

STATS 3DA3

Homework Assignment 6

Shujia Zhang(400390028), Yunxin Li(400323756), Ling Dai(400220798)

2024-04-17

(Rubini and Eswaran 2015)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import mean_squared_error, confusion_matrix, classification_report
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, plot_tree
from patsy import dmatrices, dmatrix
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
```

```
pip install ucimlrepo
```

Requirement already satisfied: ucimlrepo in /Users/shujiazhang/anaconda3/envs/proj02/lib/python

Note: you may need to restart the kernel to use updated packages.

```
from ucimlrepo import fetch_ucirepo

# fetch dataset
chronic_kidney_disease = fetch_ucirepo(id=336)

# data (as pandas dataframes)
X = chronic_kidney_disease.data.features
y = chronic_kidney_disease.data.targets

# metadata
print(chronic_kidney_disease.metadata)
```

```
# variable information
```

```
print(chronic_kidney_disease.variables)
```

```
{'uci_id': 336, 'name': 'Chronic Kidney Disease', 'repository_url': 'https://archive.ics.uci.edu/dataset/336/chronic_kidney_disease'}
```

	name	role	type	demographic	description \
0	age	Feature	Integer	Age	None
1	bp	Feature	Integer	None	blood pressure
2	sg	Feature	Categorical	None	specific gravity
3	al	Feature	Categorical	None	albumin
4	su	Feature	Categorical	None	sugar
5	rbc	Feature	Binary	None	red blood cells
6	pc	Feature	Binary	None	pus cell
7	pcc	Feature	Binary	None	pus cell clumps
8	ba	Feature	Binary	None	bacteria
9	bgr	Feature	Integer	None	blood glucose random
10	bu	Feature	Integer	None	blood urea
11	sc	Feature	Continuous	None	serum creatinine
12	sod	Feature	Integer	None	sodium
13	pot	Feature	Continuous	None	potassium
14	hemo	Feature	Continuous	None	hemoglobin
15	pcv	Feature	Integer	None	packed cell volume
16	wbcc	Feature	Integer	None	white blood cell count
17	rbcc	Feature	Continuous	None	red blood cell count
18	htn	Feature	Binary	None	hypertension
19	dm	Feature	Binary	None	diabetes mellitus
20	cad	Feature	Binary	None	coronary artery disease
21	appet	Feature	Binary	None	appetite
22	pe	Feature	Binary	None	pedal edema
23	ane	Feature	Binary	None	anemia
24	class	Target	Binary	None	ckd or not ckd

```
units missing_values
```

0	year	yes
1	mm/Hg	yes
2	None	yes
3	None	yes
4	None	yes
5	None	yes
6	None	yes
7	None	yes
8	None	yes
9	mgs/dl	yes
10	mgs/dl	yes
11	mgs/dl	yes
12	mEq/L	yes
13	mEq/L	yes
14	gms	yes
15	None	yes
16	cells/cmm	yes
17	millions/cmm	yes
18	None	yes
19	None	yes
20	None	yes
21	None	yes
22	None	yes
23	None	yes
24	None	no

X

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	hemo	pcv	wbcc	rbcc	htn	dm	cad	appet	p
0	48.0	80.0	1.020	1.0	0.0	NaN			normal			notpresent	notpresent	121.0	...	15.4	44.0	78		
1	7.0	50.0	1.020	4.0	0.0	NaN			normal			notpresent	notpresent	NaN	...	11.3	38.0	60		
2	62.0	80.0	1.010	2.0	3.0	normal			normal			notpresent	notpresent	423.0	...	9.6	31.0	75		

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	hemo	pcv	wbcc	rbcc	htn	dm	cad	appet	p
3	48.0	70.0	1.005	4.0	0.0	0.0	normal	abnormal				present		notpresent		117.0	...	11.2	32.0	67
4	51.0	80.0	1.010	2.0	0.0	0.0	normal	normal				notpresent		notpresent		106.0	...	11.6	35.0	73
...
395	55.0	80.0	1.020	0.0	0.0	0.0	normal	normal				notpresent		notpresent		140.0	...	15.7	47.0	67
396	42.0	70.0	1.025	0.0	0.0	0.0	normal	normal				notpresent		notpresent		75.0	...	16.5	54.0	78
397	12.0	80.0	1.020	0.0	0.0	0.0	normal	normal				notpresent		notpresent		100.0	...	15.8	49.0	60
398	17.0	60.0	1.025	0.0	0.0	0.0	normal	normal				notpresent		notpresent		114.0	...	14.2	51.0	72
399	58.0	80.0	1.025	0.0	0.0	0.0	normal	normal				notpresent		notpresent		131.0	...	15.8	53.0	68

y

	class
0	ckd
1	ckd
2	ckd
3	ckd
4	ckd
...	...
395	notckd
396	notckd
397	notckd
398	notckd
399	notckd

X.dtypes

```
age      float64
bp       float64
sg       float64
al       float64
```

```
su      float64
rbc      object
pc      object
pcc      object
ba      object
bgr      float64
bu      float64
sc      float64
sod      float64
pot      float64
hemo     float64
pcv      float64
wbcc     float64
rbcc     float64
htn      object
dm      object
cad      object
appet    object
pe      object
ane      object
dtype: object
```

```
y.dtypes
```

```
class      object
dtype: object
```

```
#to see if "rbc" is binary, since from the website information I found that this variable should be binary
X['rbc'].value_counts()
```

```
rbc
normal      201
```

```
abnormal      47
```

```
Name: count, dtype: int64
```

```
#to see if "pc" is binary, since from the website information I found that this variable should
```

```
X['pc'].value_counts()
```

```
pc
```

```
normal      259
```

```
abnormal     76
```

```
Name: count, dtype: int64
```

```
#to see if "pcc" is binary, since from the website information I found that this variable should
```

```
X['pcc'].value_counts()
```

```
pcc
```

```
notpresent   354
```

```
present      42
```

```
Name: count, dtype: int64
```

```
#to see if "ba" is binary, since from the website information I found that this variable should
```

```
X['ba'].value_counts()
```

```
ba
```

```
notpresent   374
```

```
present      22
```

```
Name: count, dtype: int64
```

```
#to see if "htn" is binary, since from the website information I found that this variable should
```

```
X['htn'].value_counts()
```

```
htn
```

```
no      251
```

```
yes     147
```

```
Name: count, dtype: int64
```

```
#to see if "dm" is binary, since from the website information I found that this variable should
X['dm'].value_counts()
```

```
dm
no      260
yes     137
\tno      1
Name: count, dtype: int64
```

```
#find that "dm" is not binary, but it should be binary.
X_cat = set(X["dm"])
X_cat
```

```
{'\tno', nan, 'no', 'yes'}
```

```
#make "dm" become binary
X['dm'] = X['dm'].replace('\tno', 'no')
X_cat = set(X["dm"])
X_cat
```

```
/var/folders/t7/fhrwqn1s6810tvdnfvdsd69g00000gn/T/ipykernel_63771/1663051033.py:2: SettingWithC
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/
X['dm'] = X['dm'].replace('\tno', 'no')
```

```
{nan, 'no', 'yes'}
```

```
#to double check if "dm" is binary now.
X['dm'].value_counts()
```



```
dm
no      261
yes     137
Name: count, dtype: int64
```

```
#to see if "cad" is binary, since from the website inormation I found that this variable should
X['cad'].value_counts()
```

```
cad
no      364
yes     34
Name: count, dtype: int64
```

```
#to see if "appet" is binary, since from the website inormation I found that this variable should
X['appet'].value_counts()
```

```
appet
good    317
poor     82
Name: count, dtype: int64
```

```
#to see if "pe" is binary, since from the website inormation I found that this variable should
X['pe'].value_counts()
```

```
pe
no      323
yes     76
Name: count, dtype: int64
```

```
#to see if "ane" is binary, since from the website inormation I found that this variable should
X['ane'].value_counts()
```

```
ane
no      339
yes      60
Name: count, dtype: int64
```

```
#to see if "class" is binary, since from the website information I found that this variable should be binary
y['class'].value_counts()
```

```
class
ckd      248
notckd    150
ckd\t      2
Name: count, dtype: int64
```

```
#find "class" is not binary, but it should be binary.
y_cat = set(y["class"])
y_cat
```

```
{'ckd', 'ckd\t', 'notckd'}
```

```
# make it become binary
y['class'] = y['class'].replace('ckd\t', 'ckd')
y_cat = set(y["class"])
y_cat
```

```
/var/folders/t7/fhrwqn1s6810tvdnfvsd69g00000gn/T/ipykernel_63771/3205154435.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/10min.html#setting-with-copy
y['class'] = y['class'].replace('ckd\t', 'ckd')
```

```
{'ckd', 'notckd'}
```

```
#To double check if it is binary now.
```

```
y['class'].value_counts()
```

```
class
```

```
ckd      250
```

```
notckd   150
```

```
Name: count, dtype: int64
```

```
# number of missing values in X
```

```
X.isnull().sum()
```

```
age      9
```

```
bp       12
```

```
sg       47
```

```
al       46
```

```
su       49
```

```
rbc     152
```

```
pc       65
```

```
pcc      4
```

```
ba       4
```

```
bgr     44
```

```
bu       19
```

```
sc       17
```

```
sod     87
```

```
pot     88
```

```
hemo     52
```

```
pcv      71
```

```
wbcc    106
```

```
rbcc    131
```

```
htn      2
```

```
dm       2
```

```
cad      2
```

```
appet      1
pe         1
ane        1
dtype: int64
```

```
# number of missing values in y
y.isnull().sum()
```

```
class      0
dtype: int64
```

1. **Classification Problem Identification:** Define and describe a classification problem based on the dataset.

There are 400 observations and 25 variables in Chronic Kidney Disease dataset. There are 14 float64 type variables, and they are “age”, “bp”, “sg”, “al”, “su”, “bgr”, “bu”, “sc”, “sod”, “pot”, “hemo”, “pcv”, “wbcc”, “rbcc”. There are 11 object type variables, and they are “rbc”, “pc”, “pcc”, “ba”, “htn”, “dm”, “cad”, “appet”, “pe”, “ane”, “calss”. In these 25 variables, all variables in X are covariates (consists 14 floats and 10 object) and the variable in y is the response (the variable “class” which is an object variable). Moreover, the response is “class” contains only 2 category: “ckd” or “not ckd”, so the response is binary.

There are 9 missing values in “age”, 12 missing values in “bp”, 47 missing values in “sg”, 46 missing values in “al”, 49 missing values in “su”, 152 missing values in “rbc”, 65 missing values in “pc”, 4 missing values in “pcc”, 4 missing values in “ba”, 44 missing values in “bgr”, 19 missing values in “bu”, 17 missing values in “sc”, 87 missing values in “sod”, 88 missing values in “pot”, 52 missing values in “hemo”, 71 missing values in “pcv”, 106 missing values in “wbcc”, 131 missing values in “rbcc”, 2 missing values in “htn”, 2 missing values in “dm”, 2 missing values in “cad”, 1 missing value in “appet”, 1 missing value in “pe” and 1 missing value in “ane”.

0 missing value in “class”.

2. **Variable Transformation:** Implement any transformations chosen or justify the absence of such modifications.

```

X1 = X.copy()
columns_to_convert = ['sg', 'al', 'su']
for col in columns_to_convert:
    X1[col] = pd.Categorical(X1[col])

object_convert = ["rbc", "pc", "pcc", "ba", "htn", "dm", "cad", "appet", "pe", "ane"]
for col in object_convert:
    X1[col] = X1[col].astype('category').cat.codes
X1.dtypes

```

```

age      float64
bp       float64
sg       category
al       category
su       category
rbc      int8
pc       int8
pcc      int8
ba       int8
bgr      float64
bu       float64
sc       float64
sod      float64
pot      float64
hemo     float64
pcv      float64
wbcc     float64
rbcc     float64
htn      int8
dm       int8
cad      int8
appet    int8

```

pe int8

ane int8

dtype: object

```
#cat = ['sg', 'al', 'su', 'rbc', 'pc', 'pcc', 'ba', 'htn', 'dm', 'cad', 'appet', 'pe', 'ane']
#X = pd.get_dummies(X,columns=cat)
#caty = ['class']
#y = pd.get_dummies(y,columns=caty)
```

X1

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	hemo	pcv	wbcc	rbcc	htn	dm	cad	appet	p
0	48.0	80.0	1.020	1.0	0.0	-1	1	0	0	121.0	...	15.4	44.0	7800.0	5.2	1	1	0	0	0
1	7.0	50.0	1.020	4.0	0.0	-1	1	0	0	NaN	...	11.3	38.0	6000.0	NaN	0	0	0	0	0
2	62.0	80.0	1.010	2.0	3.0	1	1	0	0	423.0	...	9.6	31.0	7500.0	NaN	0	1	0	0	0
3	48.0	70.0	1.005	4.0	0.0	1	0	1	0	117.0	...	11.2	32.0	6700.0	3.9	1	0	0	0	0
4	51.0	80.0	1.010	2.0	0.0	1	1	0	0	106.0	...	11.6	35.0	7300.0	4.6	0	0	0	0	0
...
395	55.0	80.0	1.020	0.0	0.0	1	1	0	0	140.0	...	15.7	47.0	6700.0	4.9	0	0	0	0	0
396	42.0	70.0	1.025	0.0	0.0	1	1	0	0	75.0	...	16.5	54.0	7800.0	6.2	0	0	0	0	0
397	12.0	80.0	1.020	0.0	0.0	1	1	0	0	100.0	...	15.8	49.0	6600.0	5.4	0	0	0	0	0
398	17.0	60.0	1.025	0.0	0.0	1	1	0	0	114.0	...	14.2	51.0	7200.0	5.9	0	0	0	0	0
399	58.0	80.0	1.025	0.0	0.0	1	1	0	0	131.0	...	15.8	53.0	6800.0	6.1	0	0	0	0	0

X1['sg'].value_counts()

sg

1.020 106

1.010 84

1.025 81

1.015 75

1.005 7

Name: count, dtype: int64

```
#why 3 categories?
```

```
X1['rbc'].value_counts()
```

```
rbc
```

```
1    201
-1   152
0     47
```

```
Name: count, dtype: int64
```

```
X1['pe'].value_counts()
```

```
pe
```

```
0    323
1     76
-1     1
```

```
Name: count, dtype: int64
```

```
float_col = X1.select_dtypes(include=['float64']).columns
```

```
object_col = X1.select_dtypes(include=['object']).columns
```

```
#way2
```

```
binary = {'rbc':{'normal':1,'abnormal':0},
          'pc':{'normal':1,'abnormal':0},
          'pcc':{'present':1,'notpresent':0},
          'ba':{'present':1,'notpresent':0},
          'htn':{'yes':1,'no':0},
          'dm':{'yes':1,'no':0},
          'cad':{'yes':1,'no':0},
          'appet':{'good':1,'poor':0},
          'pe':{'yes':1,'no':0},
          'ane':{'yes':1,'no':0}}
```

```

    }

for i,j in binary.items():
    X1[i] = X1[i].replace(j)

X1

```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	hemo	pcv	wbcc	rbcc	htn	dm	cad	appet	p
0	48.0	80.0	1.020	1.0	0.0	-1	1	0	0	121.0	...	15.4	44.0	7800.0	5.2	1	1	0	0	0
1	7.0	50.0	1.020	4.0	0.0	-1	1	0	0	NaN	...	11.3	38.0	6000.0	NaN	0	0	0	0	0
2	62.0	80.0	1.010	2.0	3.0	1	1	0	0	423.0	...	9.6	31.0	7500.0	NaN	0	1	0	0	0
3	48.0	70.0	1.005	4.0	0.0	1	0	1	0	117.0	...	11.2	32.0	6700.0	3.9	1	0	0	0	0
4	51.0	80.0	1.010	2.0	0.0	1	1	0	0	106.0	...	11.6	35.0	7300.0	4.6	0	0	0	0	0
...
395	55.0	80.0	1.020	0.0	0.0	1	1	0	0	140.0	...	15.7	47.0	6700.0	4.9	0	0	0	0	0
396	42.0	70.0	1.025	0.0	0.0	1	1	0	0	75.0	...	16.5	54.0	7800.0	6.2	0	0	0	0	0
397	12.0	80.0	1.020	0.0	0.0	1	1	0	0	100.0	...	15.8	49.0	6600.0	5.4	0	0	0	0	0
398	17.0	60.0	1.025	0.0	0.0	1	1	0	0	114.0	...	14.2	51.0	7200.0	5.9	0	0	0	0	0
399	58.0	80.0	1.025	0.0	0.0	1	1	0	0	131.0	...	15.8	53.0	6800.0	6.1	0	0	0	0	0

```

scale = StandardScaler()

X1[float_col] = scale.fit_transform(X1[float_col])

X1

```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	hemo	pcv	wbcc	rbcc	htn	dm	cad	appet	p
0	-0.203139		0.258373	1.020	1.0	0.0	-1	1	0	0		-0.341498	...	0.988022	0.569881	-0.2				
1	-2.594124		-1.936857	1.020	4.0	0.0	-1	1	0	0		NaN	...	-0.421688	-0.098536	-0.8				
2	0.613295		0.258373	1.010	2.0	3.0	1	1	0	0		3.473064	...	-1.006202	-0.878356	-0.3				
3	-0.203139		-0.473370	1.005	4.0	0.0	1	0	1	0		-0.392022	...	-0.456071	-0.766953	-0.5				
4	-0.028189		0.258373	1.010	2.0	0.0	1	1	0	0		-0.530963	...	-0.318538	-0.432744	-0.3				
...				

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	hemo	pcv	wbcc	rbcc	htn	dm	cad	appet	p
395	0.205078		0.258373			1.020	0.0	0.0	1	1	0	0	-0.101509	...	1.091172		0.904090		-0.5	
396	-0.553039		-0.473370			1.025	0.0	0.0	1	1	0	0	-0.922524	...	1.366237		1.683910		-0.2	
397	-2.302541		0.258373			1.020	0.0	0.0	1	1	0	0	-0.606749	...	1.125555		1.126896		-0.6	
398	-2.010957		-1.205114			1.025	0.0	0.0	1	1	0	0	-0.429915	...	0.575424		1.349701		-0.4	
399	0.380028		0.258373			1.025	0.0	0.0	1	1	0	0	-0.215188	...	1.125555		1.572507		-0.5	

3. **Dataset Overview:** Provide a detailed description of the dataset, covering variables, summaries, observation counts, data types, and distributions (at least three statements).

```
X1.describe()
```

	age	bp	rbc	pc	pcc	ba	bgr	bu	sc	sod	...	hemo	pcv	wbcc	rbcc	htn	dm	cad	appet	p
count	3.910000e+02			3.880000e+02		400.00000	400.000000	400.000000				400.000000	400.000000	400.000000	400.000000	3.560000e+02	3.			
mean	9.994847e-17			-2.380684e-16		0.12250	0.485000	0.095000				0.045000	0.045000	0.045000	0.045000	-1.796316e-16	-3.			
std	1.001281e+00			1.001291e+00		0.93256	0.759089	0.325946				0.251262	0.251262	0.251262	0.251262	1.001407e+00	1.			
min	-2.885708e+00			-1.936857e+00		-1.00000	-1.000000	-1.000000				-1.000000	-1.000000	-1.000000	-1.000000	-1.591967e+00	-1.			
25%	-5.530393e-01			-4.733701e-01		-1.00000	0.000000	0.000000				0.000000	0.000000	0.000000	0.000000	-6.193803e-01	-6.			
50%	2.050779e-01			2.583733e-01		1.00000	1.000000	1.000000				0.000000	0.000000	0.000000	0.000000	-3.414983e-01	-3.			
75%	7.590867e-01			2.583733e-01		1.00000	1.000000	1.000000				0.000000	0.000000	0.000000	0.000000	1.890038e-01	1.			
max	2.246163e+00			7.575807e+00		1.00000	1.000000	1.000000				1.000000	1.000000	1.000000	1.000000	4.319341e+00	6.			

```
X1.dtypes
```

```
age      float64
bp      float64
sg      category
al      category
su      category
rbc      int8
pc      int8
```

```

pcc          int8
ba           int8
bgr         float64
bu          float64
sc          float64
sod         float64
pot         float64
hemo        float64
pcv         float64
wbcc        float64
rbcc        float64
htn         int8
dm          int8
cad         int8
appet       int8
pe          int8
ane         int8
dtype: object

```

Since there are 25 variables and 400 observations in Chronic Kidney Disease dataset.

1). There are 14 float64 type variables, and they are “age”, “bp”, “sg”, “al”, “su”, “bgr”, “bu”, “sc”, “sod”, “pot”, “hemo”, “pcv”, “wbcc”, “rbcc”. There are 11 object type variables, and they are “rbc”, “pc”, “pcc”, “ba”, “htn”, “dm”, “cad”, “appet”, “pe”, “ane”, “calss”. In these 25 variables, all variables in X are covariates (consists 14 floats and 10 object) and the variable in y is the response (the variable “class” which is an object variable). Moreover, the response is “class” contains only 2 category: “ckd” or “not ckd”, so the response is binary.

2). The number of features is 24, and they are: “age”, “bp”, “sg”, “al”, “su”, “bgr”, “bu”, “sc”, “sod”, “pot”, “hemo”, “pcv”, “wbcc”, “rbcc”, “rbc”, “pc”, “pcc”, “ba”, “htn”, “dm”, “cad”, “appet”, “pe”, “ane”, “calss”.

3). The number of observations is 400. But we can see that the count of some variables is less than 400, for example: the count of age is 391 which is less than 400. This means there are missing

values in some variables in this dataset.

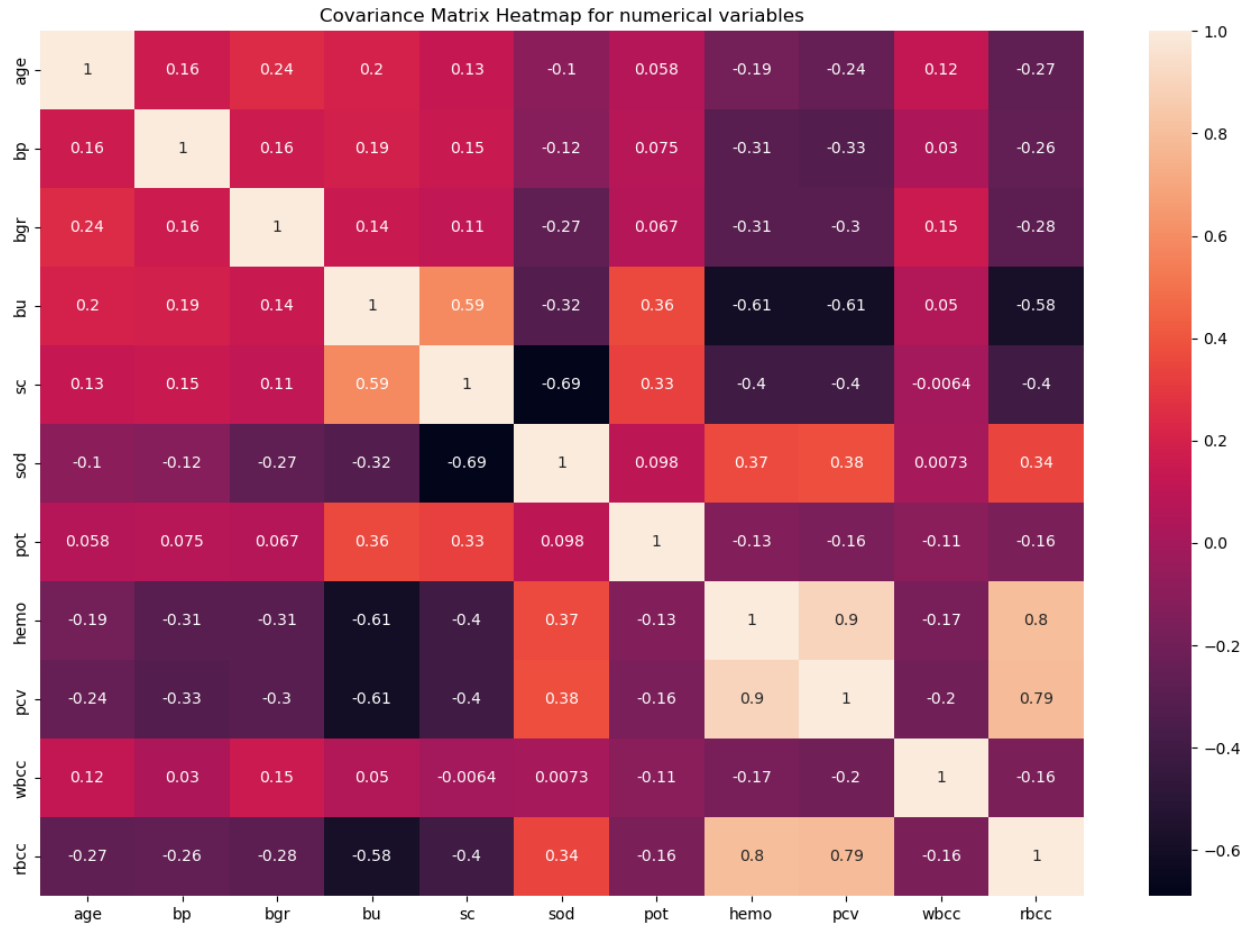
4). The type of variables: There are 14 float64 type variables, and they are “age”, “bp”, “sg”, “al”, “su”, “bgr”, “bu”, “sc”, “sod”, “pot”, “hemo”, “pcv”, “wbcc”, “rbcc”. There are 11 object type variables, and they are “rbc”, “pc”, “pcc”, “ba”, “htn”, “dm”, “cad”, “appet”, “pe”, “ane”, “calss”. In these 25 variables, all variables in X are covariates (consists 14 floats and 10 object) and the variable in y is the response (the variable “class” which is an object variable). Moreover, the response is “class” contains only 2 category: “ckd” or “not ckd”, so the response is binary.

5). Missing values: There are 9 missing values in “age”, 12 missing values in “bp”, 47 missing values in “sg”, 46 missing values in “al”, 49 missing values in “su”, 152 missing values in “rbc”, 65 missing values in “pc”, 4 missing values in “pcc”, 4 missing values in “ba”, 44 missing values in “bgr”, 19 missing values in “bu”, 17 missing values in “sc”, 87 missing values in “sod”, 88 missing values in “pot”, 52 missing values in “hemo”, 71 missing values in “pcv”, 106 missing values in “wbcc”, 131 missing values in “rbcc”, 2 missing values in “htn”, 2 missing values in “dm”, 2 missing values in “cad”, 1 missing value in “appet”, 1 missing value in “pe” and 1 missing value in “ane”.

0 missing value in “class”.

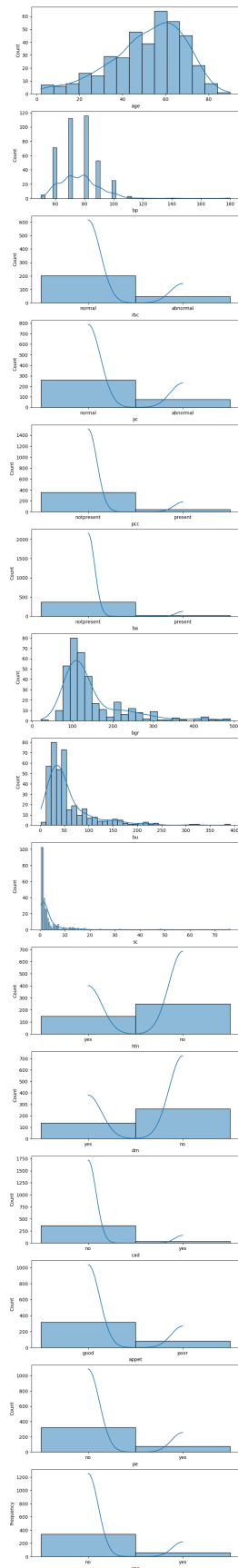
4. **Association Between Variables:** Analyze variable relationships and their implications for feature selection or extraction (at least three statements).

```
# prof says numerical variables should use heatmap, categorical variables should use barplot
#
#
#
#
#numerical variables:
float64=X1.select_dtypes(include=['float64'])
plt.figure(figsize=(15, 10))
sns.heatmap(float64.corr(),
            annot=True)
plt.title('Covariance Matrix Heatmap for numerical variables')
plt.show()
```



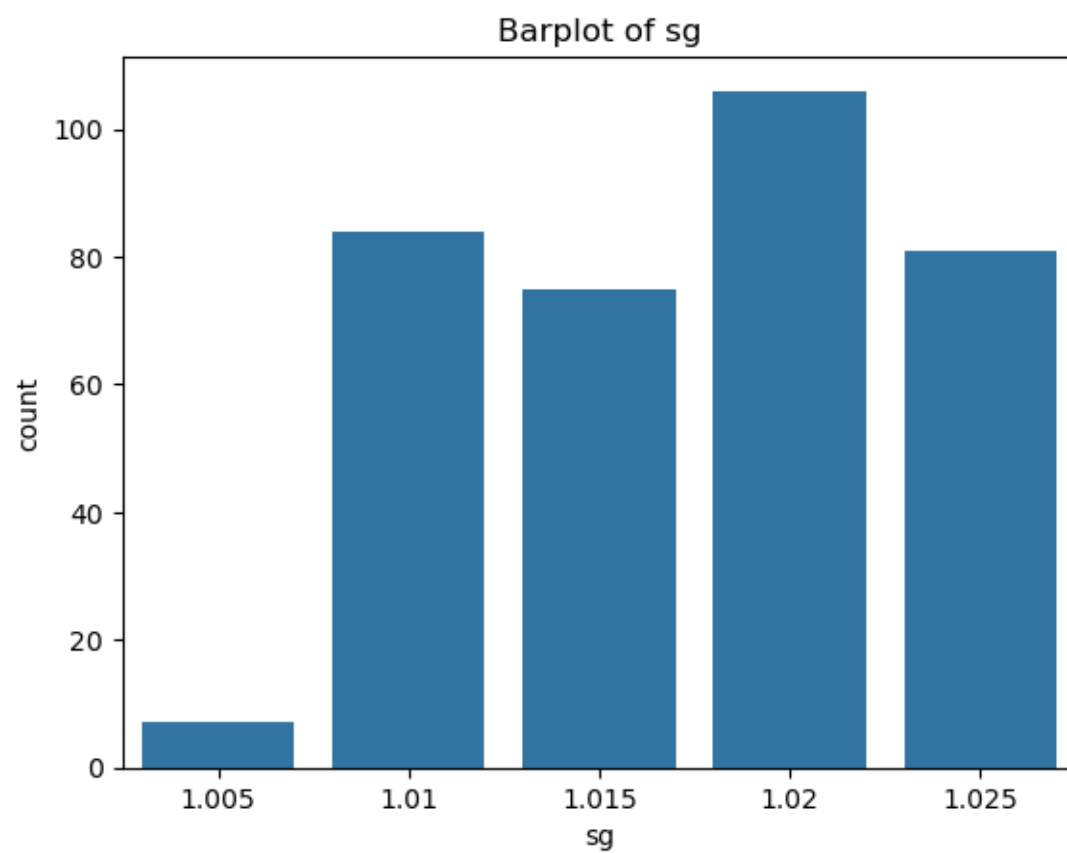
```
def plot_dist(data, features):
    fig, axes = plt.subplots(nrows=len(features), ncols=1, figsize=(8, 4*len(features)))
    if len(features) == 1:
        axes = [axes]
    for ax, feature in zip(axes, features):
        sns.histplot(data[feature], kde=True, ax=ax)
        ax.set_xlabel(feature)
        ax.set_ylabel('Frequency')

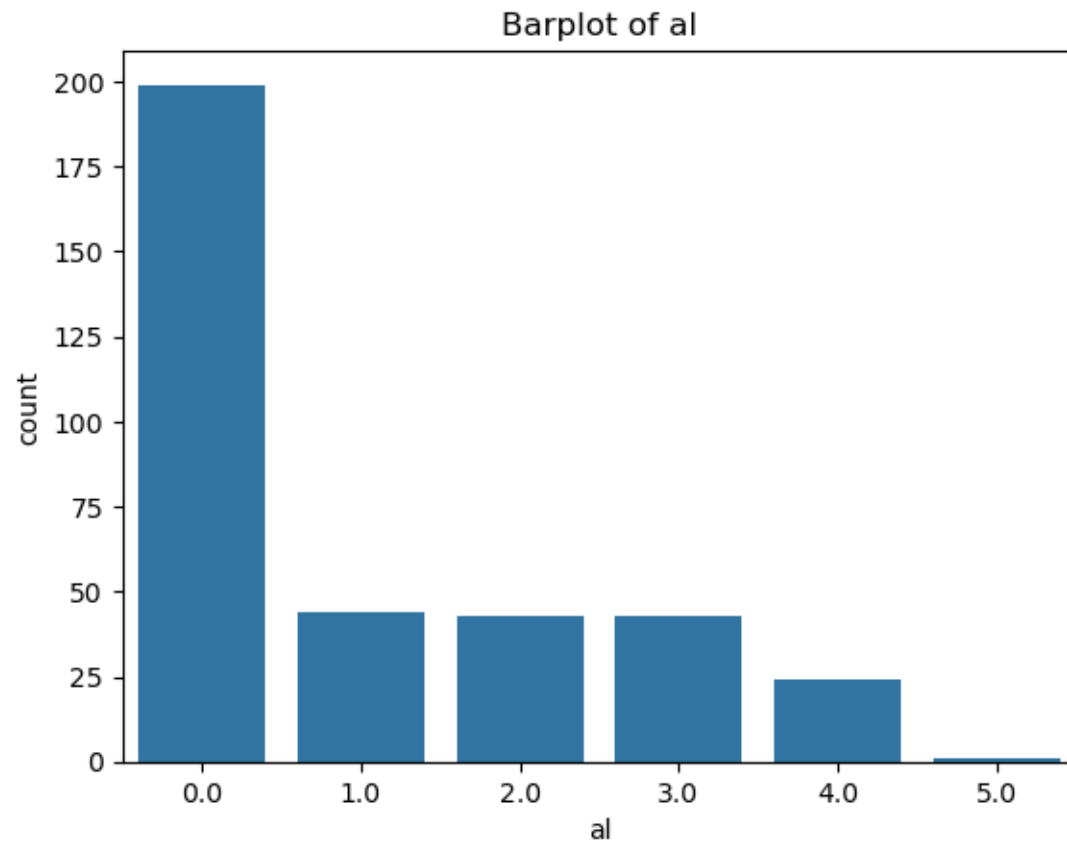
plt.show()
plot_dist(X, numerical_cols)
```

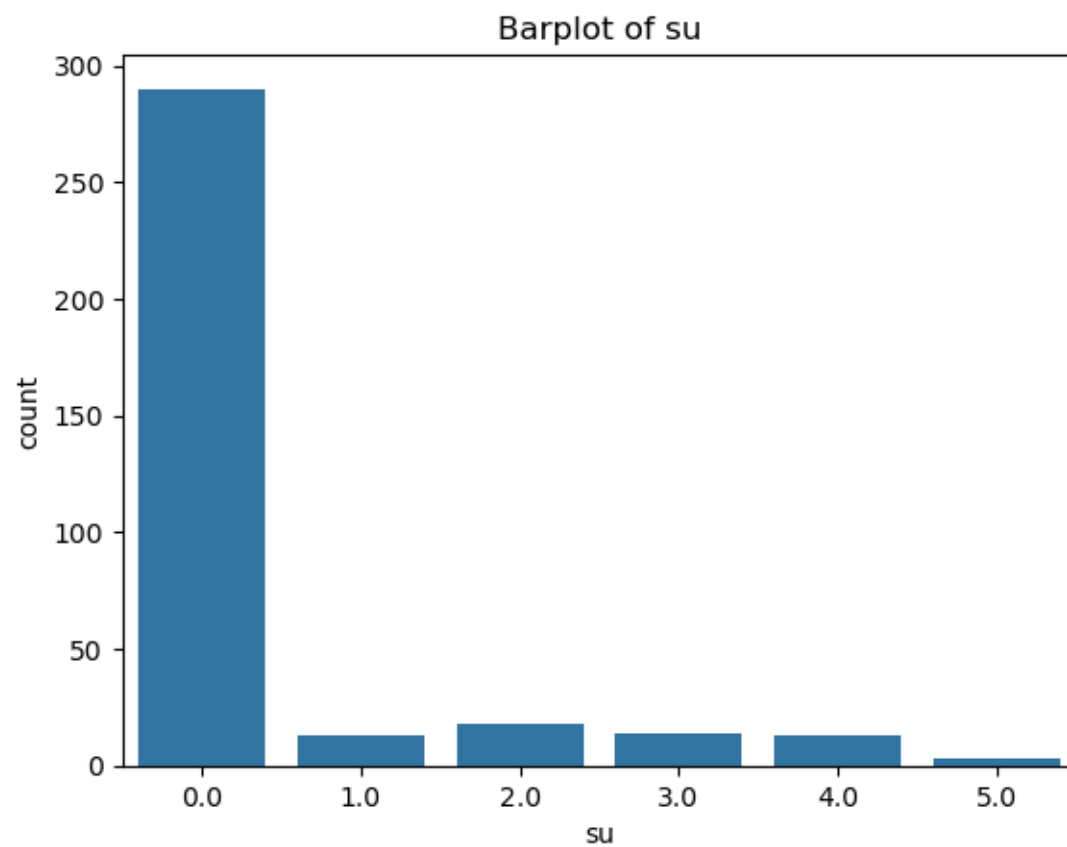


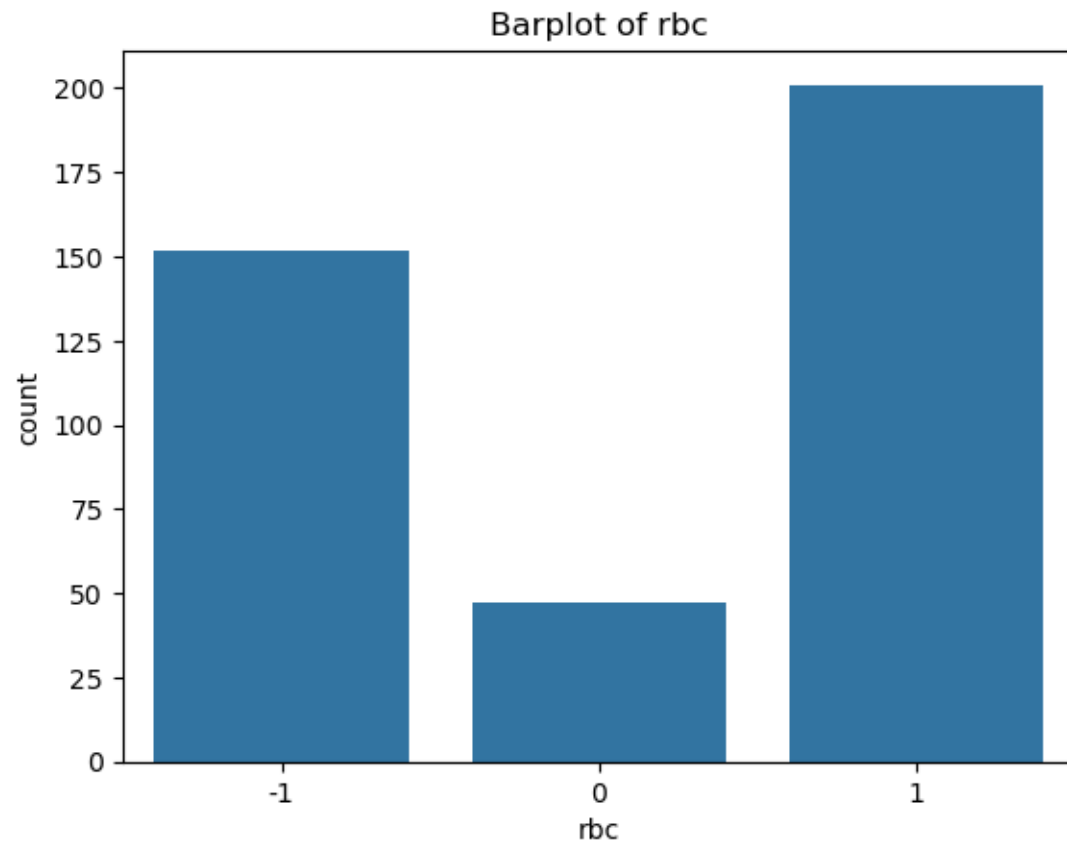
```
#categorical=X1.select_dtypes(include=['category','int8'])
#sns.histplot(
#    data=categorical,
#    color='random',
#    edgecolor='black',
#    binwidth=0.25
#    )
#plt.xlabel('barplot')
#plt.ylabel('Frequency')
#plt.show()
```

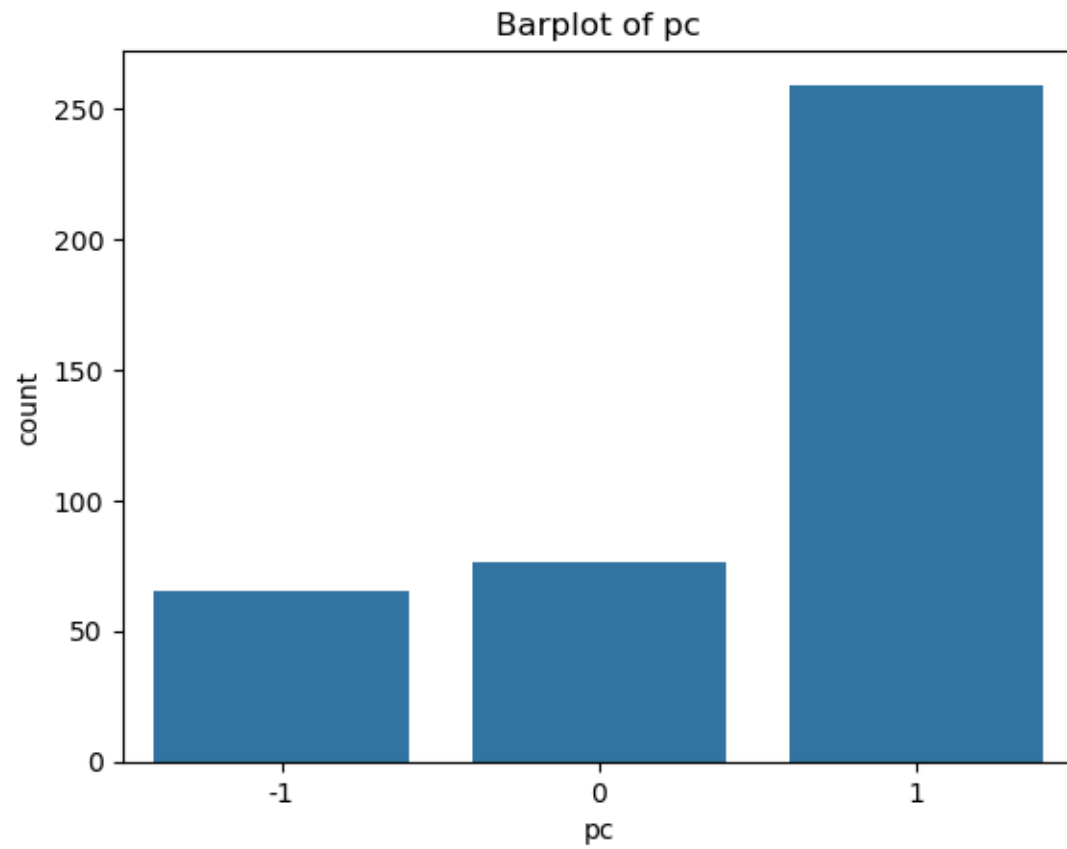
```
#categorical variables
categorical=X1.select_dtypes(include=['category','int8'])
for column in categorical.columns:
    plt.figure()
    sns.countplot(x=column, data=categorical)
    plt.title(f'Barplot of {column}')
    plt.show()
```

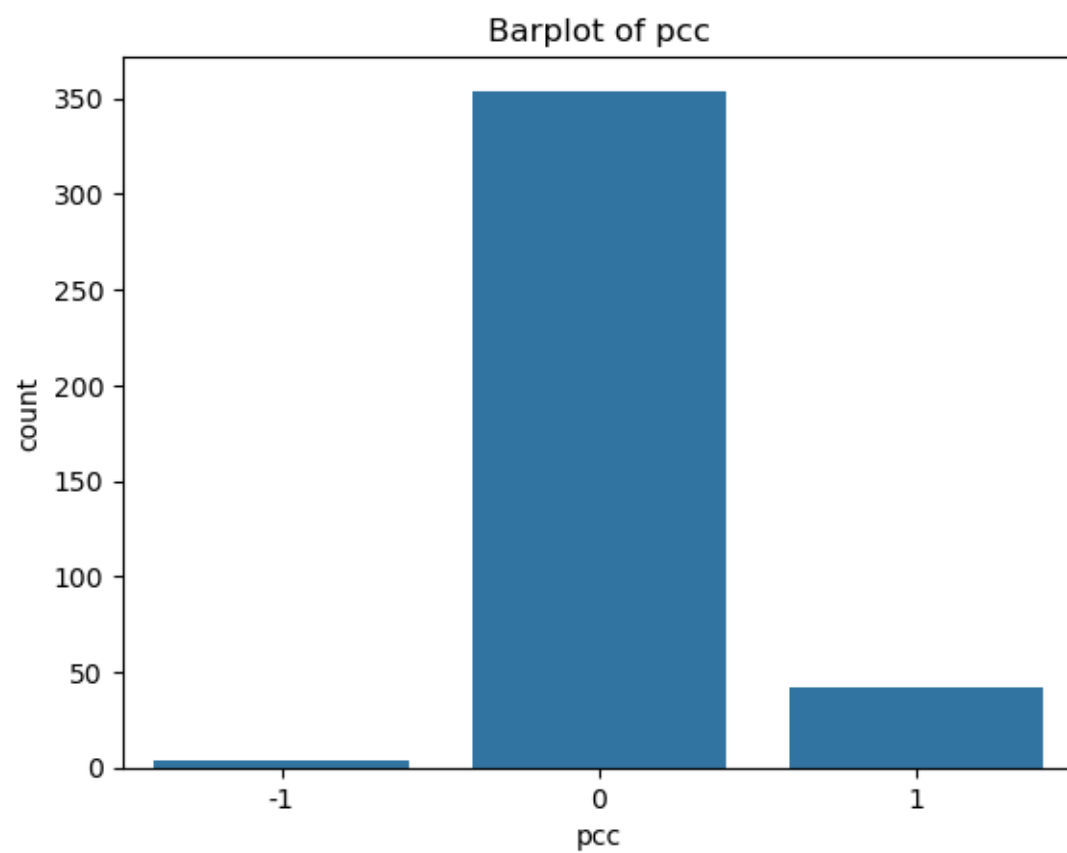


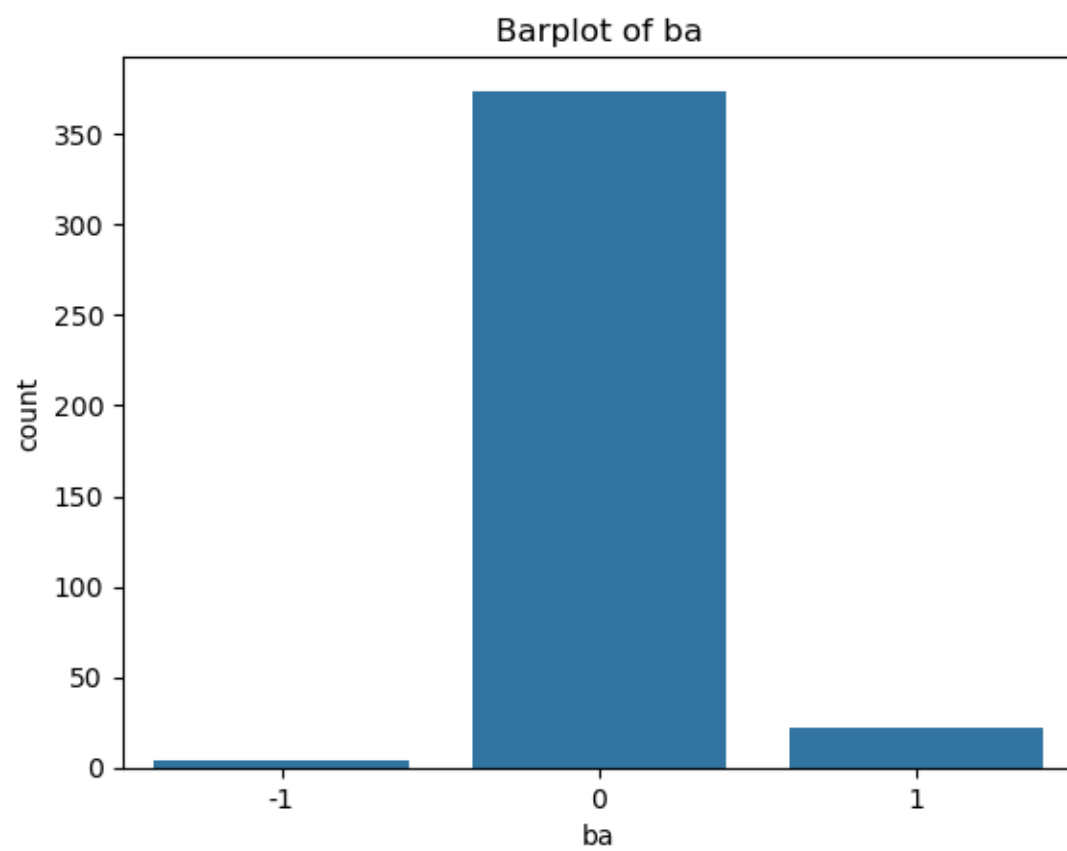


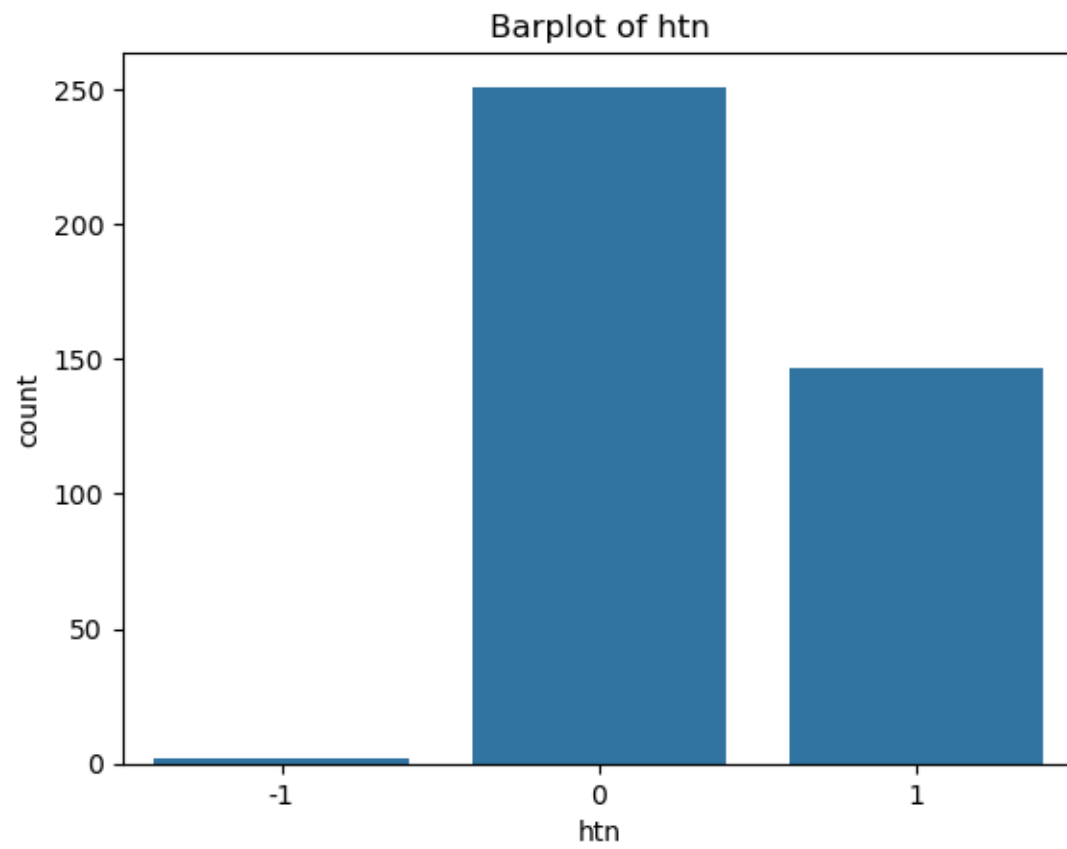


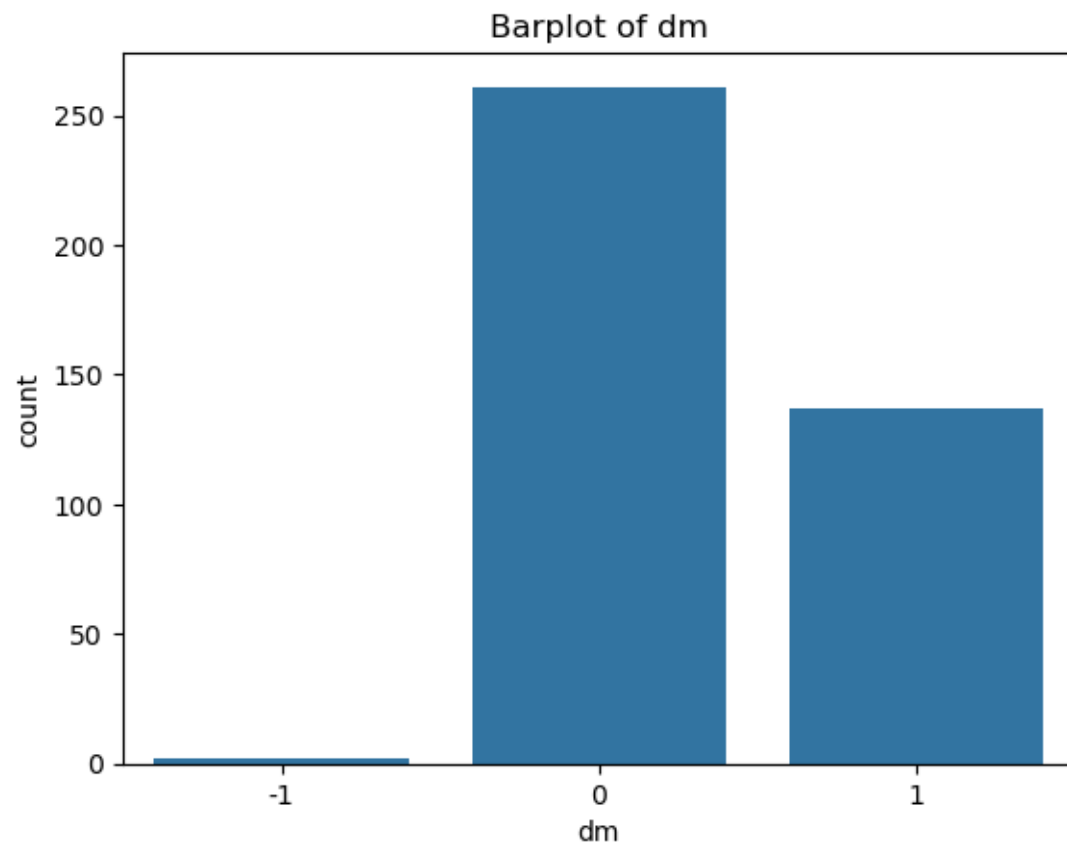


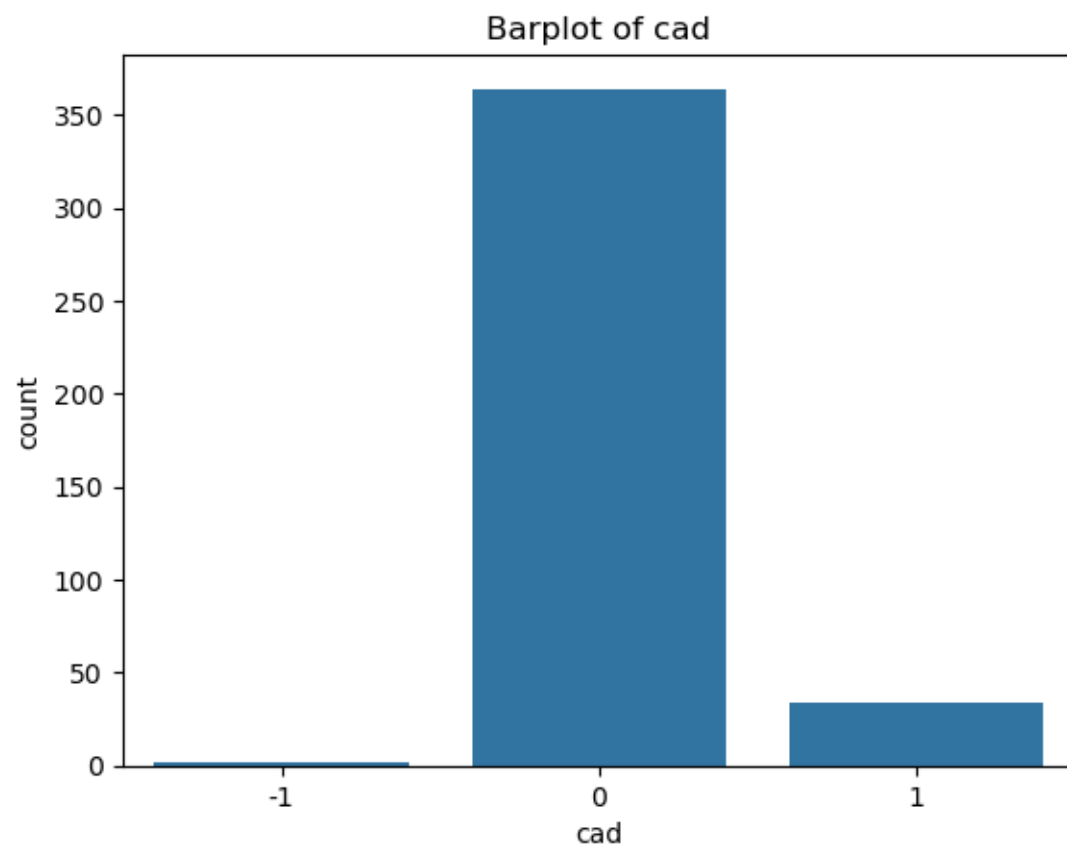


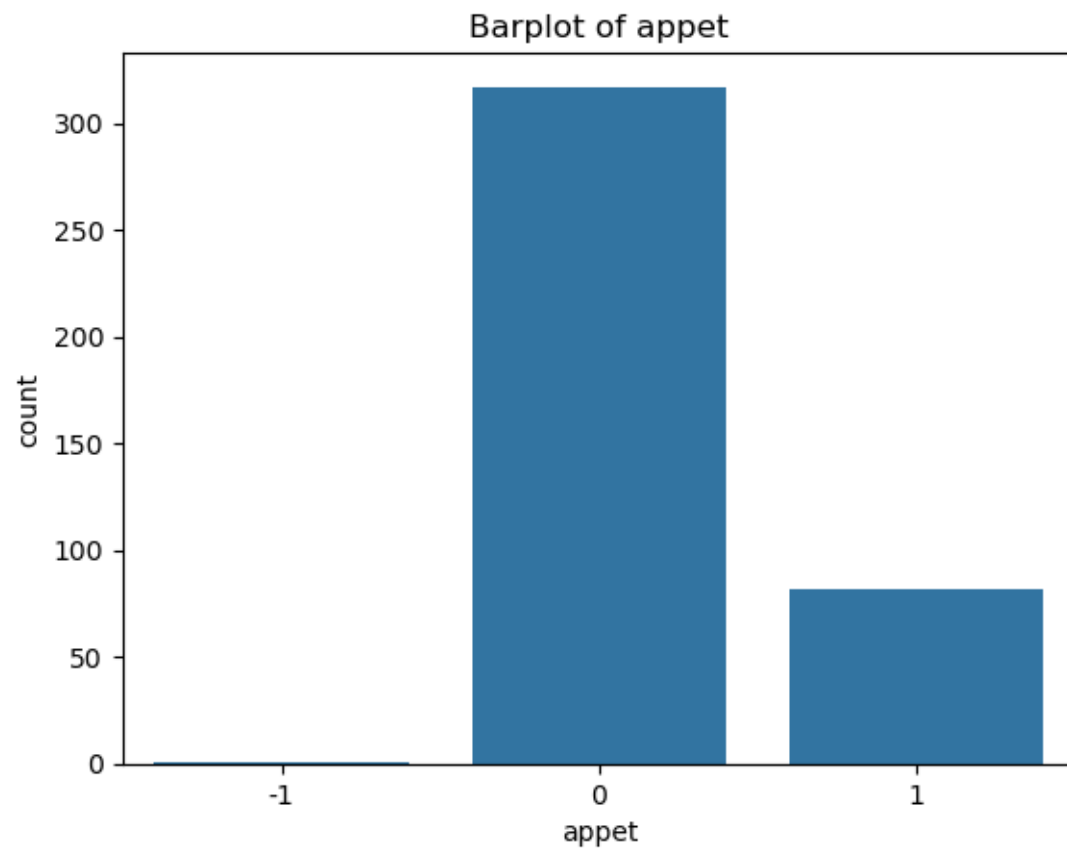


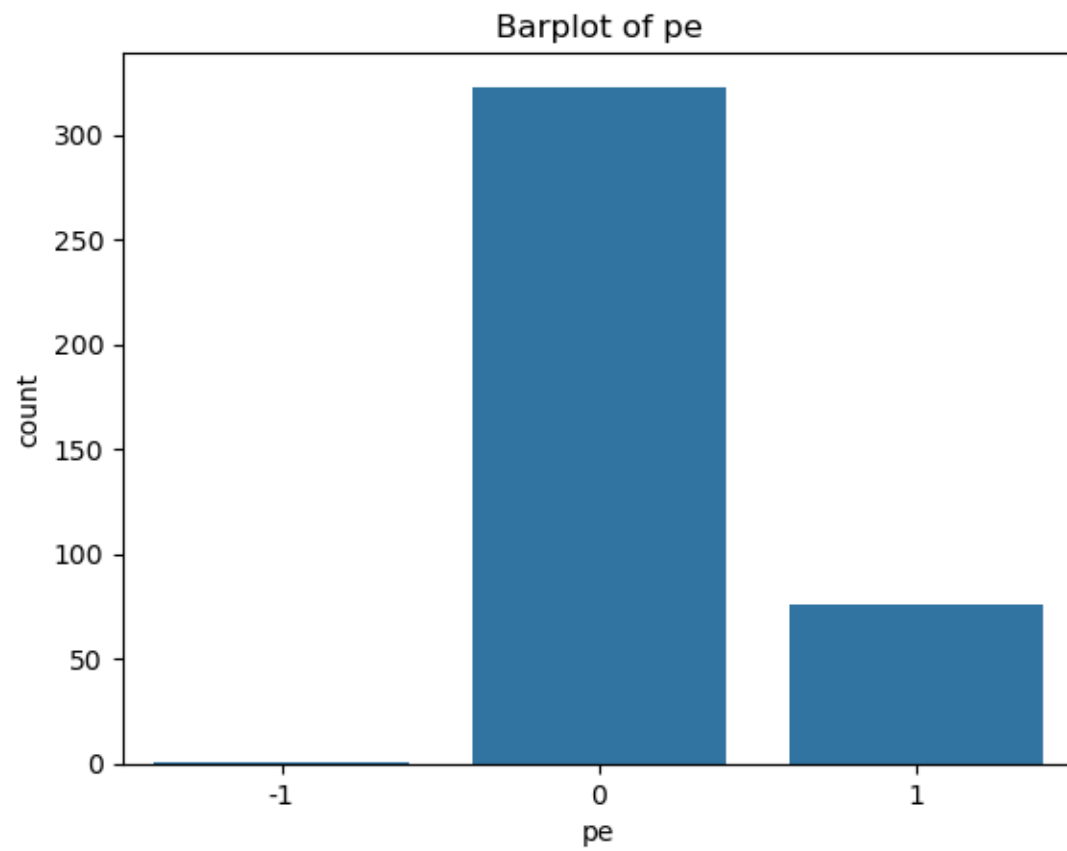


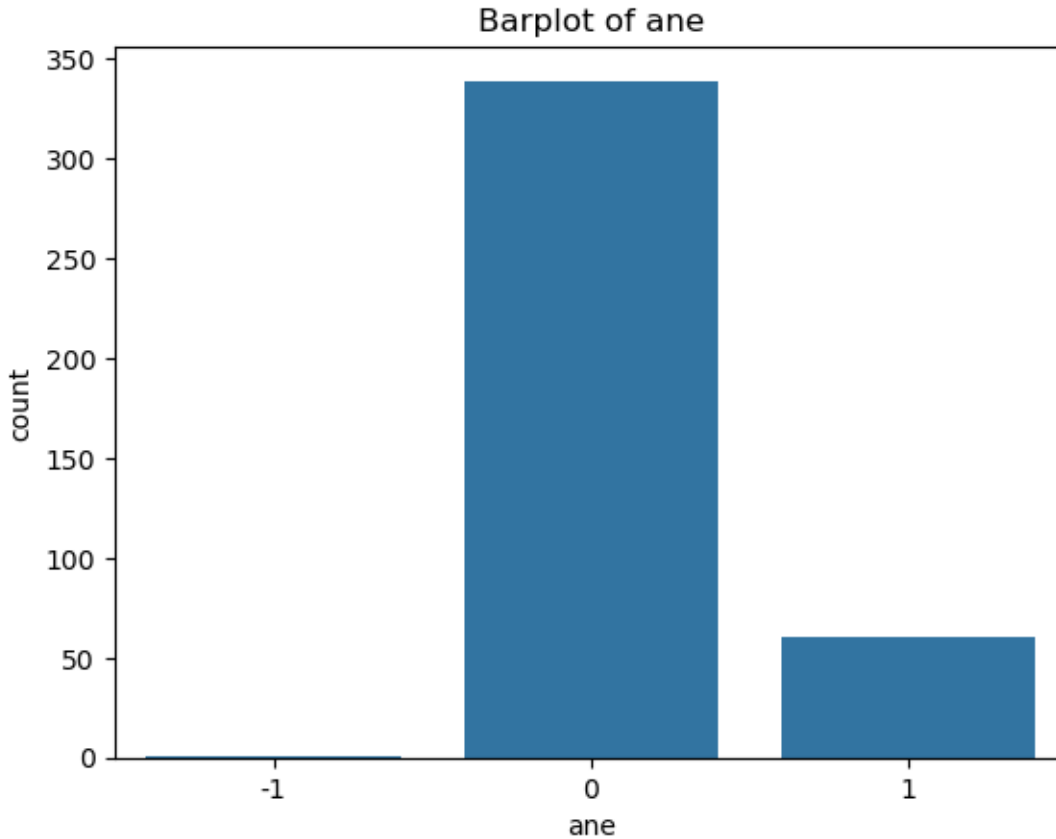












We use heatmap for numerical variables and barplot for categorical variables.

For heatmap:

- 1). This is a symmetric matrix and the elements on the diagonal are the variance. The diagonal of the correlation matrix is 1.
- 2). The elements not on the diagonal are the covariance.
- 3). The smallest number in the matrix is -0.69 which means "sod" and "sc" are negatively correlated.
- 4). The largest number in the matrix is 1 (the diagonal).
- 5). The largest number other than 1 is 0.9 which means "hemo" and "pcv" are positively correlated.

For barplots:

- 1). In those binary variables, the category "-1" could suggest missing values.

2). For example, in “pe” (pedal edema), the most common category is “0”, which could imply the absence of pedal edema, and the second most common category is “1”, which could imply the presence of pedal edema, and the least category “-1” are missing values.

3). “htn” (hypertension) and “dm” (diabetes mellitus) are not good since the diistribution is not extreme.

5. Missing Value Analysis and Handling: Implement your strategy for identifying and addressing missing values in the dataset, or provide reasons for not addressing them.

```
# prof says cannot drop all the missing values ,44 44 drop variables
#         dropped numerical variable can replaced by mean
#         dropped variables missing values replaced by mean.
# When missing vlues is too large, lets say 90 missing vlues out of 100, then even though we us
X.isnull().sum()
```

age	9
bp	12
sg	47
al	46
su	49
rbc	152
pc	65
pcc	4
ba	4
bgr	44
bu	19
sc	17
sod	87
pot	88
hemo	52
pcv	71
wbcc	106
rbcc	131

```
htn      2
dm       2
cad      2
appet    1
pe       1
ane      1
dtype: int64
```

```
y.isnull().sum()
```

```
class    0
dtype: int64
```

```
#way2
float_na=X1[float_col].isna()
float_nasum=float_na.sum()
print(float_nasum)

cat_na=X1[columns_to_convert].isna()
cat_nasum=cat_na.sum()
print(cat_nasum)

binary_na = (X1[object_convert] == -1)
binary_nasum=binary_na.sum()
print(binary_nasum)
```

```
age      9
bp       12
bgr      44
bu       19
sc       17
sod      87
```

```

pot      88
hemo     52
pcv      71
wbcc     106
rbcc     131
dtype: int64
sg       47
al       46
su       49
dtype: int64
rbc      152
pc       65
pcc      4
ba       4
htn      2
dm       2
cad      2
appet    1
pe       1
ane      1
dtype: int64

```

Missing values: There are 9 missing values in “age”, 12 missing values in “bp”, 47 missing values in “sg”, 46 missing values in “al”, 49 missing values in “su”, 152 missing values in “rbc”, 65 missing values in “pc”, 4 missing values in “pcc”, 4 missing values in “ba”, 44 missing values in “bgr”, 19 missing values in “bu”, 17 missing values in “sc”, 87 missing values in “sod”, 88 missing values in “pot”, 52 missing values in “hemo”, 71 missing values in “pcv”, 106 missing values in “wbcc”, 131 missing values in “rbcc”, 2 missing values in “htn”, 2 missing values in “dm”, 2 missing values in “cad”, 1 missing value in “appet”, 1 missing value in “pe” and 1 missing value in “ane”.

0 missing value in “class”.

```
#the stragegy for addressing missing values
```

```
X1_drop= X1.dropna()
```

```
X1_drop
```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	hemo	pcv	wbcc	rbcc	htn	dm	cad	appet	p
3	-0.203139		-0.473370			1.005	4.0	0.0	1	0	1	0	-0.392022	...	-0.456071	-0.766953	-0.5			
5	0.496661		0.990117			1.015	3.0	0.0	-1	-1	0	0	-0.935155	...	-0.112239	0.012867	-0.2			
9	0.088445		0.990117			1.020	2.0	0.0	0	0	1	0	-0.985679	...	-1.040585	-1.101161	1.2			
11	0.671612		-0.473370			1.010	3.0	0.0	0	0	1	0	2.929931	...	-0.593604	-0.766953	-1.3			
12	0.963195		-0.473370			1.015	3.0	1.0	-1	1	1	0	0.757399	...	-0.971819	-1.212564	1.2			
...		
395	0.205078		0.258373			1.020	0.0	0.0	1	1	0	0	-0.101509	...	1.091172	0.904090	-0.5			
396	-0.553039		-0.473370			1.025	0.0	0.0	1	1	0	0	-0.922524	...	1.366237	1.683910	-0.2			
397	-2.302541		0.258373			1.020	0.0	0.0	1	1	0	0	-0.606749	...	1.125555	1.126896	-0.6			
398	-2.010957		-1.205114			1.025	0.0	0.0	1	1	0	0	-0.429915	...	0.575424	1.349701	-0.4			
399	0.380028		0.258373			1.025	0.0	0.0	1	1	0	0	-0.215188	...	1.125555	1.572507	-0.5			

dropped the columns if the missing value of this variable is greater than 44.

```
missing_values_count = X1.isna().sum()
```

```
columns_to_drop = missing_values_count[missing_values_count > 50].index
```

```
columns_to_drop
```

```
Index(['sod', 'pot', 'hemo', 'pcv', 'wbcc', 'rbcc'], dtype='object')
```

```
X1_dropped = X1.drop(columns=columns_to_drop)
```

```
X1_dropped
```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	sc	htn	dm	cad	appet	pe	ane	
0	-0.203139		0.258373			1.020	1.0	0.0	-1	1	0	0	-0.341498	-0.424804	-0.326574	1		1	
1	-2.594124		-1.936857			1.020	4.0	0.0	-1	1	0	0	NaN		-0.781687	-0.396338	0		0

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	sc	htn	dm	cad	appet	pe	ane
2		0.613295		0.258373		1.010	2.0	3.0	1	1	0	0		3.473064	-0.087748	-0.221928	0	1
3		-0.203139		-0.473370		1.005	4.0	0.0	1	0	1	0		-0.392022	-0.028268	0.126891	1	0
4		-0.028189		0.258373		1.010	2.0	0.0	1	1	0	0		-0.530963	-0.623073	-0.291692	0	0
...
395		0.205078		0.258373		1.020	0.0	0.0	1	1	0	0		-0.101509	-0.167055	-0.448661	0	0
396		-0.553039		-0.473370		1.025	0.0	0.0	1	1	0	0		-0.922524	-0.523939	-0.326574	0	0
397		-2.302541		0.258373		1.020	0.0	0.0	1	1	0	0		-0.606749	-0.623073	-0.431220	0	0
398		-2.010957		-1.205114		1.025	0.0	0.0	1	1	0	0		-0.429915	-0.147229	-0.361456	0	0
399		0.380028		0.258373		1.025	0.0	0.0	1	1	0	0		-0.215188	-0.781687	-0.344015	0	0

```
numerical_cols = X1_dropped.select_dtypes(include=['number']).columns
column_means = X1_dropped[numerical_cols].mean()

X1_filled = X1_dropped.copy()
X1_filled[numerical_cols] = X1_filled[numerical_cols].fillna(column_means)
```

Replacing the missing value by the mean of the variables for numerical variables.

```
from sklearn.impute import SimpleImputer
```

```
obj_col = X1_filled.select_dtypes(include=['category']).columns.tolist()
cat_imputer = SimpleImputer(strategy='most_frequent',missing_values=np.nan)
X1_filled.loc[:, obj_col] = cat_imputer.fit_transform(X1_filled.loc[:, obj_col])
```

Replacing the missing value by the most frequency input of the variables for categorical variables.

```
X1_filled.isna().sum()
```

```
age      0
bp       0
sg       0
```



```

al      0
su      0
rbc     0
pc      0
pcc     0
ba      0
bgr     0
bu      0
sc      0
htn     0
dm      0
cad     0
appet   0
pe      0
ane     0
dtype: int64

```

The reason for not strategy for addressing them is : ???

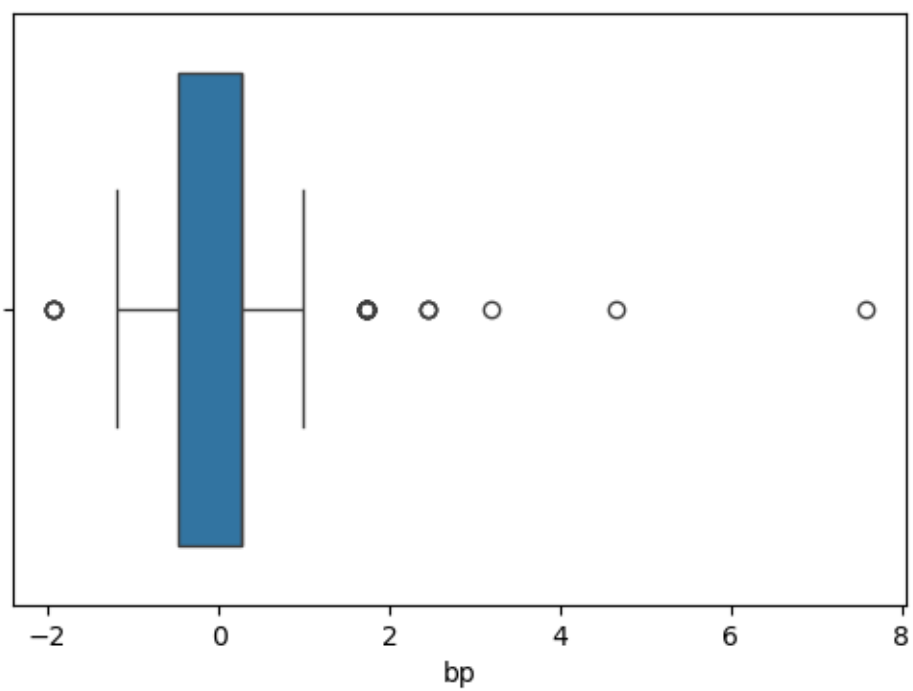
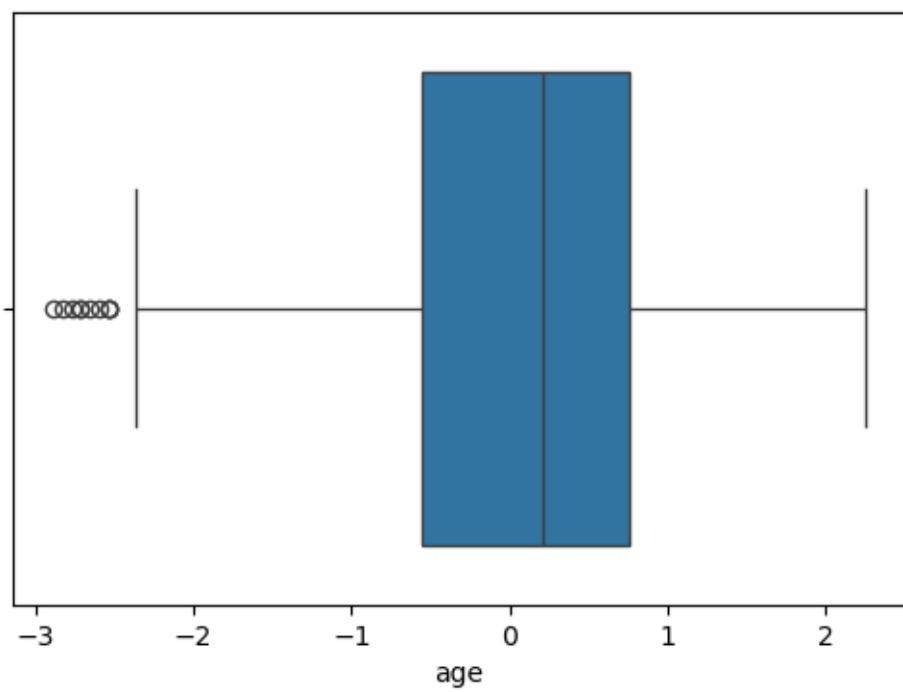
In some algorithms, the missing values can be seen as a seperate category. For instance, in decision tree, KNN, etc.

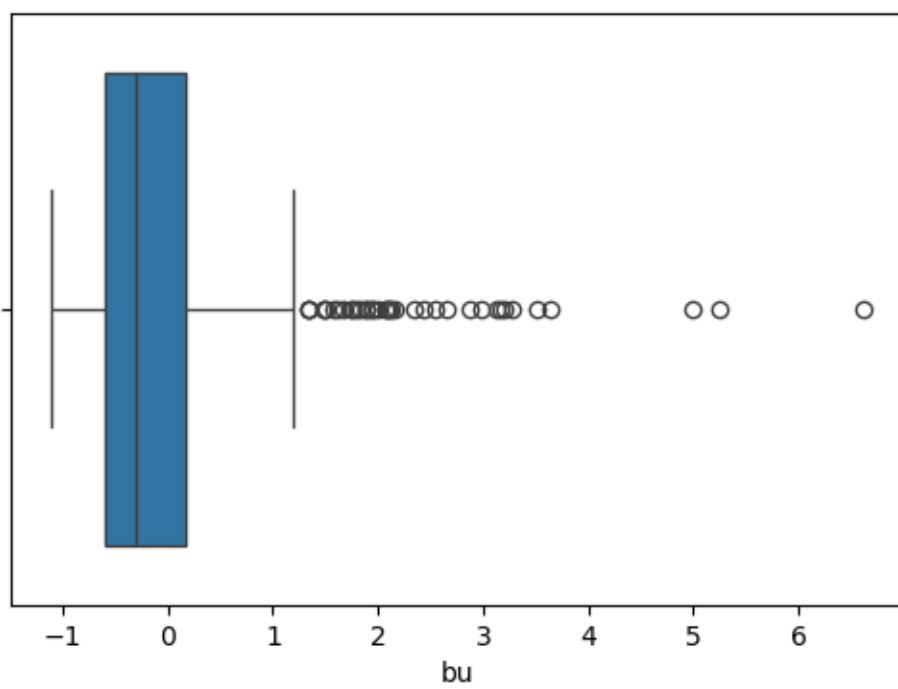
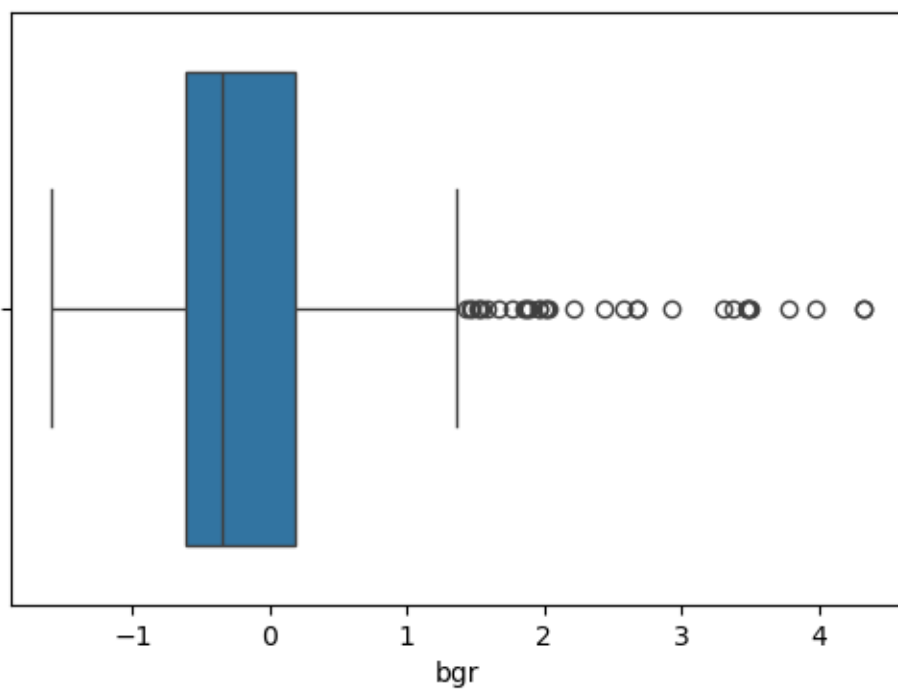
6. **Outlier Analysis:** Implement your approach for identifying and managing outliers, or provide reasons for not addressing them.

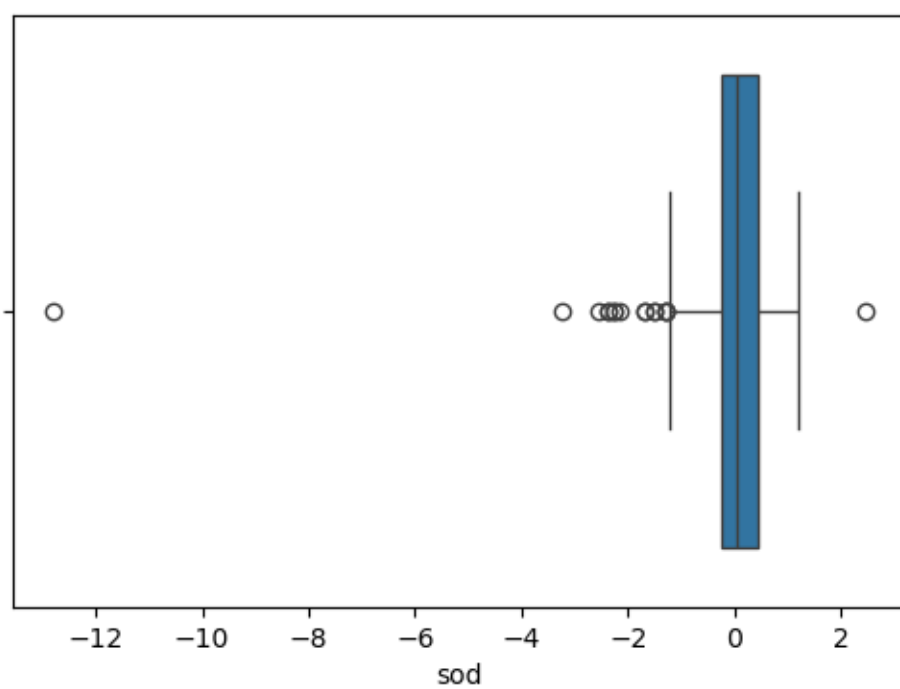
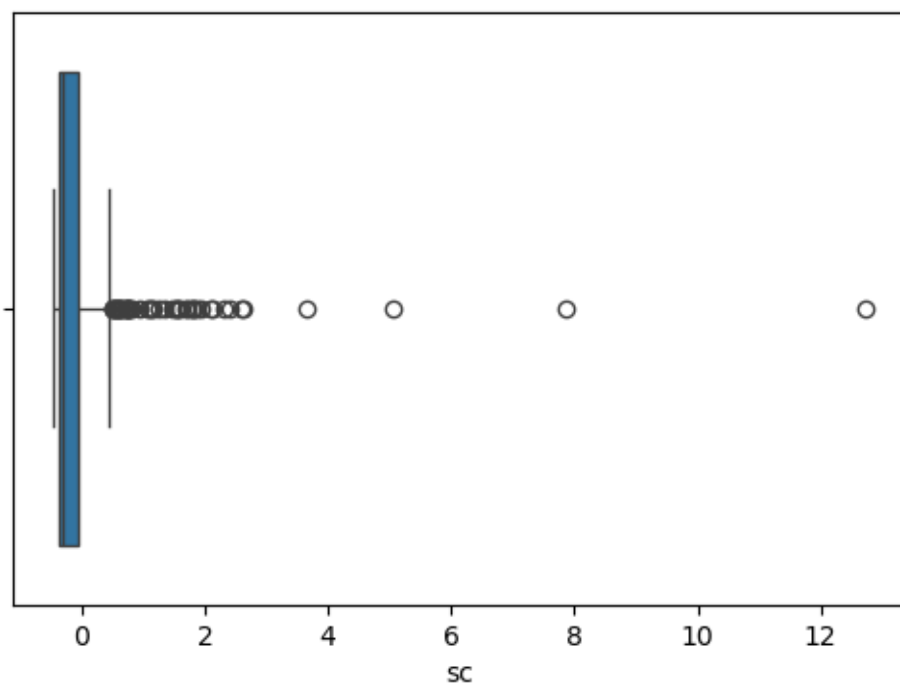
```

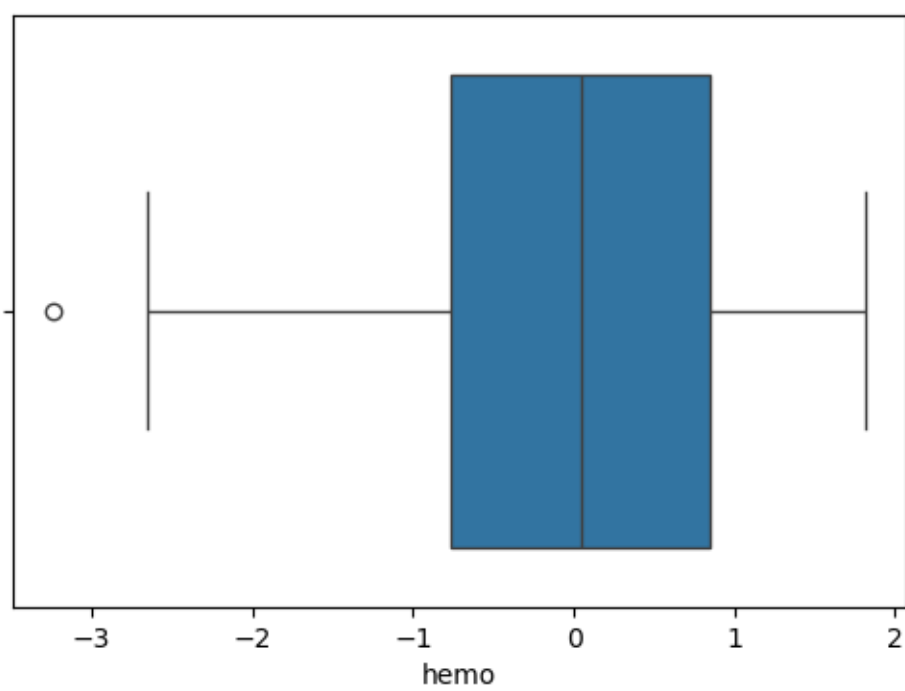
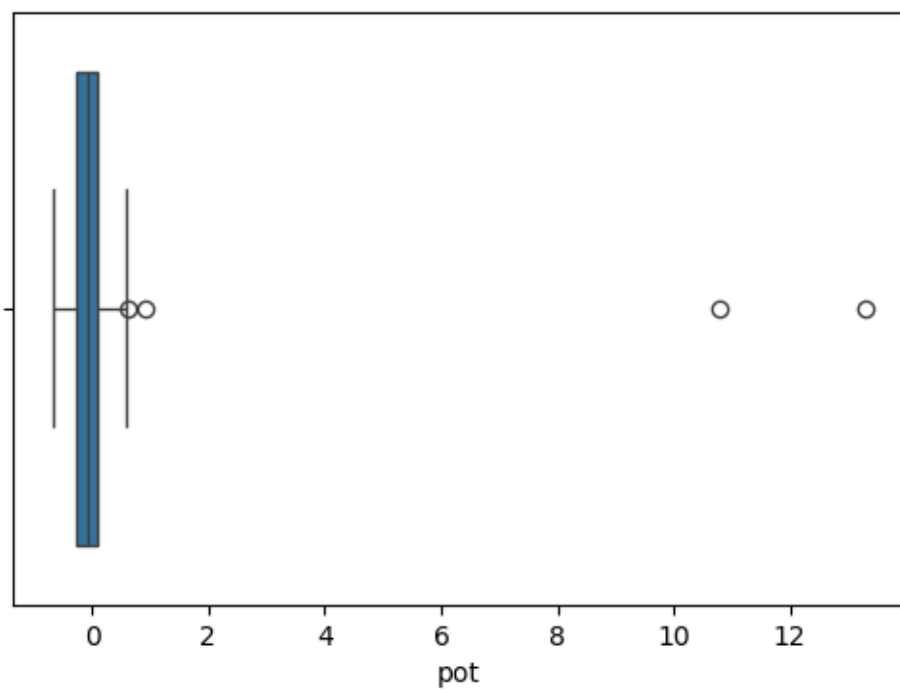
for col in float_col:
    plt.figure(figsize=(6, 4))
    sns.boxplot(x=X1.select_dtypes(include=['float64'])[col])
    plt.xlabel(col)
    plt.show()

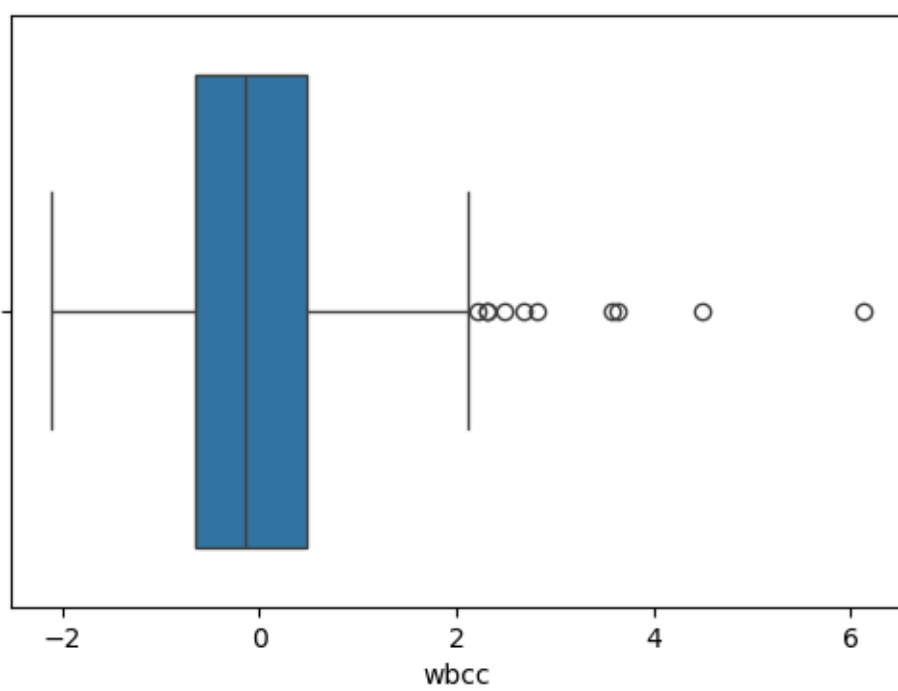
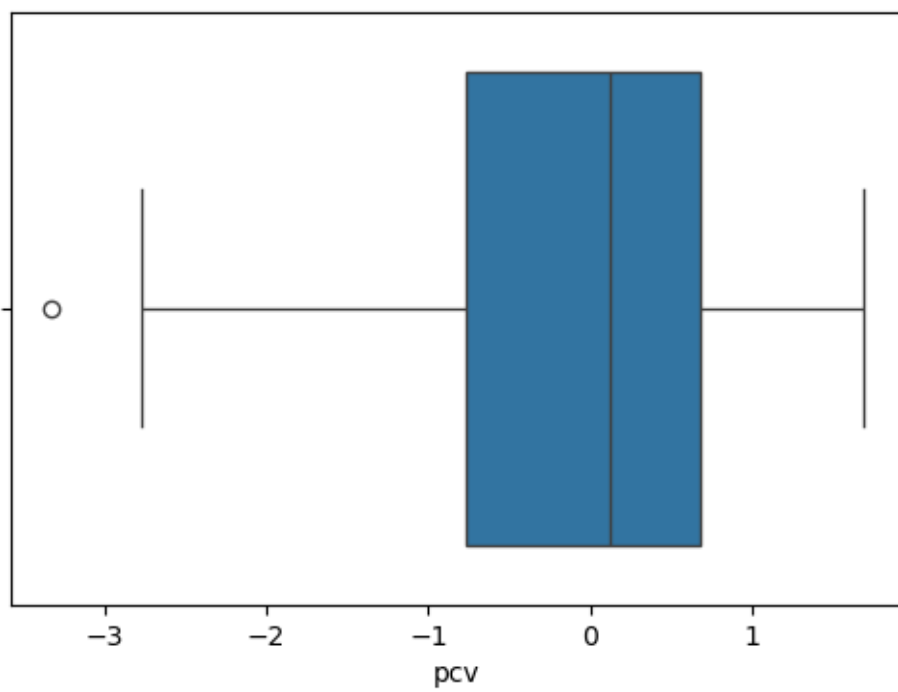
```

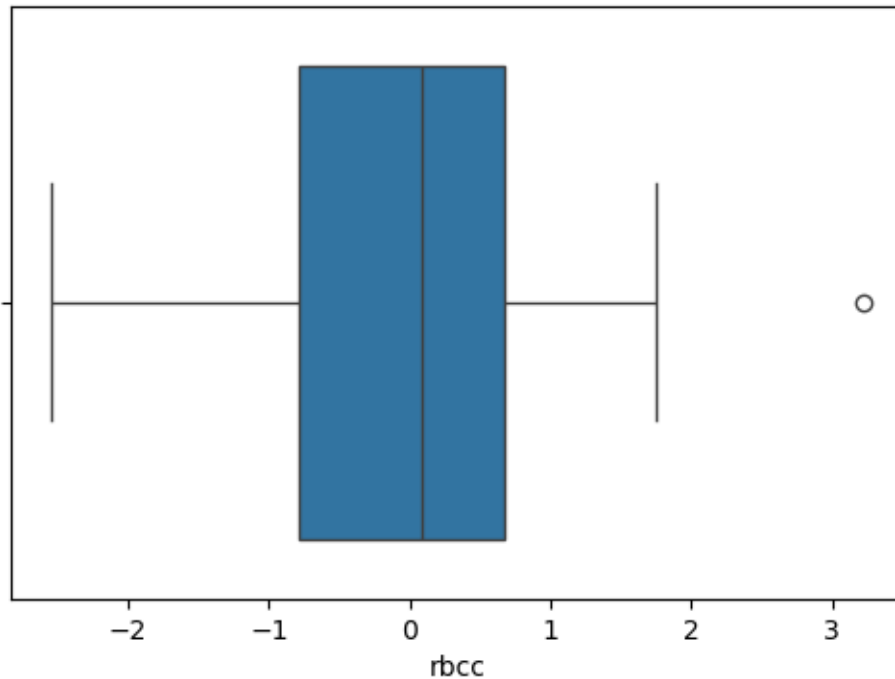












7. **Sub-group Analysis:** Explore potential sub-groups within the data, employing appropriate data science methods to find the sub-groups of patients and visualize the sub-groups. The sub-group analysis must not include the labels (for CKD patients and healthy controls).

```
#prof says use K-means
from sklearn.preprocessing import scale
from sklearn.decomposition import PCA, TruncatedSVD
from sklearn.cluster import KMeans
from scipy.cluster import hierarchy
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.metrics.cluster import rand_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import adjusted_rand_score
from sklearn.decomposition import PCA, TruncatedSVD, FactorAnalysis
import matplotlib.pyplot as plt
from sklearn.metrics import silhouette_score, silhouette_samples
from matplotlib import cm
```

```
y['class'] = y["class"].astype('category').cat.codes
y
```

A value is trying to be set on a copy of a slice from a DataFrame.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/

	class
0	0
1	0
2	0
3	0
4	0
...	...
395	1
396	1
397	1
398	1
399	1

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```



```

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1], dtype=int32)

```

```

# imputer = KNNImputer(n_neighbors=5)
# X_imputed = pd.DataFrame(imputer.fit_transform(X1), columns=X1.columns)
# X_imputed.head()
# X_imputed.isna().sum()

```

```

range_n_clusters = [2, 3, 4, 5, 6]
for n_clusters in range_n_clusters:
    km = KMeans(n_clusters=n_clusters, n_init=20, random_state=0)
    labels = km.fit_predict(X1_drop)

    silhouette_avg = silhouette_score(X1_drop, labels)
    sample_silhouette_values = silhouette_samples(X1_drop, labels)
    fig, ax1 = plt.subplots(1, 1)
    fig.set_size_inches(18, 7)

```

```

ax1.set_xlim([-0.3, 1])

y_lower = 10

for i in range(n_clusters):
    ith_cluster_silhouette_values = sample_silhouette_values[labels == i]
    ith_cluster_silhouette_values.sort()

    size_cluster_i = ith_cluster_silhouette_values.shape[0]
    y_upper = y_lower + size_cluster_i

    # Use the colormap for coloring
    color = cm.nipy_spectral(float(i) / n_clusters)
    ax1.fill_betweenx(
        y=np.arange(y_lower, y_upper),
        x1=0,
        x2=ith_cluster_silhouette_values,
        facecolor=color,
        edgecolor=color,
        alpha=0.7,
    )

    ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

    y_lower = y_upper + 10

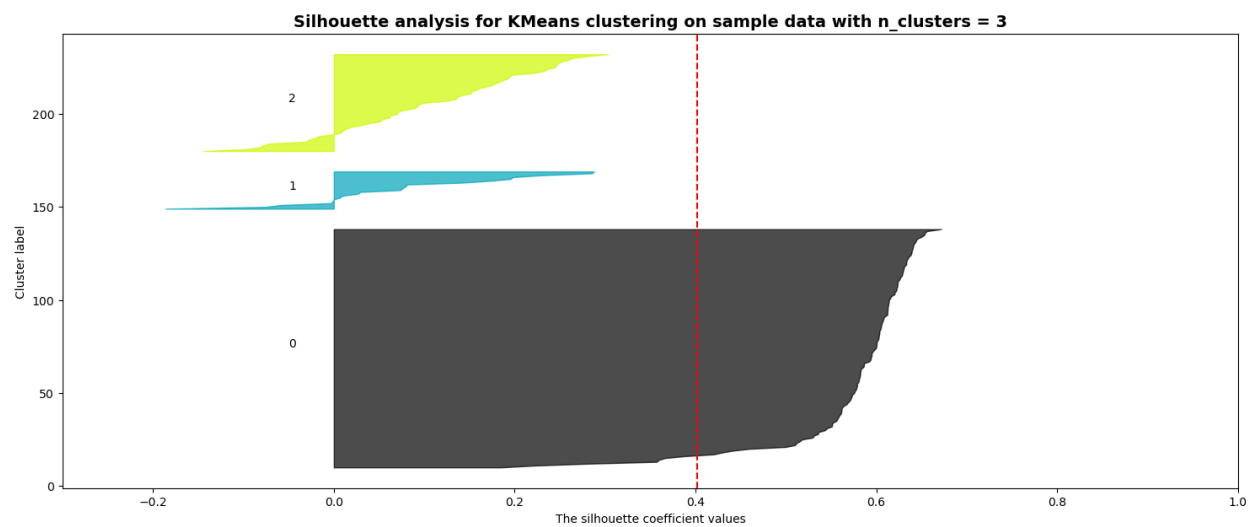
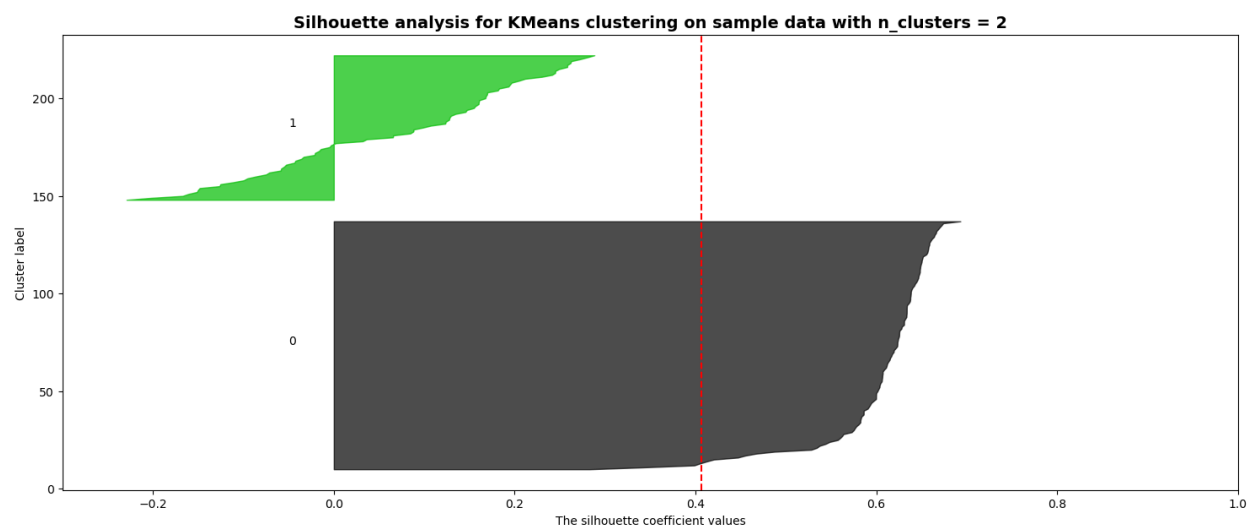
ax1.set_title("The silhouette plot for various clusters")
ax1.set_xlabel("The silhouette coefficient values")
ax1.set_ylabel("Cluster label")

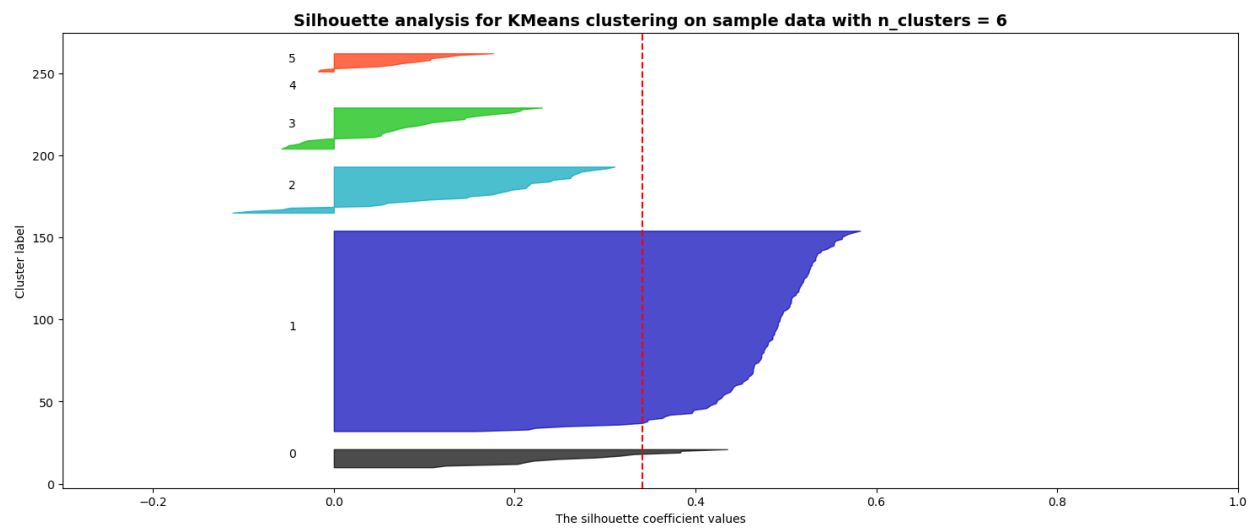
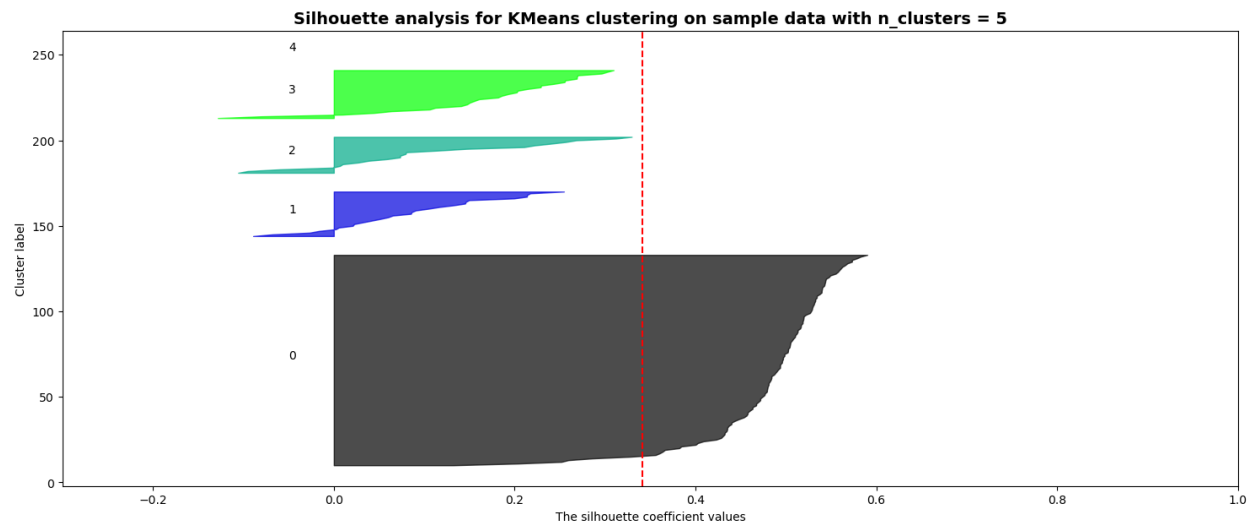
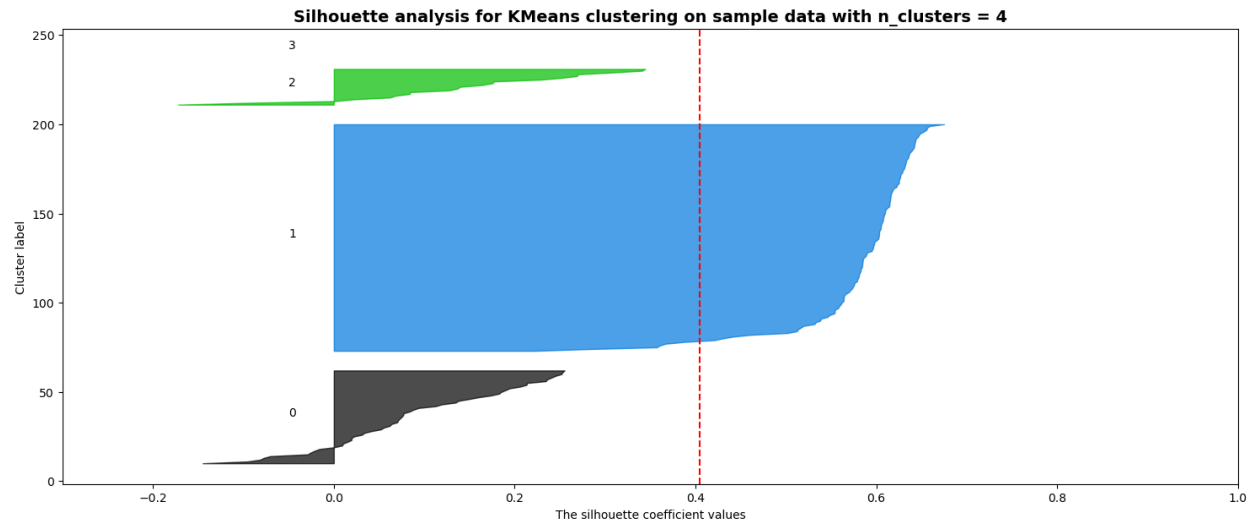
ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

```

```
plt.title(
    "Silhouette analysis for KMeans clustering on sample data with n_clusters = %d"
    % n_clusters,
    fontsize=14,
    fontweight="bold",
)

plt.show()
```





```
kmeans = KMeans(n_clusters=2, n_init=20, random_state=0)
labels = kmeans.fit_predict(y)
labels
```

[illegible]

```
pca_X = PCA()
X = pd.DataFrame(X1_filled, X1_filled.index, X1_filled.columns)
```

```
pca_loading = pd.DataFrame(pca_X.fit(X).components_.T, index=X1_filled.columns, columns=['pc1',
'pc11', 'pc12', 'pc13', 'pc14', 'pc15', 'pc16', 'pc17', 'pc18'
# , 'pc19', 'pc20', 'pc21', 'pc22', 'pc23', 'pc24']
])

pca_loading
```

	pc1	pc2	pc3	pc4	pc5	pc6	pc7	pc8	pc9	pc10	pc11	pc12	pc13	pc14	pc15	pc16	pc17
age	0.227990		0.230990		0.295718		-0.031051		-0.575582		-0.656010		0.133337		0.002424		0.021785
bp	0.218657		0.195864		0.182461		-0.453550		0.717207		-0.375424		-0.036042		0.093023		0.039444
sg	-0.001662		0.000410		0.000035		0.000870		0.000296		-0.000717		-0.001237		0.000471		0.001026
al	0.553457		-0.487707		-0.482395		-0.347007		-0.140393		0.010337		0.180866		0.053118		0.087756
su	0.357932		-0.308756		0.426065		0.400909		0.206522		0.046562		0.154558		-0.475340		0.349520
rbc	-0.160721		-0.314032		-0.325148		0.422191		0.147226		-0.553442		-0.342586		0.238952		0.237721
pc	-0.174043		-0.281904		-0.051507		0.116840		0.165232		-0.277618		0.505481		-0.186919		-0.676205
pcc	0.071946		-0.044355		-0.032420		-0.030809		-0.042205		-0.030729		-0.036475		0.017117		0.033291
ba	0.043953		-0.029370		-0.034667		-0.031891		0.001018		-0.018845		-0.006731		-0.005783		0.039313
bgr	0.337320		-0.236555		0.377282		0.237826		0.014970		0.158208		-0.269555		0.548677		-0.422632
bu	0.348920		0.344499		-0.344170		0.250931		0.056552		-0.049455		-0.410254		-0.465145		-0.346444
sc	0.266765		0.457515		-0.264006		0.424086		0.151383		0.091202		0.523546		0.375210		0.133144
htn	0.187133		0.065908		0.030741		-0.085847		-0.078187		-0.009468		-0.061016		0.053573		-0.091387
dm	0.180008		0.035135		0.115632		-0.019354		-0.062214		0.046096		-0.091541		0.012088		-0.044199
cad	0.061433		0.013987		0.014001		0.013954		-0.026179		-0.008642		0.008890		-0.007925		-0.000104
appet	0.094301		0.028995		-0.029747		-0.083728		-0.020176		0.013642		-0.047902		0.015283		-0.123596
pe	0.099282		0.005992		-0.074849		-0.057811		-0.050081		0.052186		-0.058089		-0.086412		0.013031
ane	0.076863		0.059741		-0.053879		-0.025595		0.035715		0.027753		-0.130954		-0.035215		-0.097805

```
pc_scores = pd.DataFrame(pca_X.fit_transform(X), columns=['pc1', 'pc2', 'pc3', 'pc4', 'pc5', 'pc6', 'pc7', 'pc8', 'pc9', 'pc10', 'pc11', 'pc12', 'pc13', 'pc14', 'pc15', 'pc16', 'pc17', 'pc18'], index=X.index)
pc_scores
```

	pc1	pc2	pc3	pc4	pc5	pc6	pc7	pc8	pc9	pc10	pc11	pc12	pc13	pc14	pc15	pc16	pc17
0	-0.159842		0.123121		0.340298		-1.076715		-0.028920		0.446685		0.604284		-0.200452		-0.546039
1	0.080324		-2.658949		-2.090773		-0.980571		-0.533556		2.899035		1.077587		0.010434		-0.315841
2	2.748405		-2.445865		1.908401		1.610708		0.460511		-0.398526		-0.557039		0.808465		-0.761516
3	1.597823		-1.458225		-2.420973		-0.957120		-0.769911		-0.046338		-0.110910		0.412311		0.653469
4	-0.378850		-1.060810		-0.899482		-0.559600		0.156071		-0.818241		0.426550		0.266685		0.309952

	pc1	pc2	pc3	pc4	pc5	pc6	pc7	pc8	pc9	pc10	pc11	pc12	pc13	pc14	pc15	pc16	pc17
...
395	-1.170494		-0.047818		0.180807		0.277176		0.311051		-0.960873		-0.289116		0.125640		-0.220852
396	-1.872246		-0.239130		-0.396054		0.399566		0.208608		-0.289936		0.067806		-0.182926		0.220656
397	-2.067094		-0.656654		-0.599016		0.127848		1.723678		0.628361		-0.291070		0.061005		0.098355
398	-2.076334		-0.721928		-0.895292		0.973608		0.546345		0.997310		-0.405810		-0.172560		-0.183314
399	-1.355503		-0.144376		0.373565		0.134861		0.189736		-1.053690		0.071790		0.388850		0.057877

```
var=pc_scores.var()
```

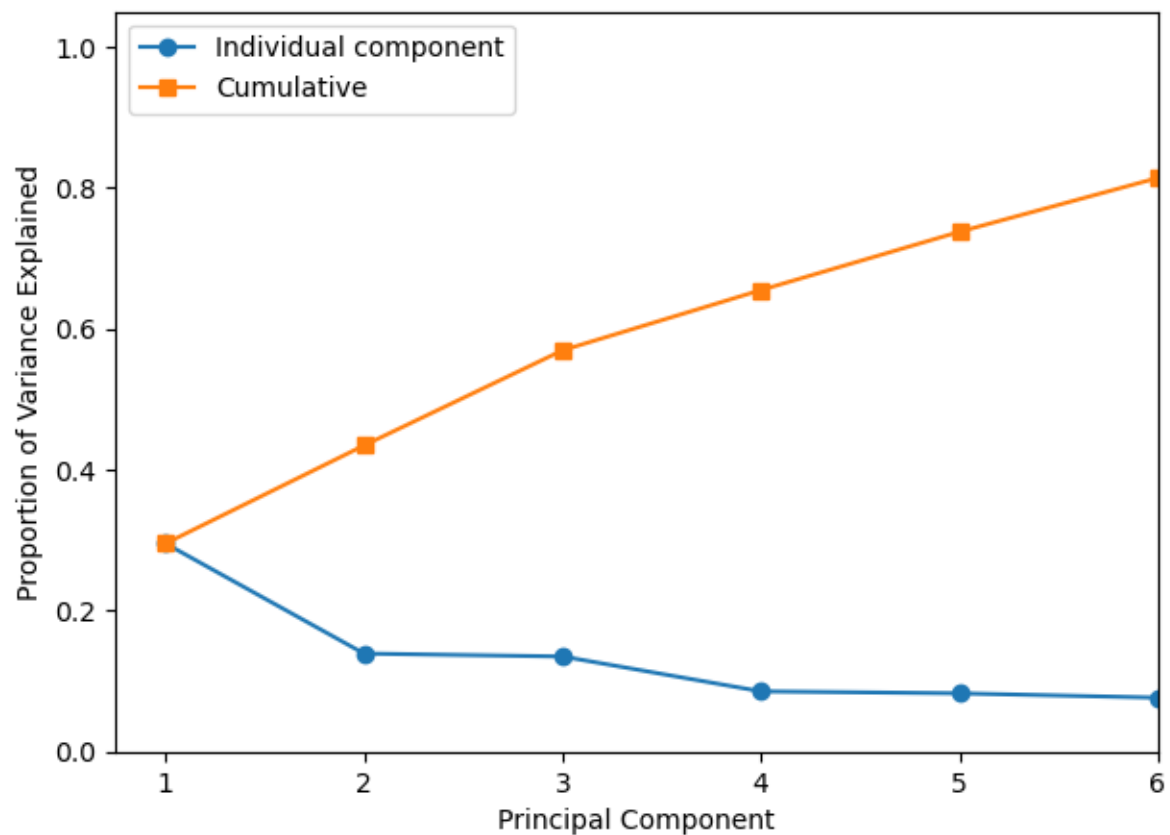
```
var
```

```
pc1      3.011006
pc2      1.418726
pc3      1.376123
pc4      0.871790
pc5      0.843322
pc6      0.780234
pc7      0.413204
pc8      0.356221
pc9      0.333497
pc10     0.193763
pc11     0.135449
pc12     0.107061
pc13     0.098757
pc14     0.082265
pc15     0.071608
pc16     0.058557
pc17     0.044481
pc18     0.000018
dtype: float64
```

```
plt.figure(figsize=(7,5))

plt.plot([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]
, pca_X.explained_variance_ratio_, '-o', label='Individual component')
plt.plot([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]
, np.cumsum(pca_X.explained_variance_ratio_), '-s', label='Cumulative')

plt.ylabel('Proportion of Variance Explained')
plt.xlabel('Principal Component')
plt.xlim(0.75,4.25)
plt.ylim(0,1.05)
plt.xticks([1,2,3,4,5,6])
plt.legend(loc=2);
```



pca_loading

	pc1	pc2	pc3	pc4	pc5	pc6	pc7	pc8	pc9	pc10	pc11	pc12	pc13	pc14	pc15	pc16	pc17
age	0.227990		0.230990		0.295718		-0.031051		-0.575582		-0.656010		0.133337		0.002424		0.021785
bp	0.218657		0.195864		0.182461		-0.453550		0.717207		-0.375424		-0.036042		0.093023		0.039444
sg	-0.001662		0.000410		0.000035		0.000870		0.000296		-0.000717		-0.001237		0.000471		0.001026
al	0.553457		-0.487707		-0.482395		-0.347007		-0.140393		0.010337		0.180866		0.053118		0.087756
su	0.357932		-0.308756		0.426065		0.400909		0.206522		0.046562		0.154558		-0.475340		0.349520
rbc	-0.160721		-0.314032		-0.325148		0.422191		0.147226		-0.553442		-0.342586		0.238952		0.237721
pc	-0.174043		-0.281904		-0.051507		0.116840		0.165232		-0.277618		0.505481		-0.186919		-0.676205
pcc	0.071946		-0.044355		-0.032420		-0.030809		-0.042205		-0.030729		-0.036475		0.017117		0.033291
ba	0.043953		-0.029370		-0.034667		-0.031891		0.001018		-0.018845		-0.006731		-0.005783		0.039313
bgr	0.337320		-0.236555		0.377282		0.237826		0.014970		0.158208		-0.269555		0.548677		-0.422632
bu	0.348920		0.344499		-0.344170		0.250931		0.056552		-0.049455		-0.410254		-0.465145		-0.346444
sc	0.266765		0.457515		-0.264006		0.424086		0.151383		0.091202		0.523546		0.375210		0.133144
htn	0.187133		0.065908		0.030741		-0.085847		-0.078187		-0.009468		-0.061016		0.053573		-0.091387
dm	0.180008		0.035135		0.115632		-0.019354		-0.062214		0.046096		-0.091541		0.012088		-0.044199
cad	0.061433		0.013987		0.014001		0.013954		-0.026179		-0.008642		0.008890		-0.007925		-0.000104
appet	0.094301		0.028995		-0.029747		-0.083728		-0.020176		0.013642		-0.047902		0.015283		-0.123596
pe	0.099282		0.005992		-0.074849		-0.057811		-0.050081		0.052186		-0.058089		-0.086412		0.013031
ane	0.076863		0.059741		-0.053879		-0.025595		0.035715		0.027753		-0.130954		-0.035215		-0.097805

```
PCA_var=sum(pca_X.explained_variance_ratio_[0:2])
```

```
PCA_var
```

```
0.4344542793757309
```

```
plt.figure(figsize=(8, 6))
```

```
principal_components = pca_X.fit_transform(X)
```

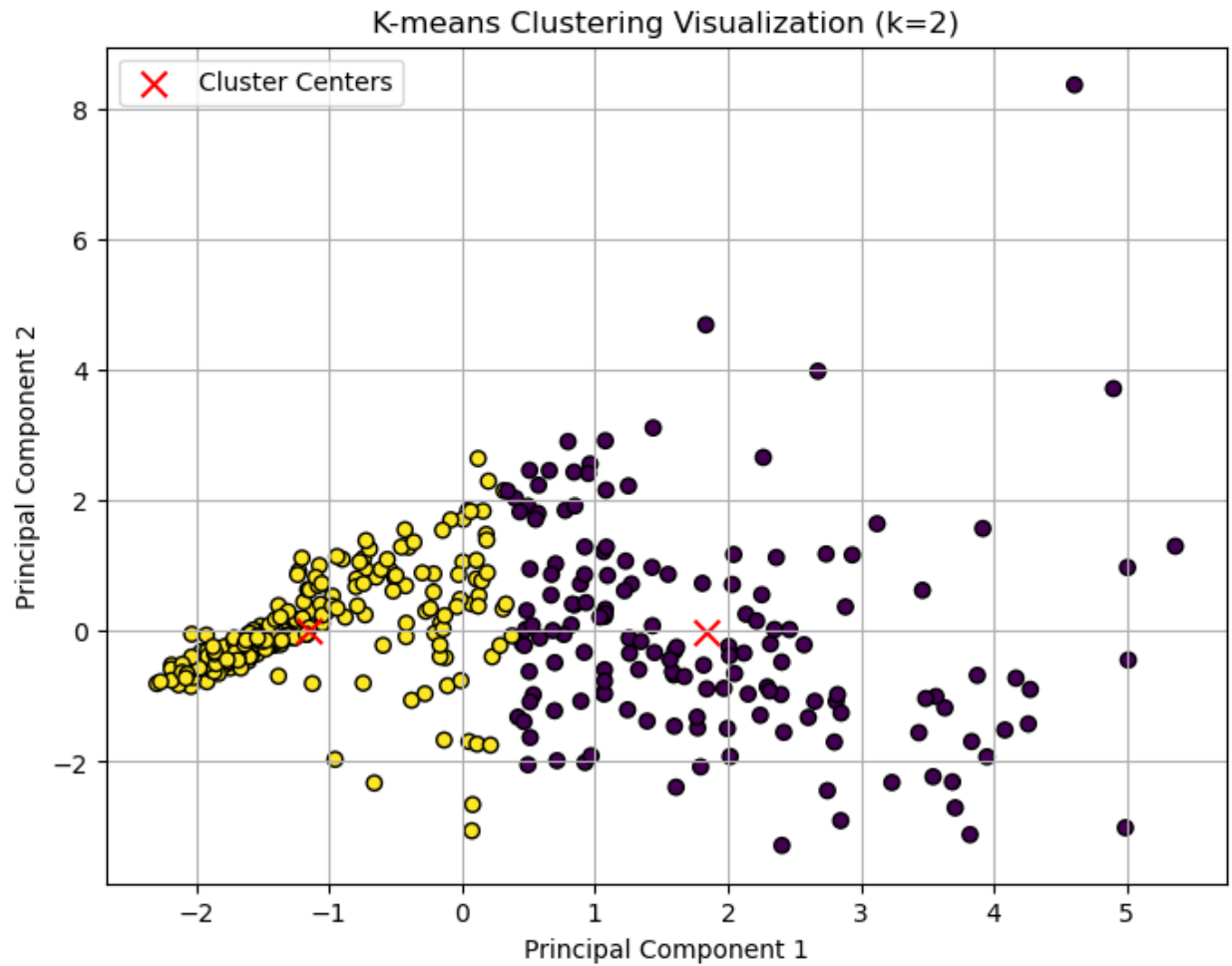
```
kmeans = KMeans(n_clusters=2, n_init=20, random_state=0)
```

```
kmeans.fit(principal_components)
cluster_labels = kmeans.labels_

# Plot data points
plt.scatter(principal_components[:, 0], principal_components[:, 1], c=cluster_labels, cmap='viridis')

# Plot cluster centers
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], marker='x', c='red', s=100)

plt.title('K-means Clustering Visualization (k=2)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.grid(True)
plt.show()
```



```
y_1d = np.ravel(y)
```

```
adjusted_Rand_index = adjusted_rand_score(y_1d, cluster_labels)
```

```
adjusted_Rand_index
```

```
0.2663417112554181
```

8. **Data Splitting:** Segregate 30% of the data for testing, using a random seed of 1. Use the remaining 70% for training and model selection.

```
X_train, X_test, y_train, y_test = train_test_split(
    X1_filled, y, test_size=0.3, random_state=0, stratify=y) #or test_size=0.7
```

9. **Classifier Choices:** Identify the two classifiers you have chosen and justify your selections.

Algorithm 1: Decision tree

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import mean_squared_error, confusion_matrix, classification_report
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, plot_tree
```

```
cs_dt = DecisionTreeClassifier(
    max_depth =30,
    random_state=1
)
```

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

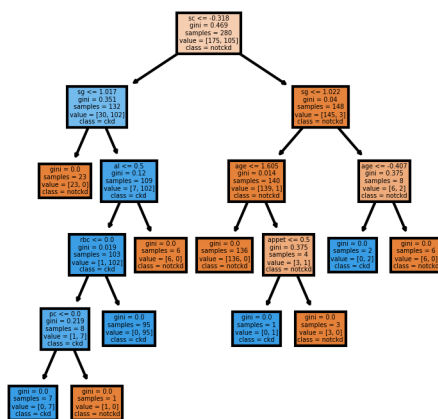
```
DecisionTreeClassifier(max_depth=30, random_state=1)
```

```
fig, axes = plt.subplots(
    nrows = 1,ncols = 1,figsize = (3,3), dpi=300
)
plot_tree(
    cs_dt,
    max_depth= 30,
    feature_names = X_train.columns.tolist(),
    class_names=['notckd', 'ckd'],
    filled = True
)
```

```

[Text(0.4642857142857143, 0.9166666666666666, 'sc <= -0.318\ngini = 0.469\nsamples = 280\nvalue = [175, 105]\nclass = notckd'),
Text(0.21428571428571427, 0.75, 'sg <= 1.017\ngini = 0.351\nsamples = 132\nvalue = [30, 102]\nclass = notckd'),
Text(0.14285714285714285, 0.5833333333333334, 'gini = 0.0\nsamples = 23\nvalue = [23, 0]\nclass = notckd'),
Text(0.2857142857142857, 0.5833333333333334, 'al <= 0.5\ngini = 0.12\nsamples = 109\nvalue = [10, 99]\nclass = notckd'),
Text(0.21428571428571427, 0.4166666666666667, 'rbc <= 0.0\ngini = 0.019\nsamples = 103\nvalue = [103, 0]\nclass = notckd'),
Text(0.14285714285714285, 0.25, 'pc <= 0.0\ngini = 0.219\nsamples = 8\nvalue = [1, 7]\nclass = notckd'),
Text(0.07142857142857142, 0.08333333333333333, 'gini = 0.0\nsamples = 7\nvalue = [0, 7]\nclass = notckd'),
Text(0.21428571428571427, 0.08333333333333333, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]\nclass = notckd'),
Text(0.2857142857142857, 0.25, 'gini = 0.0\nsamples = 95\nvalue = [0, 95]\nclass = ckd'),
Text(0.35714285714285715, 0.4166666666666667, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]\nclass = notckd'),
Text(0.7142857142857143, 0.75, 'sg <= 1.022\ngini = 0.04\nsamples = 148\nvalue = [145, 3]\nclass = notckd'),
Text(0.5714285714285714, 0.5833333333333334, 'age <= 1.605\ngini = 0.014\nsamples = 140\nvalue = [139, 1]\nclass = notckd'),
Text(0.5, 0.4166666666666667, 'gini = 0.0\nsamples = 136\nvalue = [136, 0]\nclass = notckd'),
Text(0.6428571428571429, 0.4166666666666667, 'appet <= 0.5\ngini = 0.375\nsamples = 4\nvalue = [3, 1]\nclass = notckd'),
Text(0.5714285714285714, 0.25, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]\nclass = ckd'),
Text(0.7142857142857143, 0.25, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]\nclass = notckd'),
Text(0.8571428571428571, 0.5833333333333334, 'age <= -0.407\ngini = 0.375\nsamples = 8\nvalue = [7, 1]\nclass = notckd'),
Text(0.7857142857142857, 0.4166666666666667, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]\nclass = notckd'),
Text(0.9285714285714286, 0.4166666666666667, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]\nclass = notckd')
]

```



```

pred = cs_dt.predict(X_test)
pred[:5]

```

```
array([0, 0, 0, 0, 0], dtype=int8)
```

```
cm = pd.DataFrame(confusion_matrix(y_test, pred), index=['No', 'Yes'], columns=['No', 'Yes'])
cm.index.name = 'True'
cm.columns.name = 'Predicted'
cm
```

	Predicted	
	No	Yes
True	No	Yes
	72	3
Yes	1	44

```
print(classification_report(y_test, pred))
```

	precision	recall	f1-score	support
0	0.99	0.96	0.97	75
1	0.94	0.98	0.96	45
accuracy			0.97	120
macro avg	0.96	0.97	0.96	120
weighted avg	0.97	0.97	0.97	120

```
path = cs_dt.cost_complexity_pruning_path(
    X_train,
    y_train
)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
```

```
clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(
```

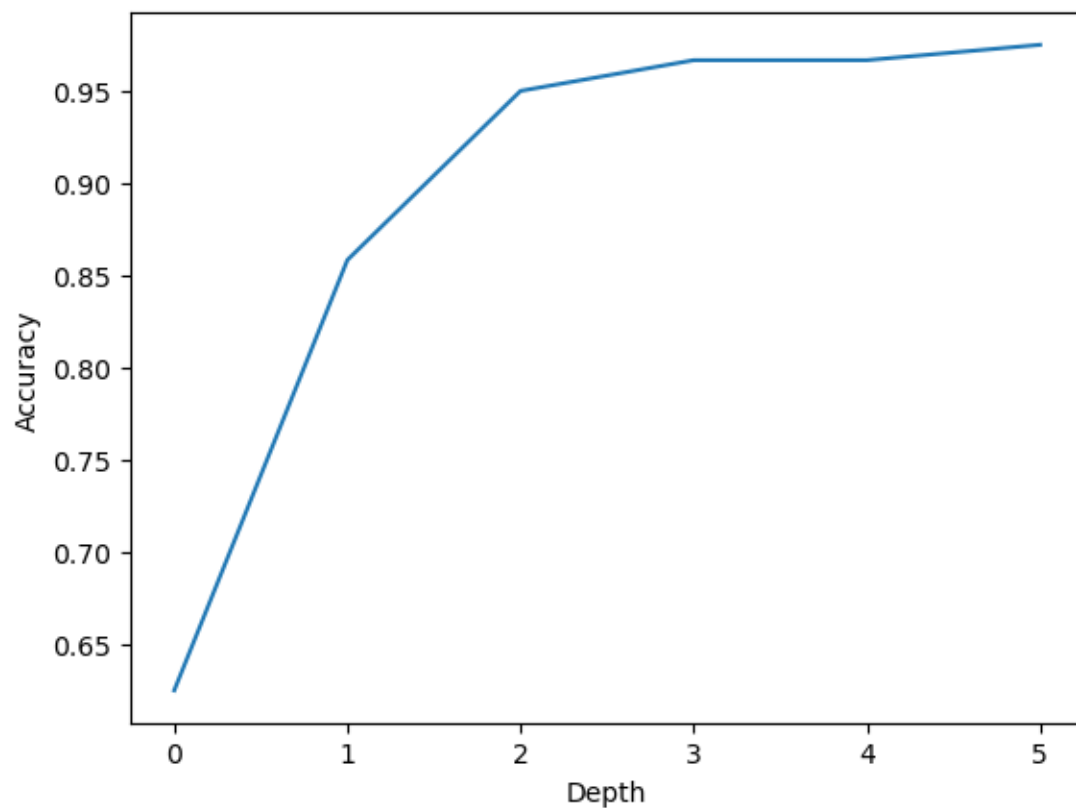
```
    random_state=0,  
    ccp_alpha=ccp_alpha  
)  
clf.fit(X_train, y_train)  
clfs.append(clf)
```

```
depth = [clf.tree_.max_depth for clf in clfs]  
depth
```

```
[5, 4, 3, 3, 2, 1, 0]
```

```
test_score = [clf.score(X_test, y_test) for clf in clfs]
```

```
plt.plot(depth, test_score)  
plt.xlabel('Depth')  
plt.ylabel('Accuracy')  
plt.show()
```



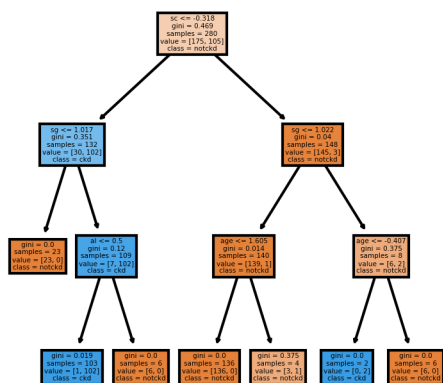
```
cs_dt_new = DecisionTreeClassifier(
    max_depth =3,
    random_state=1
)
```

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

```
DecisionTreeClassifier(max_depth=3, random_state=1)
```

```
fig, axes = plt.subplots(
    nrows = 1,ncols = 1,figsize = (3,3), dpi=300
)
plot_tree(
    cs_dt_new,
    max_depth= 3,
    feature_names = X_train.columns.tolist(),
    class_names=['notckd', 'ckd'],
    filled = True
)
```

```
[Text(0.4230769230769231, 0.875, 'sc <= -0.318\ngini = 0.469\nsamples = 280\nvalue = [175, 105]'),
Text(0.15384615384615385, 0.625, 'sg <= 1.017\ngini = 0.351\nsamples = 132\nvalue = [30, 102]'),
Text(0.07692307692307693, 0.375, 'gini = 0.0\nsamples = 23\nvalue = [23, 0]\nclass = notckd'),
Text(0.23076923076923078, 0.375, 'al <= 0.5\ngini = 0.12\nsamples = 109\nvalue = [7, 102]\nclass = notckd'),
Text(0.15384615384615385, 0.125, 'gini = 0.019\nsamples = 103\nvalue = [1, 102]\nclass = ckd'),
Text(0.3076923076923077, 0.125, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]\nclass = notckd'),
Text(0.6923076923076923, 0.625, 'sg <= 1.022\ngini = 0.04\nsamples = 148\nvalue = [145, 3]\nclass = notckd'),
Text(0.5384615384615384, 0.375, 'age <= 1.605\ngini = 0.014\nsamples = 140\nvalue = [139, 1]\nclass = notckd'),
Text(0.46153846153846156, 0.125, 'gini = 0.0\nsamples = 136\nvalue = [136, 0]\nclass = notckd'),
Text(0.6153846153846154, 0.125, 'gini = 0.375\nsamples = 4\nvalue = [3, 1]\nclass = notckd'),
Text(0.8461538461538461, 0.375, 'age <= -0.407\ngini = 0.375\nsamples = 8\nvalue = [6, 2]\nclass = notckd'),
Text(0.7692307692307693, 0.125, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]\nclass = ckd'),
Text(0.9230769230769231, 0.125, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]\nclass = notckd')]
```

```
pred_new = cs_dt_new.predict(X_test)
pred_new[:5]
```

```
array([0, 0, 0, 0, 0], dtype=int8)
```

```
cm_new = pd.DataFrame(confusion_matrix(y_test, pred_new), index=['No', 'Yes'], columns=['No', 'Yes'],
cm_new.index.name = 'True'
cm_new.columns.name = 'Predicted'
cm_new
```

	Predicted	
True	No	Yes
	No	72
Yes	1	44

```
print(classification_report(y_test, pred_new))
```

```

precision    recall  f1-score   support

0           0.99      0.96      0.97         75
1           0.94      0.98      0.96         45

```

accuracy			0.97	120
macro avg	0.96	0.97	0.96	120
weighted avg	0.97	0.97	0.97	120

Algorithm 2: KNN

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import neighbors
from sklearn.preprocessing import scale
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

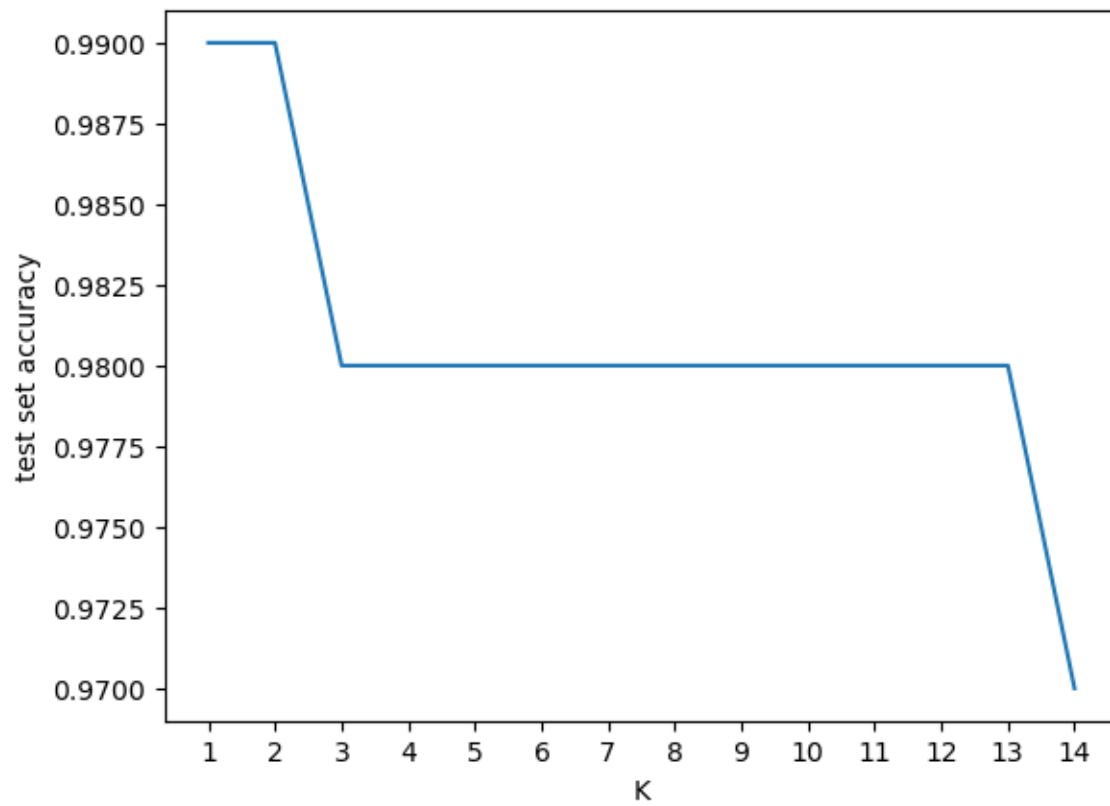
```
x2_train, x2_test, y2_train, y2_test = train_test_split(
    X1_filled,
    y,
    test_size = 0.3,
    random_state = 0,
    stratify = y
)
```

```
k_range = range(1, 15)
scores_r = []

for k in k_range:
    knn_r = neighbors.KNeighborsClassifier(n_neighbors=k)
    y_train_flat = np.ravel(y2_train)
    knn_r.fit(x2_train, y_train_flat)
```



```
plt.plot(k_range, scores_r)
plt.xlabel('K')
plt.ylabel('test set accuracy')
plt.xticks(range(1,15))
plt.show()
```



```
knn3 = neighbors.KNeighborsClassifier(
    n_neighbors = 3,
    algorithm = 'brute'
)

y_train_flat = np.ravel(y2_train)

knn3.fit(x2_train, y_train_flat)
```

```
KNeighborsClassifier(algorithm='brute', n_neighbors=3)
```

```
pred3 = knn3.predict(x2_test.values)
pred3[1:5]
```

```
/Users/shujiashang/anaconda3/envs/proj02/lib/python3.9/site-packages/sklearn/base.py:464: UserWarning:
  warnings.warn(

array([0, 0, 0, 0], dtype=int8)
```

```
round(metrics.accuracy_score(y2_test, pred3), 2)
```

0.98

```
conf_matrix = confusion_matrix(y2_test, pred3)
conf_matrix
```

```
array([[73,  2],
       [ 0, 45]])
```

10. **Performance Metrics:** Outline the two metrics for comparing the performance of the classifiers.

```
cm = pd.DataFrame(confusion_matrix(y_test, pred), index=['No', 'Yes'], columns=['No', 'Yes'])
cm.index.name = 'True'
cm.columns.name = 'Predicted'
cm
```

	No	Yes
Predicted		
True		
No	72	3
Yes	1	44

```
conf_matrix = confusion_matrix(y2_test, pred3)
conf_matrix
```

```
array([[73,  2],
       [ 0, 45]])
```

11. **Feature Selection/Extraction:** Implement methods to enhance the performance of at least one classifier in (9). The answer for this question can be included in (12).
12. **Classifier Comparison:** Utilize the selected metrics to compare the classifiers based on the test set. Discuss your findings (at least two statements).
13. **Interpretable Classifier Insight:** After re-training the interpretable classifier with all available data, analyze and interpret the significance of predictor variables in the context of the data and the challenge (at least two statements).
14. **[Bonus] Sub-group Improvement Strategy:** If sub-groups were identified, propose and implement a method to improve one classifier performance further. Compare the performance of the new classifier with the results in (12).
15. **Team Contributions:** Document each team member's specific contributions related to the questions above.
16. **Link** to the public GitHub repository.

<https://github.com/dail9/3da-assignment6/blob/main/assignment6.ipynb>

Rubini, Soundarapandian, L., and P. Eswaran. 2015. "Chronic Kidney Disease." UCI Machine Learning Repository.