtree, f)
f.close()
```

In [40]:
```
zipfile.ZipFile('model.zip', mode = 'w').write('model.pkl')
```

The dtree model is pickled into a variable **model.pkl.**

Then the model.pkl is zipped into a zipped folder **model.zip**

In [ ]:

THE END