Early Prediction of Chronic Kidney Disease using Machine Learning

Chronic Kidney Disease(CKD) is a critical condition caused due to kidney malfunction or kidney malignancy. Early prediction of such diseases will help address the issue at an early stage and suppress it as much as possible

Team ID: SWTID1720090815

Objective: *Develop a model capable of diagnosing a person for Chornic Kidney Disease as precisely and acurately as possible.*

Step: 1 Importing necessary modules

In [ ]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split,GridSearchCV,KFold
from sklearn.feature_selection import RFECV
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score,confusion_matrix,ConfusionMatrixDisplay,classification_report, preci
from copulas.multivariate import GaussianMultivariate
from xgboost import XGBClassifier
from imblearn.over_sampling import SMOTE
from pickle import dump
import warnings
```

In [ ]:
```
warnings.filterwarnings("ignore") #Removing warnings of depricated features of certain modules in the output
```

Step: 2 reading and storing the dataset in a pandas Dataframe.

*The dataset **chronickidneydisease.csv** is downloaded from UC Irvine Machine Learning Repository*

In [ ]:
```
kidney_data= pd.read_csv("chronickidneydisease.csv")
kidney_data.reset_index(drop=True,inplace=True)
kidney_data.head()
```

Out[ ]:

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	cl
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	...	44	7800	5.2	yes	yes	no	good	no	no	
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	...	38	6000	NaN	no	no	no	good	no	no	
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31	7500	NaN	no	yes	no	poor	no	yes	
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32	6700	3.9	yes	no	no	poor	yes	yes	
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	...	35	7300	4.6	no	no	no	good	no	no	

5 rows × 26 columns

Step: 3 Understanding Data Type and the Summary of Features

Dropping unnecessary column: `id`

In [ ]:
```
kidney_data.drop(columns=['id'],inplace=True)
```

In [ ]:
```
kidney_data.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    float64
 3   al              354 non-null    float64
 4   su              351 non-null    float64
 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             330 non-null    object 
 16  wc              295 non-null    object 
 17  rc              270 non-null    object 
 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  classification  400 non-null    object 
dtypes: float64(11), object(14)
memory usage: 78.3+ KB
```

Attributes and their meanings

Attributes	Attributes Meaning
Age	Age
bp	Blood Pressure
sg	Specific Gravity
al	Albumin
su	Sugar
rbc	Red Blood Cells
pc	Pus Cell
pcc	Pus Cell Clumps
ba	Bacteria
bgr	Blood Glucose random
bu	Blood Urea
sc	Serum Creatinine
sod	Sodium
pot	Pottasium
Hemo	Haemoglobin
pcv	Packed Cell Volume
wc	White Blood Cell Count
rb	Red Blood Cell Count
htn	Hypertension
dm	Diabetes mellitus
cad	Coronary Artery Disease
appet	Appetite
pe	Peda Edema
ane	Anemia
Classification	Class

In [ ]:
```
features = {'age':'Age','bp':'Blood_Pressure','sg':'Specific_gravity','al':'Albumin','su':'Sugar',
            'rbc':'Red_blood_cells','pc':'Pus_cell','pcc':'Pus_cell_clumps','ba':'Bacteria',
            'bgr':'Blood_Glucose_Random','bu':'Blood_urea','sc':'Serum_creatinine','sod':'Sodium',
            'pot':'Potassium','hemo':'Haemoglobin','pcv':'Packed_cell_volume','wc':'White_blood_cell_count',
```

```
            'rc':'Red_blood_cell_count','htn':'Hypertension','dm':'Diabetes_mellitus','cad':'Coronary_artery_di
            'appet':'Appetite','pe':'Peda_edema','ane':'Anemia'}
```

Renaming Columns

In [ ]:
```
kidney_data.rename(columns=features)
```

Out[ ]:

	Age	Blood_Pressure	Specific_gravity	Albumin	Sugar	Red_blood_cells	Pus_cell	Pus_cell_clumps	Bacteria	Blood_Gluco
0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	
1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	
2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	
3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	
4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	
...	...	...	...	...	...	...	...	...	...	
395	55.0	80.0	1.020	0.0	0.0	normal	normal	notpresent	notpresent	
396	42.0	70.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	
397	12.0	80.0	1.020	0.0	0.0	normal	normal	notpresent	notpresent	
398	17.0	60.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	
399	58.0	80.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	

400 rows × 25 columns

Identifying categorical and continuous type features for better understanding of unique values and handling missing values.

In [ ]:
```
categorical=pd.DataFrame(columns=['Categorical Columns','Unique Values'])
object_columns=list(kidney_data.columns[kidney_data.dtypes=='O'])
for columns in object_columns:
    unique_val= kidney_data[columns].unique()
    categorical= pd.concat([categorical, pd.DataFrame({'Categorical Columns': [columns], 'Unique Values': [uniq
categorical
```

Out[ ]:

	Categorical Columns	Unique Values
0	rbc	[nan, normal, abnormal]
1	pc	[normal, abnormal, nan]
2	pcc	[notpresent, present, nan]
3	ba	[notpresent, present, nan]
4	pcv	[44, 38, 31, 32, 35, 39, 36, 33, 29, 28, nan, ...
5	wc	[7800, 6000, 7500, 6700, 7300, nan, 6900, 9600...
6	rc	[5.2, nan, 3.9, 4.6, 4.4, 5, 4.0, 3.7, 3.8, 3....
7	htn	[yes, no, nan]
8	dm	[yes, no, yes, \tno, \tyes, nan]
9	cad	[no, yes, \tno, nan]
10	appet	[good, poor, nan]
11	pe	[no, yes, nan]
12	ane	[no, yes, nan]
13	classification	[ckd, ckd\t, notckd]

Based on the above dataset, we will clean the categorical features to achieve proper values

In [ ]:
```
kidney_data['dm']=kidney_data['dm'].replace('\tno','no').replace('\tyes','yes').replace(' yes','yes')
kidney_data['cad']=kidney_data['cad'].replace('\tno','no')
kidney_data['classification']=kidney_data['classification'].replace('ckd\t','ckd')
```

In [ ]:
```
Continuous=pd.DataFrame(columns=['Continuous Columns','Unique Values'])
object_columns=list(kidney_data.columns[kidney_data.dtypes!='O'])
for columns in object_columns:
    unique_val= kidney_data[columns].unique()
    Continuous = pd.concat([Continuous, pd.DataFrame({'Continuous Columns': [columns], 'Unique Values': [unique_
Continuous
```

Out[ ]:

	Continuous Columns	Unique Values
0	age	[48.0, 7.0, 62.0, 51.0, 60.0, 68.0, 24.0, 52.0...
1	bp	[80.0, 50.0, 70.0, 90.0, nan, 100.0, 60.0, 110...
2	sg	[1.02, 1.01, 1.005, 1.015, nan, 1.025]
3	al	[1.0, 4.0, 2.0, 3.0, 0.0, nan, 5.0]
4	su	[0.0, 3.0, 4.0, 1.0, nan, 2.0, 5.0]
5	bgr	[121.0, nan, 423.0, 117.0, 106.0, 74.0, 100.0,...
6	bu	[36.0, 18.0, 53.0, 56.0, 26.0, 25.0, 54.0, 31....
7	sc	[1.2, 0.8, 1.8, 3.8, 1.4, 1.1, 24.0, 1.9, 7.2,...
8	sod	[nan, 111.0, 142.0, 104.0, 114.0, 131.0, 138.0...
9	pot	[nan, 2.5, 3.2, 4.0, 3.7, 4.2, 5.8, 3.4, 6.4, ...
10	hemo	[15.4, 11.3, 9.6, 11.2, 11.6, 12.2, 12.4, 10.8...

From the above two Dataframes, it is observed that `pcv`, `rc`, and `wc` are of continuous type. Whereas, `sg`, `al`, and `su` are of categorical type.
The columns will hence be transfered to their appropriate dataframes.

In [ ]:
```
Continuous=pd.concat([Continuous,pd.DataFrame({'Continuous Columns':['pcv'],'Unique Values':[categorical.iloc[4
Continuous=pd.concat([Continuous,pd.DataFrame({'Continuous Columns':['rc'],'Unique Values':[categorical.iloc[6]
Continuous=pd.concat([Continuous,pd.DataFrame({'Continuous Columns':['wc'],'Unique Values':[categorical.iloc[5]
categorical=pd.concat([categorical,pd.DataFrame({'Categorical Columns':['sg'],'Unique Values':[Continuous.iloc[
categorical=pd.concat([categorical,pd.DataFrame({'Categorical Columns':['al'],'Unique Values':[Continuous.iloc[
categorical=pd.concat([categorical,pd.DataFrame({'Categorical Columns':['su'],'Unique Values':[Continuous.iloc[
Continuous.drop(index=[2,3,4],inplace=True)
categorical.drop(index=[4,5,6],inplace=True)
```

Converting `pcv`, `rc`, and `wc` to numeric type

In [ ]:
```
kidney_data['pcv']=pd.to_numeric(kidney_data['pcv'],errors='coerce')
kidney_data['rc']=pd.to_numeric(kidney_data['rc'],errors='coerce')
kidney_data['wc']=pd.to_numeric(kidney_data['wc'],errors='coerce')
kidney_data['sg']=kidney_data['sg'].astype(object)
kidney_data['al']=kidney_data['al'].astype(object)
kidney_data['su']=kidney_data['su'].astype(object)
```

Cleaned and updated features:

In [ ]:
```
categorical
```

Out[ ]:

	Categorical Columns	Unique Values
0	rbc	[nan, normal, abnormal]
1	pc	[normal, abnormal, nan]
2	pcc	[notpresent, present, nan]
3	ba	[notpresent, present, nan]
7	htn	[yes, no, nan]
8	dm	[yes, no, yes, \tno, \tyes, nan]
9	cad	[no, yes, \tno, nan]
10	appet	[good, poor, nan]
11	pe	[no, yes, nan]
12	ane	[no, yes, nan]
13	classification	[ckd, ckd\t, notckd]
14	sg	[1.02, 1.01, 1.005, 1.015, nan, 1.025]
15	al	[1.0, 4.0, 2.0, 3.0, 0.0, nan, 5.0]
16	su	[0.0, 3.0, 4.0, 1.0, nan, 2.0, 5.0]

In [ ]:
```
Continuous
```

Out[ ]:

	Continuous Columns	Unique Values
0	age	[48.0, 7.0, 62.0, 51.0, 60.0, 68.0, 24.0, 52.0...
1	bp	[80.0, 50.0, 70.0, 90.0, nan, 100.0, 60.0, 110...
5	bgr	[121.0, nan, 423.0, 117.0, 106.0, 74.0, 100.0,...
6	bu	[36.0, 18.0, 53.0, 56.0, 26.0, 25.0, 54.0, 31....
7	sc	[1.2, 0.8, 1.8, 3.8, 1.4, 1.1, 24.0, 1.9, 7.2,...
8	sod	[nan, 111.0, 142.0, 104.0, 114.0, 131.0, 138.0...
9	pot	[nan, 2.5, 3.2, 4.0, 3.7, 4.2, 5.8, 3.4, 6.4, ...
10	hemo	[15.4, 11.3, 9.6, 11.2, 11.6, 12.2, 12.4, 10.8...
11	pcv	[44, 38, 31, 32, 35, 39, 36, 33, 29, 28, nan, ...
12	rc	[5.2, nan, 3.9, 4.6, 4.4, 5, 4.0, 3.7, 3.8, 3....
13	wc	[7800, 6000, 7500, 6700, 7300, nan, 6900, 9600...

Step: 4 Handling Missing Values

Checking for null values

In [ ]:
```
kidney_data.isnull().sum()
```

Out[ ]:
```
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
wc               106
rc               131
htn                2
dm                 2
cad                2
appet              1
pe                 1
ane                1
classification     0
dtype: int64
```

Step: 5 Replacing Missing Values

Addressing missing values appropriately based on their feature type i.e.

Feature type	Method of addressing
Categorical	Mode
Continuous	Mean

In [ ]:
```
#Mode of Categorical features
kidney_data['age']=kidney_data['age'].fillna(kidney_data['age'].mode()[0])
kidney_data['rbc']=kidney_data['rbc'].fillna(kidney_data['rbc'].mode()[0])
kidney_data['pc']=kidney_data['pc'].fillna(kidney_data['pc'].mode()[0])
kidney_data['pcc']=kidney_data['pcc'].fillna(kidney_data['pcc'].mode()[0])
kidney_data['ba']=kidney_data['ba'].fillna(kidney_data['ba'].mode()[0])
kidney_data['htn']=kidney_data['htn'].fillna(kidney_data['htn'].mode()[0])
kidney_data['dm']=kidney_data['dm'].fillna(kidney_data['dm'].mode()[0])
kidney_data['cad']=kidney_data['cad'].fillna(kidney_data['cad'].mode()[0])
kidney_data['appet']=kidney_data['appet'].fillna(kidney_data['appet'].mode()[0])
kidney_data['pe']=kidney_data['pe'].fillna(kidney_data['pe'].mode()[0])
kidney_data['ane']=kidney_data['ane'].fillna(kidney_data['ane'].mode()[0])
kidney_data['sg']=kidney_data['sg'].fillna(kidney_data['sg'].mode()[0])
kidney_data['al']=kidney_data['al'].fillna(kidney_data['al'].mode()[0])
kidney_data['su']=kidney_data['su'].fillna(kidney_data['su'].mode()[0])
#Mean of continuous columns
kidney_data['bp']=kidney_data['bp'].fillna(kidney_data['bp'].mean())
kidney_data['bgr']=kidney_data['bgr'].fillna(kidney_data['bgr'].mean())
kidney_data['bu']=kidney_data['bu'].fillna(kidney_data['bu'].mean())
```

```
kidney_data['sc']=kidney_data['sc'].fillna(kidney_data['sc'].mean())
kidney_data['sod']=kidney_data['sod'].fillna(kidney_data['sod'].mean())
kidney_data['pot']=kidney_data['pot'].fillna(kidney_data['pot'].mean())
kidney_data['hemo']=kidney_data['hemo'].fillna(kidney_data['hemo'].mean())
kidney_data['pcv']=kidney_data['pcv'].fillna(kidney_data['pcv'].mean())
kidney_data['rc']=kidney_data['rc'].fillna(kidney_data['rc'].mean())
kidney_data['wc']=kidney_data['wc'].fillna(kidney_data['wc'].mean())
```

In [ ]:
```
kidney_data.isnull().sum()
```

Out[ ]:
```
age               0
bp                0
sg                0
al                0
su                0
rbc               0
pc                0
pcc               0
ba                0
bgr               0
bu                0
sc                0
sod               0
pot               0
hemo              0
pcv               0
wc                0
rc                0
htn               0
dm                0
cad               0
appet             0
pe                0
ane               0
classification    0
dtype: int64
```

Missing values have been addressed successfully

Step: 6 Analyzing the dataset

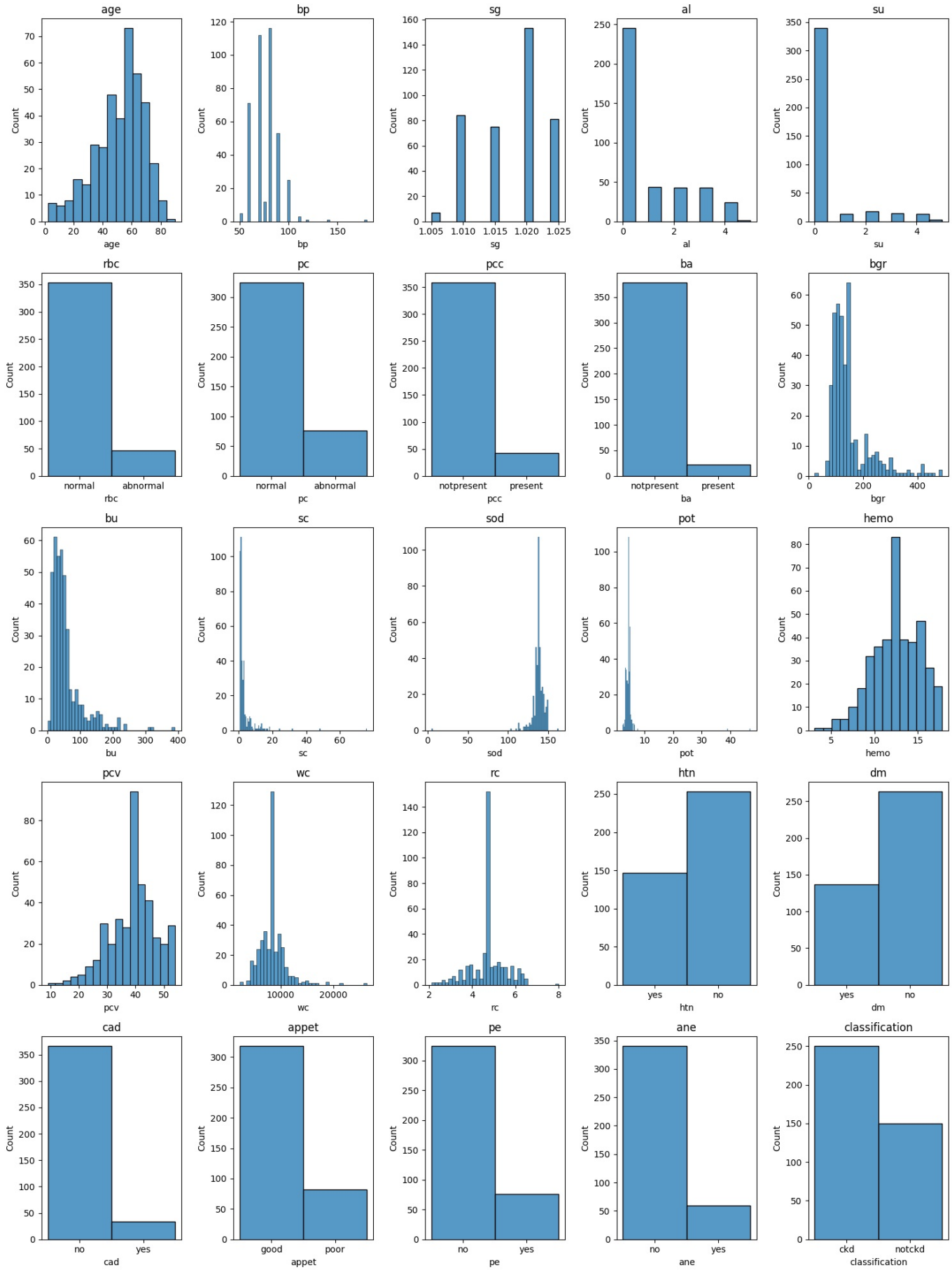
Univariate Analysis

In [ ]:
```
kidney_data.describe()
```

Out[ ]:

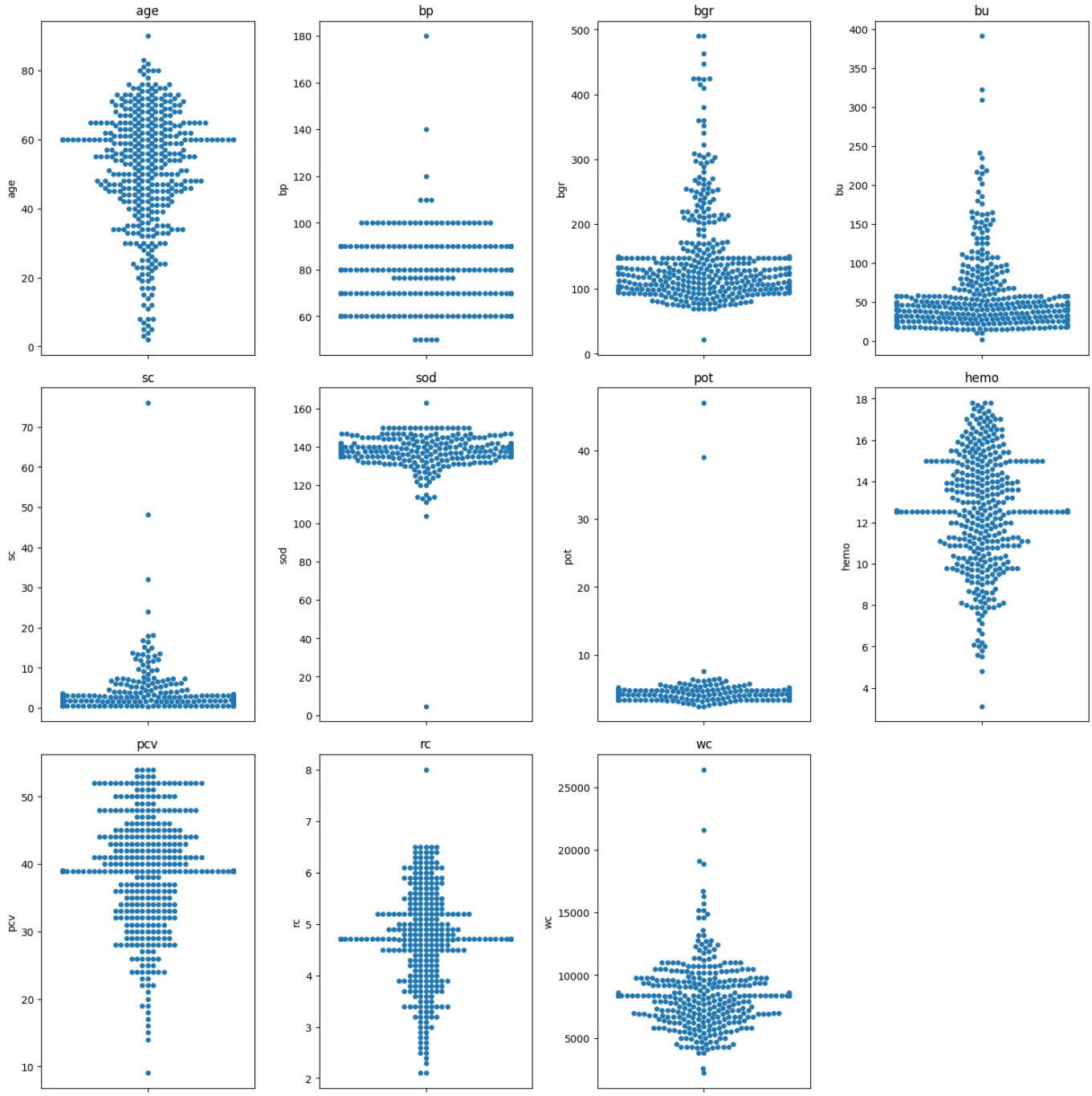
	age	bp	sg	al	su	bgr	bu	sc	sod	pot	
count	400.000000	400.000000	400.000000	400.00000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	40
mean	51.675000	76.469072	1.017712	0.90000	0.395000	148.036517	57.425722	3.072454	137.528754	4.627244	1
std	17.022008	13.476298	0.005434	1.31313	1.040038	74.782634	49.285887	5.617490	9.204273	2.819783	
min	2.000000	50.000000	1.005000	0.00000	0.000000	22.000000	1.500000	0.400000	4.500000	2.500000	
25%	42.000000	70.000000	1.015000	0.00000	0.000000	101.000000	27.000000	0.900000	135.000000	4.000000	1
50%	55.000000	78.234536	1.020000	0.00000	0.000000	126.000000	44.000000	1.400000	137.528754	4.627244	1
75%	64.000000	80.000000	1.020000	2.00000	0.000000	150.000000	61.750000	3.072454	141.000000	4.800000	1
max	90.000000	180.000000	1.025000	5.00000	5.000000	490.000000	391.000000	76.000000	163.000000	47.000000	1

In [ ]:
```
num=0
plt.figure(figsize=(15,20))
for col in kidney_data.columns:
  num+=1
  plt.subplot(5,5,num)
  sns.histplot(kidney_data[col])
  plt.title(f'{col}')
  plt.tight_layout()
plt.show()
```



Observation: `kidney_data` is imbalanced. This will be addressed while splitting the dataset into independent and target variables

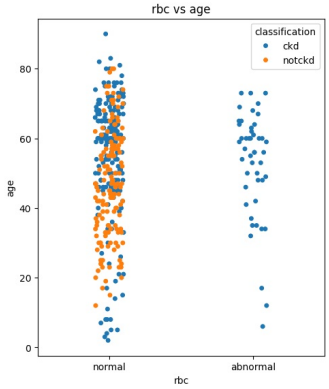
In [ ]:
```
num=0
plt.figure(figsize=(15,30))
for col in Continuous['Continuous Columns']:
  num+=1
  plt.subplot(6,4,num)
  sns.swarmplot(kidney_data[col])
  plt.title(f'{col}')
  plt.tight_layout()
plt.show()
```

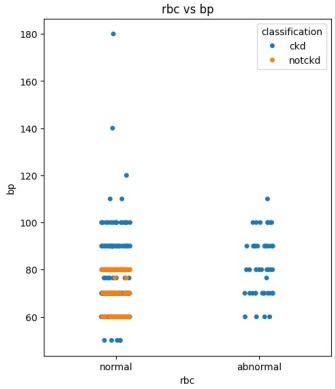


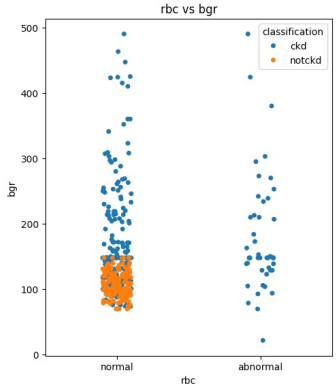
Observations: Outliers present in almost all the continuous features. These outliers will be retained as in the medical field there are probability of rare/anamolous cases. These cases are cruicial for early prediction of CKD

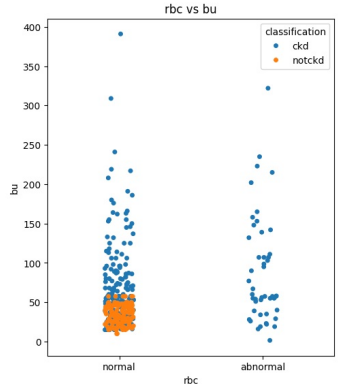
Bivariate Analysis

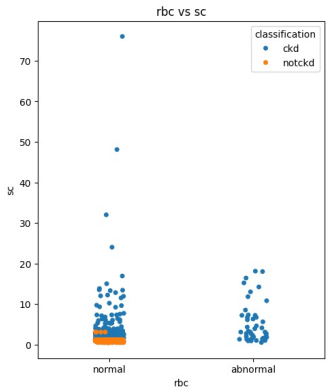
In [ ]:
```
numerical_features=list(Continuous['Continuous Columns'])
columns = list(kidney_data.columns)
for i in columns:
  row=6
  col=4
  if kidney_data[i].dtype=='object':
    plt_counter=1
    plt.figure(figsize=(25,45))
    for j in numerical_features:
      plt.subplot(row,col,plt_counter)
      sns.stripplot(data=kidney_data,x=i,y=j,hue='classification')
      plt.title(f'{i} vs {j}' )
      plt_counter+=1
    plt.show()
    row+=1
```

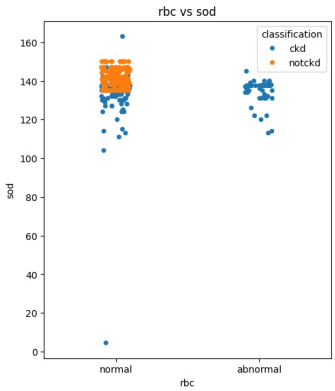

rbc vs age
classification
ckd
notckd
age
rbc
normal
abnormal

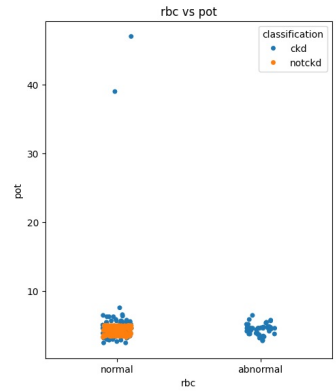

rbc vs bp
classification
ckd
notckd
bp
rbc
normal
abnormal

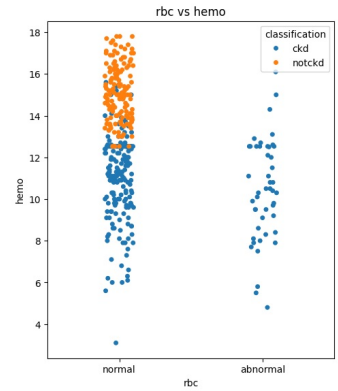

rbc vs bgr
classification
ckd
notckd
bgr
rbc
normal
abnormal

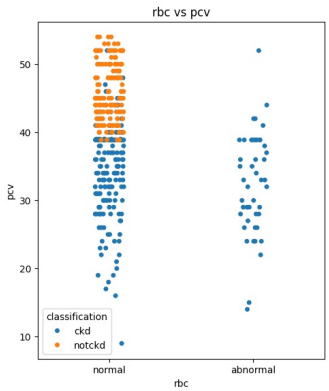

rbc vs bu
classification
ckd
notckd
bu
rbc
normal
abnormal

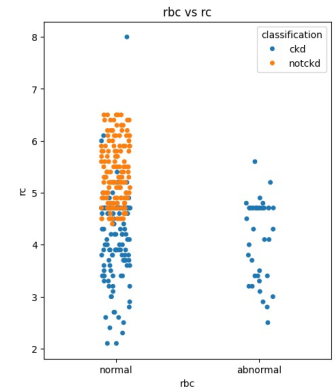

rbc vs sc
classification
ckd
notckd
sc
rbc
normal
abnormal

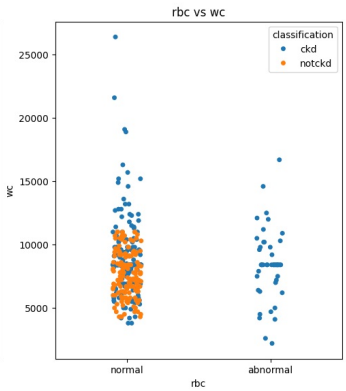

rbc vs sod
classification
ckd
notckd
sod
rbc
normal
abnormal

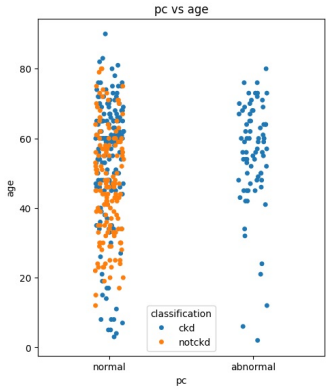

rbc vs pot
classification
ckd
notckd
pot
rbc
normal
abnormal

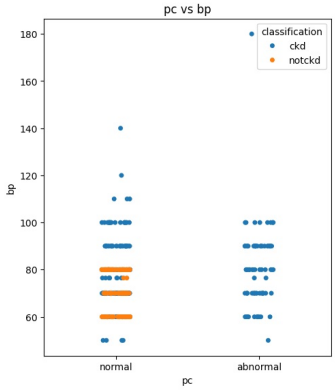

rbc vs hemo
classification
ckd
notckd
hemo
rbc
normal
abnormal

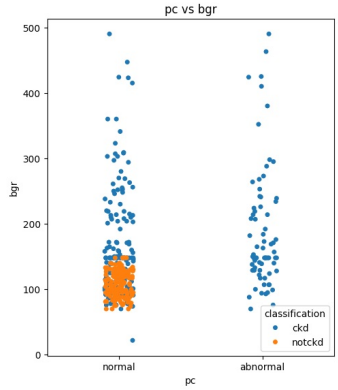

rbc vs pcv
classification
ckd
notckd
pcv
rbc
normal
abnormal

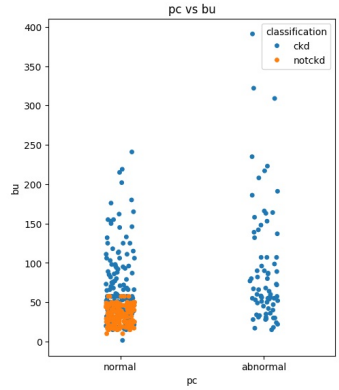

rbc vs rc
classification
ckd
notckd
rc
rbc
normal
abnormal

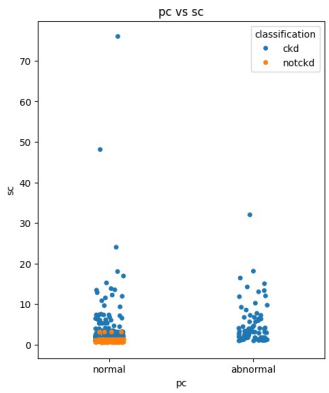

rbc vs wc
classification
ckd
notckd
wc
rbc
normal
abnormal

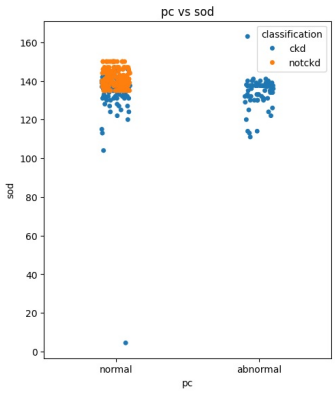

pc vs age
age
pc
normal
abnormal
classification
ckd
notckd

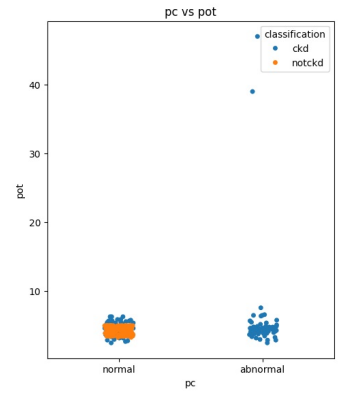

pc vs bp
bp
pc
normal
abnormal
classification
ckd
notckd

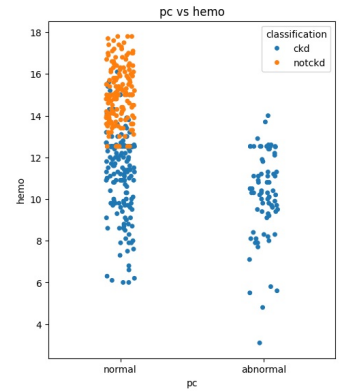

pc vs bgr
bgr
pc
normal
abnormal
classification
ckd
notckd

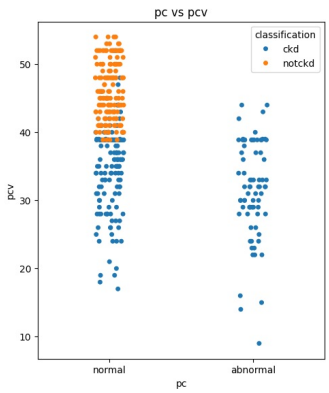

pc vs bu
bu
pc
normal
abnormal
classification
ckd
notckd

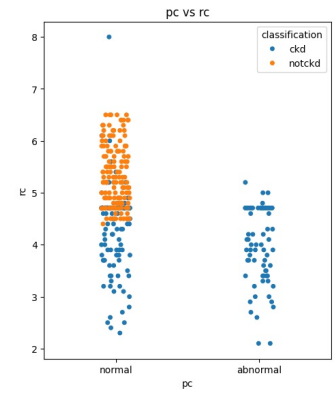

pc vs sc
sc
pc
normal
abnormal
classification
ckd
notckd

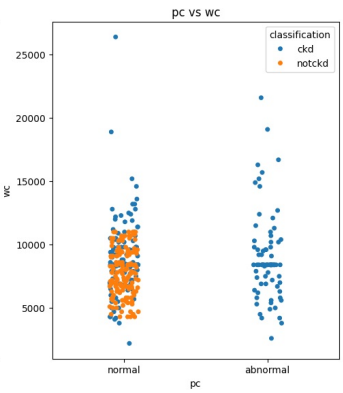

pc vs sod
sod
pc
normal
abnormal
classification
ckd
notckd

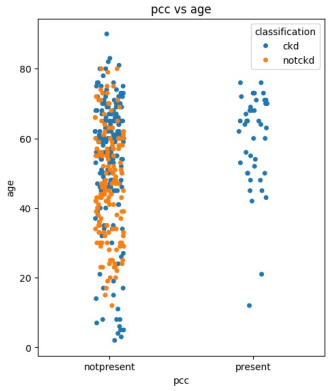

pc vs pot
pot
pc
normal
abnormal
classification
ckd
notckd

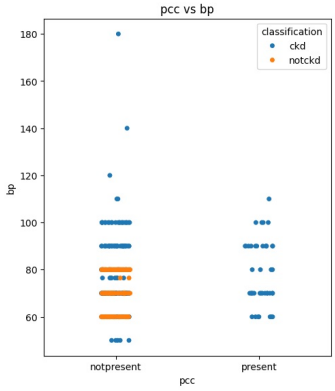

pc vs hemo
hemo
pc
normal
abnormal
classification
ckd
notckd

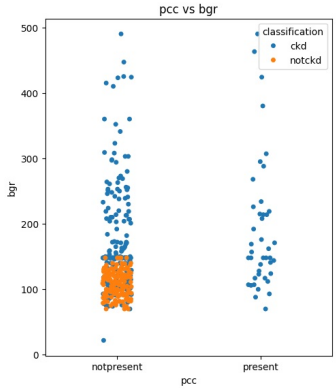

pc vs pcv
pcv
pc
normal
abnormal
classification
ckd
notckd

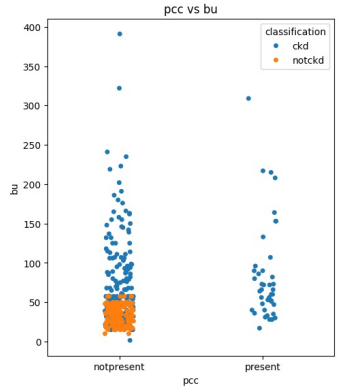

pc vs rc
rc
pc
normal
abnormal
classification
ckd
notckd

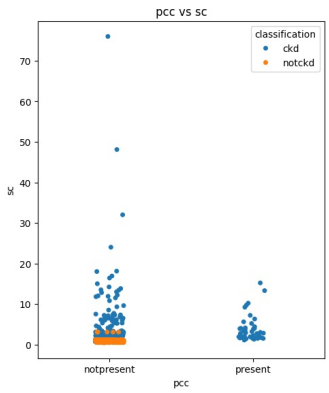

pc vs wc
wc
pc
normal
abnormal
classification
ckd
notckd

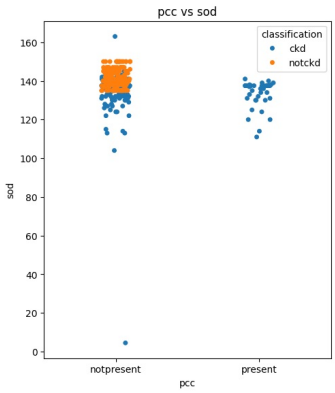

pcc vs age
classification
ckd
notckd
age
pcc
notpresent
present

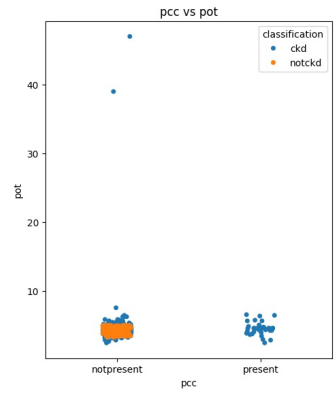

pcc vs bp
classification
ckd
notckd
bp
pcc
notpresent
present

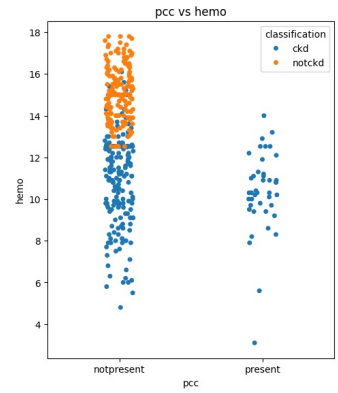

pcc vs bgr
classification
ckd
notckd
bgr
pcc
notpresent
present

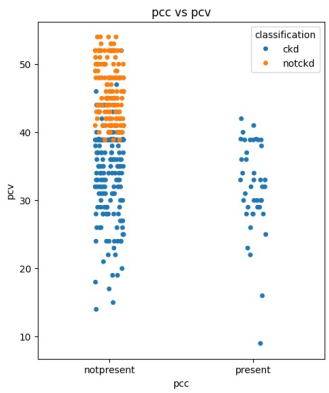

pcc vs bu
classification
ckd
notckd
bu
pcc
notpresent
present

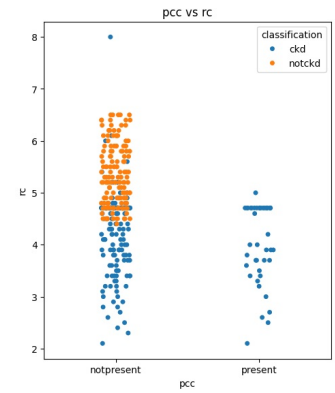

pcc vs sc
classification
ckd
notckd
sc
pcc
notpresent
present

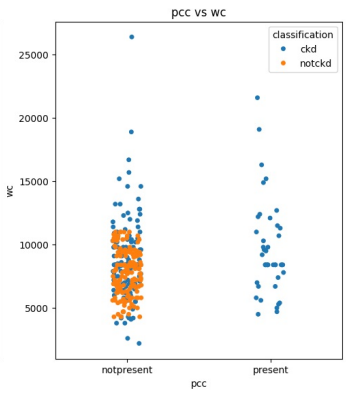

pcc vs sod
classification
ckd
notckd
sod
pcc
notpresent
present

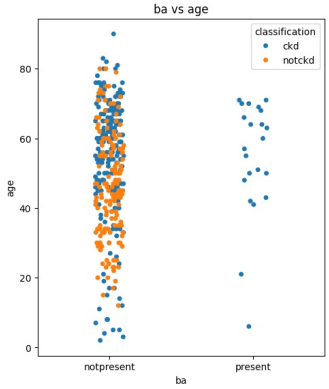

pcc vs pot
classification
ckd
notckd
pot
pcc
notpresent
present

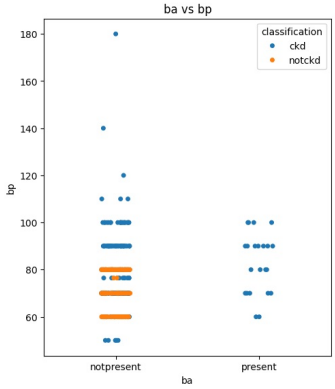

pcc vs hemo
classification
ckd
notckd
hemo
pcc
notpresent
present

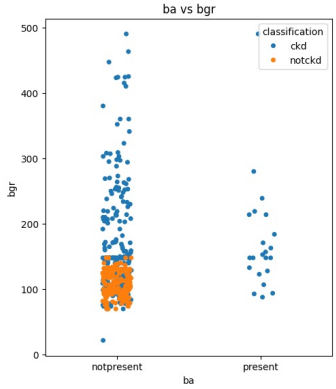

pcc vs pcv
classification
ckd
notckd
pcv
pcc
notpresent
present

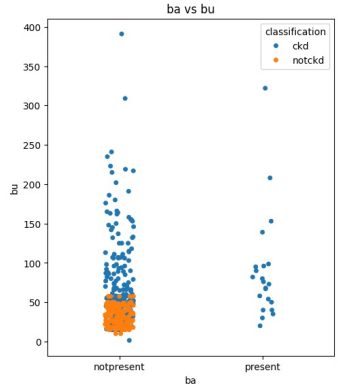

pcc vs rc
classification
ckd
notckd
rc
pcc
notpresent
present

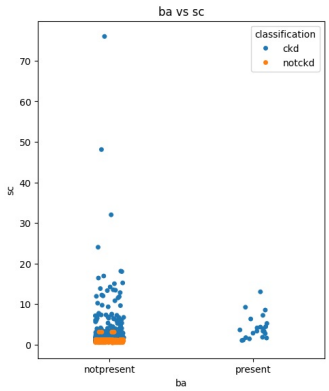

pcc vs wc
classification
ckd
notckd
wc
pcc
notpresent
present

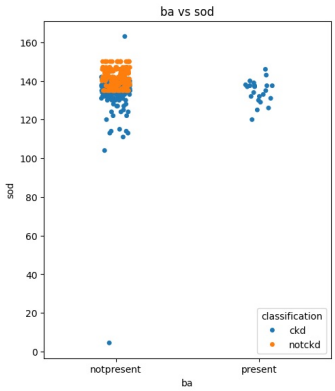

ba vs age
classification
ckd
notckd
age
ba
notpresent
present

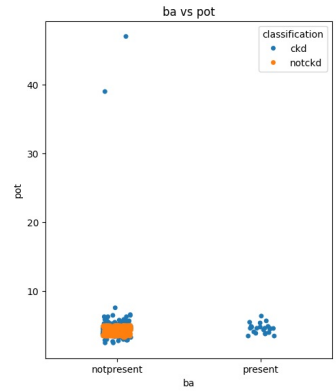

ba vs bp
classification
ckd
notckd
bp
ba
notpresent
present

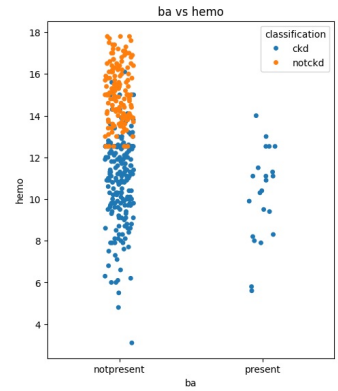

ba vs bgr
classification
ckd
notckd
bgr
ba
notpresent
present

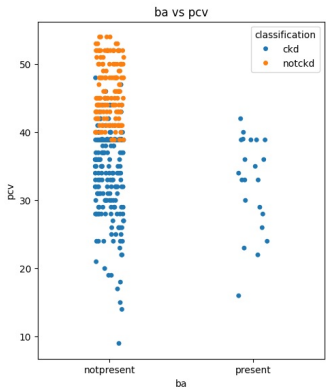

ba vs bu
classification
ckd
notckd
bu
ba
notpresent
present

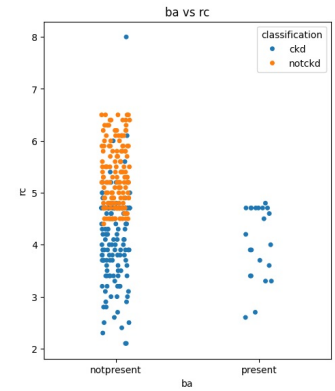

ba vs sc
classification
ckd
notckd
sc
ba
notpresent
present

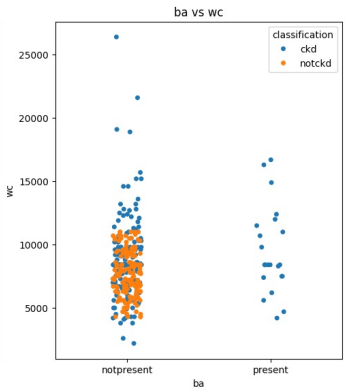

ba vs sod
classification
ckd
notckd
sod
ba
notpresent
present

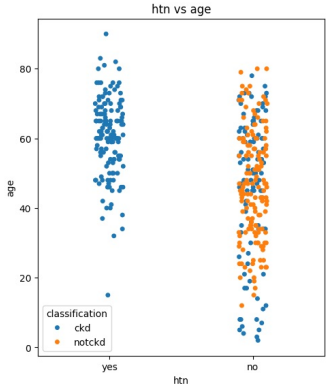

ba vs pot
classification
ckd
notckd
pot
ba
notpresent
present

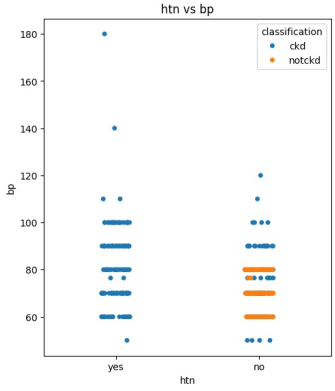

ba vs hemo
classification
ckd
notckd
hemo
ba
notpresent
present

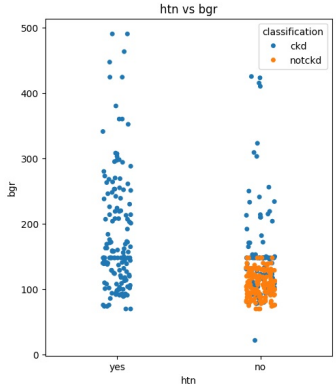

ba vs pcv
classification
ckd
notckd
pcv
ba
notpresent
present

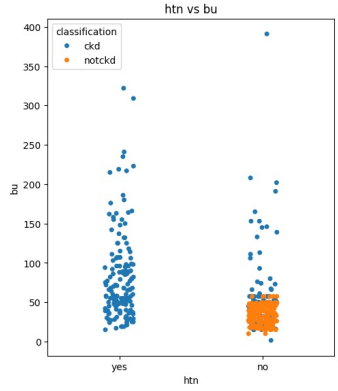

ba vs rc
classification
ckd
notckd
rc
ba
notpresent
present

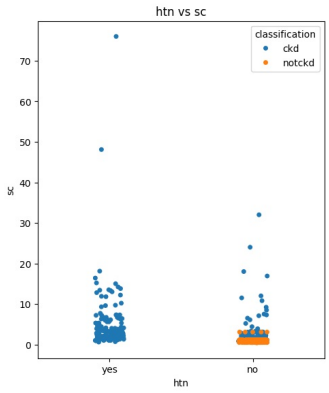

ba vs wc
classification
ckd
notckd
wc
ba
notpresent
present

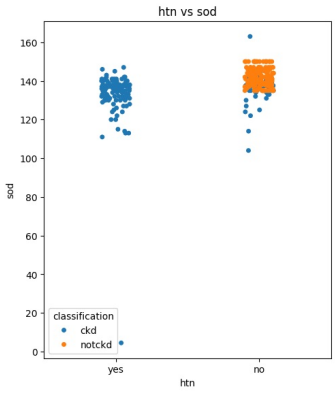

htn vs age
classification
ckd
notckd
age
htn
yes
no

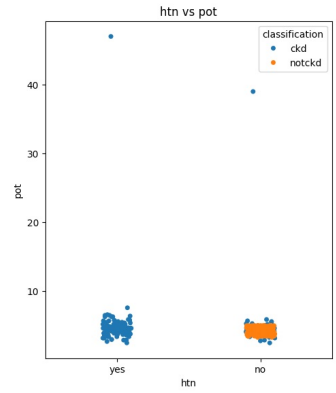

htn vs bp
classification
ckd
notckd
bp
htn
yes
no

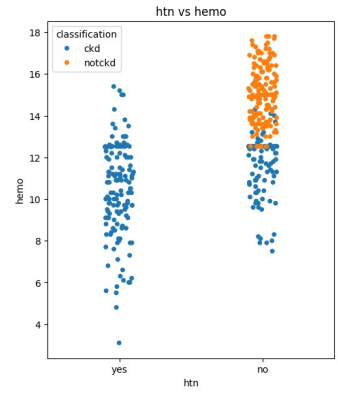

htn vs bgr
classification
ckd
notckd
bgr
htn
yes
no

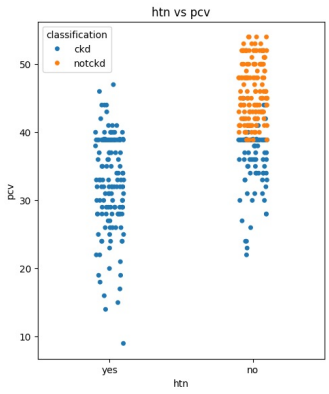

htn vs bu
classification
ckd
notckd
bu
htn
yes
no

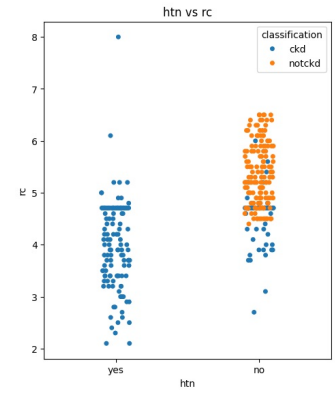

htn vs sc
classification
ckd
notckd
sc
htn
yes
no

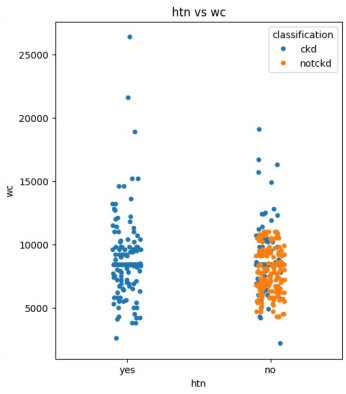

htn vs sod
classification
ckd
notckd
sod
htn
yes
no

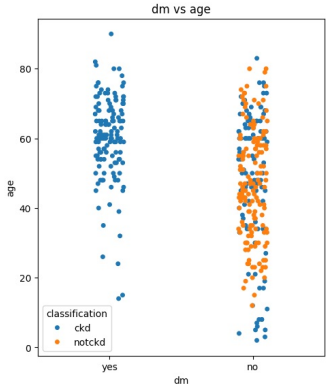

htn vs pot
classification
ckd
notckd
pot
htn
yes
no

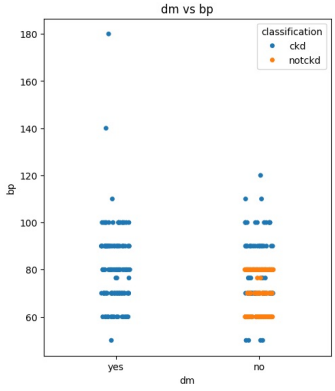

htn vs hemo
classification
ckd
notckd
hemo
htn
yes
no

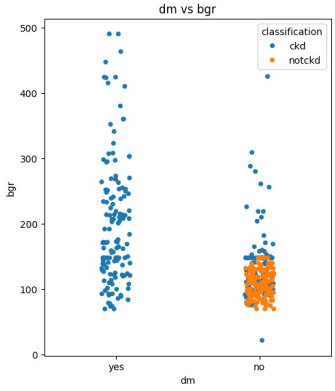

htn vs pcv
classification
ckd
notckd
pcv
htn
yes
no

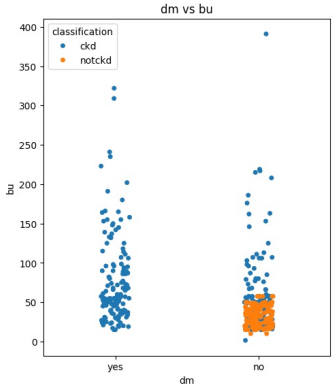

htn vs rc
classification
ckd
notckd
rc
htn
yes
no

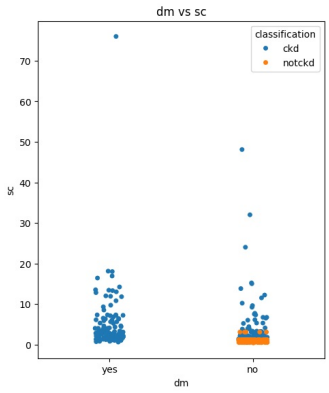

htn vs wc
classification
ckd
notckd
wc
htn
yes
no

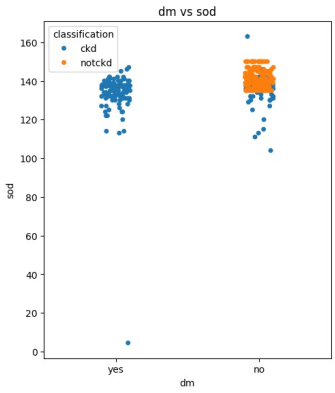

dm vs age
age
dm
yes
no
classification
ckd
notckd

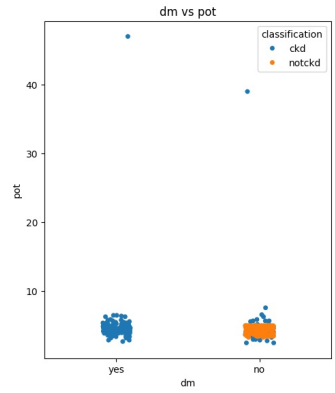

dm vs bp
bp
dm
yes
no
classification
ckd
notckd

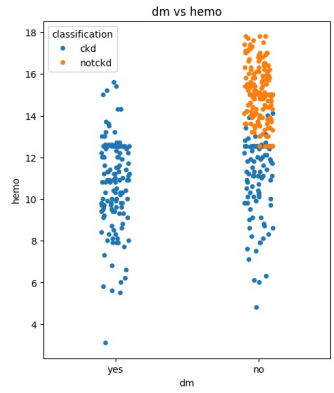

dm vs bgr
bgr
dm
yes
no
classification
ckd
notckd

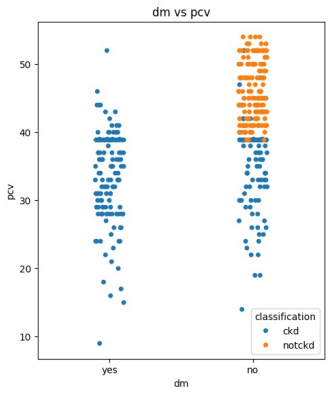

dm vs bu
bu
dm
yes
no
classification
ckd
notckd

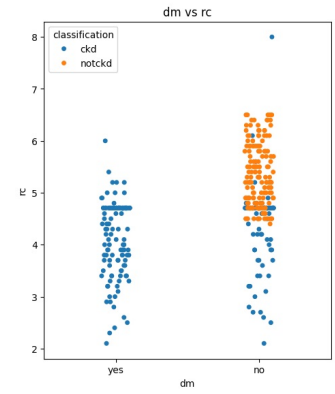

dm vs sc
sc
dm
yes
no
classification
ckd
notckd

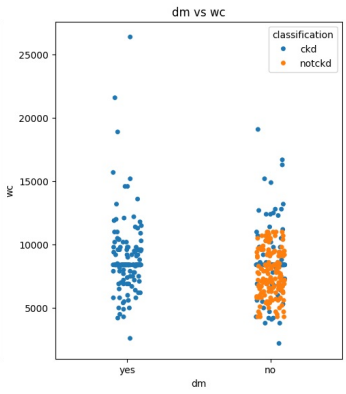

dm vs sod
sod
dm
yes
no
classification
ckd
notckd

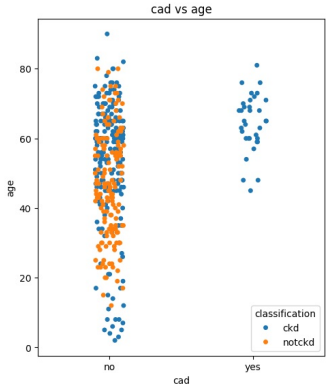

dm vs pot
pot
dm
yes
no
classification
ckd
notckd

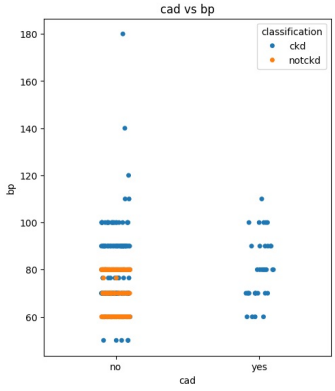

dm vs hemo
hemo
dm
yes
no
classification
ckd
notckd

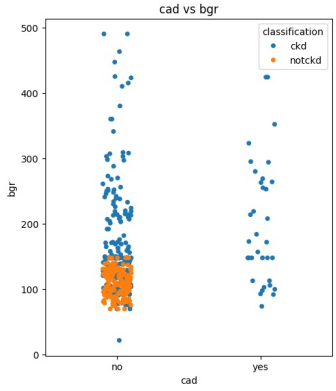

dm vs pcv
pcv
dm
yes
no
classification
ckd
notckd

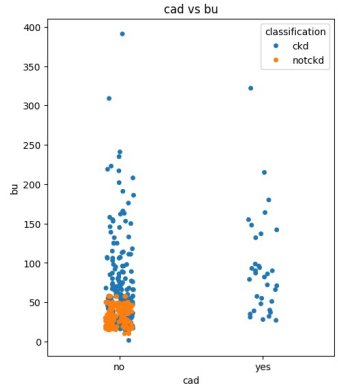

dm vs rc
rc
dm
yes
no
classification
ckd
notckd

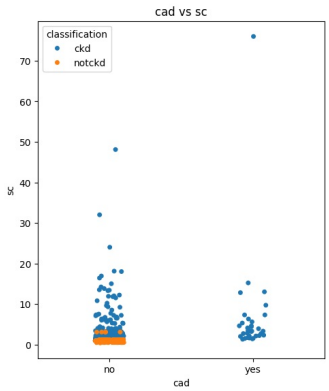

dm vs wc
wc
dm
yes
no
classification
ckd
notckd

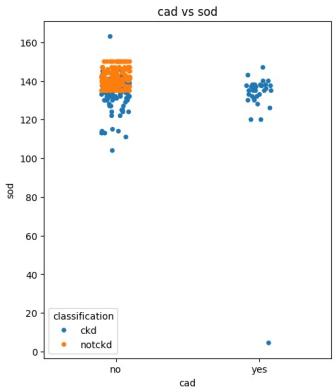

cad vs age
classification
ckd
notckd
age
cad
no
yes

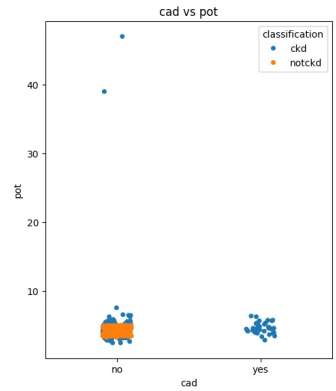

cad vs bp
classification
ckd
notckd
bp
cad
no
yes

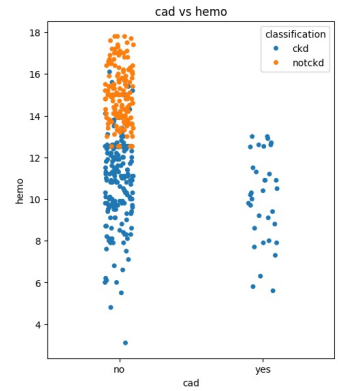

cad vs bgr
classification
ckd
notckd
bgr
cad
no
yes

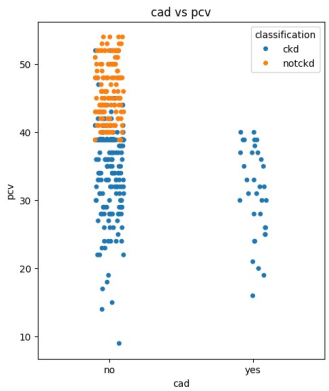

cad vs bu
classification
ckd
notckd
bu
cad
no
yes

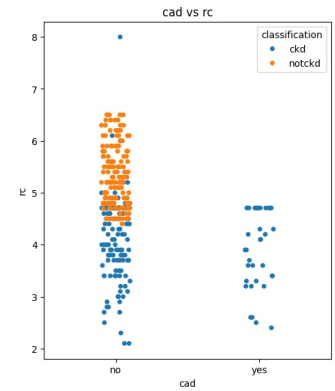

cad vs sc
classification
ckd
notckd
sc
cad
no
yes

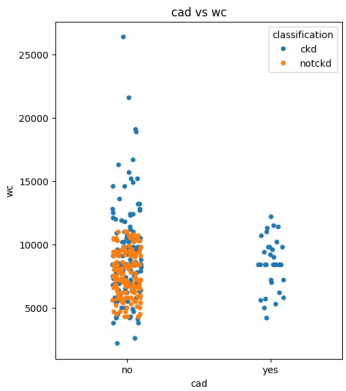

cad vs sod
classification
ckd
notckd
sod
cad
no
yes

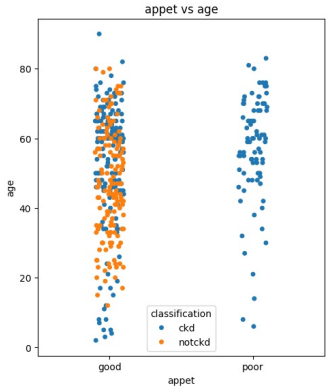

cad vs pot
classification
ckd
notckd
pot
cad
no
yes

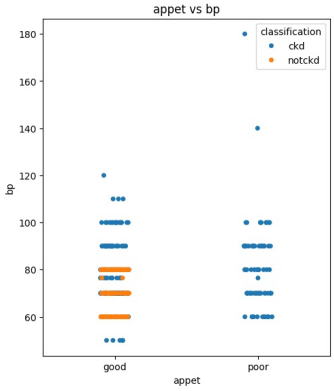

cad vs hemo
classification
ckd
notckd
hemo
cad
no
yes

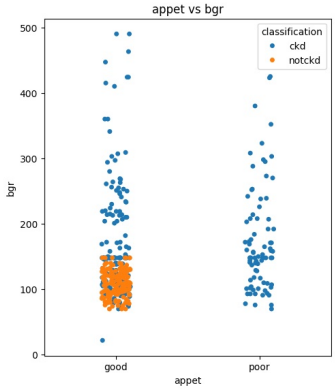

cad vs pcv
classification
ckd
notckd
pcv
cad
no
yes

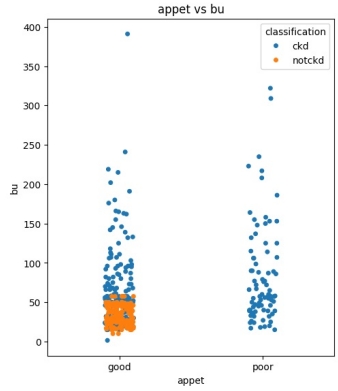

cad vs rc
classification
ckd
notckd
rc
cad
no
yes

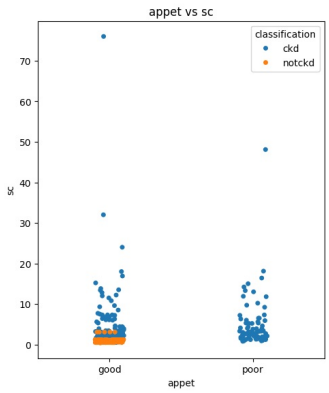

cad vs wc
classification
ckd
notckd
wc
cad
no
yes

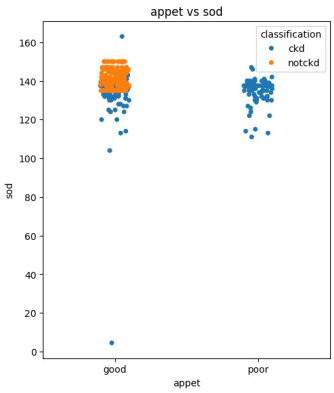

appet vs age
age
appet
good
poor
classification
ckd
notckd

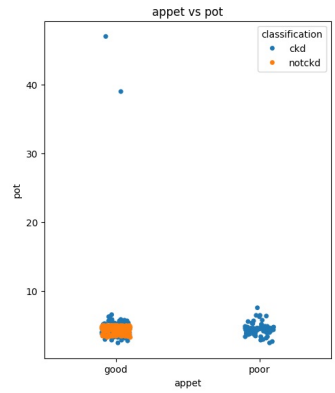

appet vs bp
bp
appet
good
poor
classification
ckd
notckd

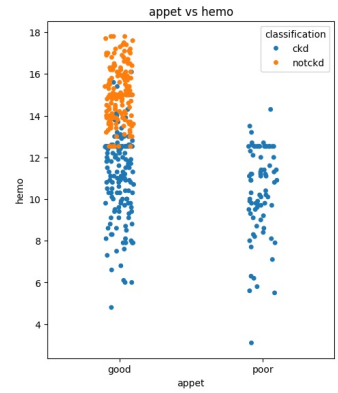

appet vs bgr
bgr
appet
good
poor
classification
ckd
notckd

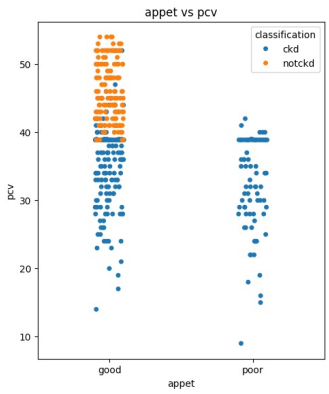

appet vs bu
bu
appet
good
poor
classification
ckd
notckd

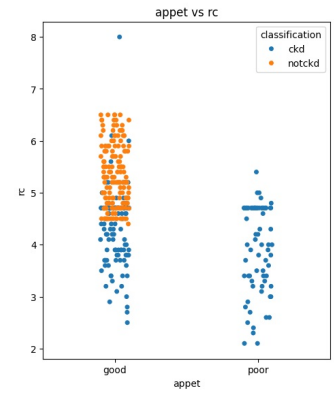

appet vs sc
sc
appet
good
poor
classification
ckd
notckd

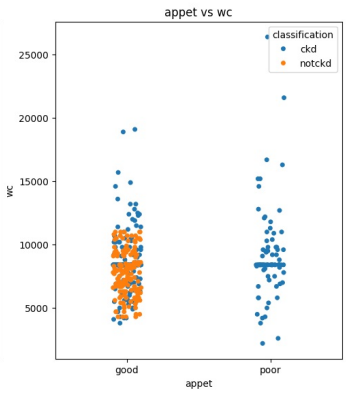

appet vs sod
sod
appet
good
poor
classification
ckd
notckd

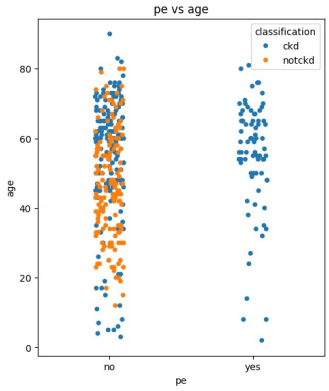

appet vs pot
pot
appet
good
poor
classification
ckd
notckd

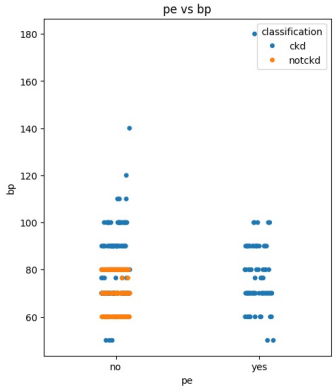

appet vs hemo
hemo
appet
good
poor
classification
ckd
notckd

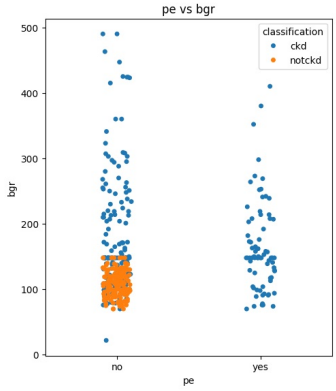

appet vs pcv
pcv
appet
good
poor
classification
ckd
notckd

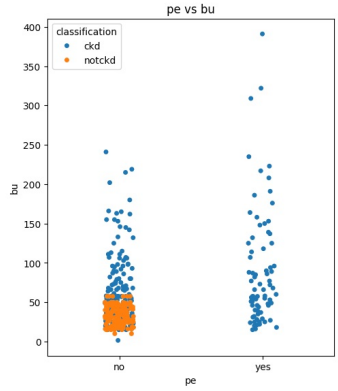

appet vs rc
rc
appet
good
poor
classification
ckd
notckd

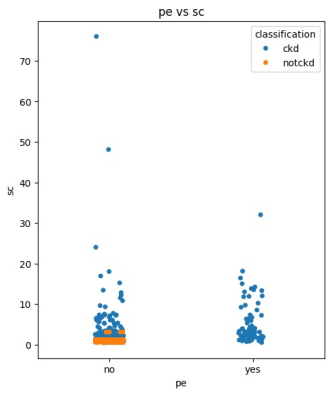

appet vs wc
wc
appet
good
poor
classification
ckd
notckd

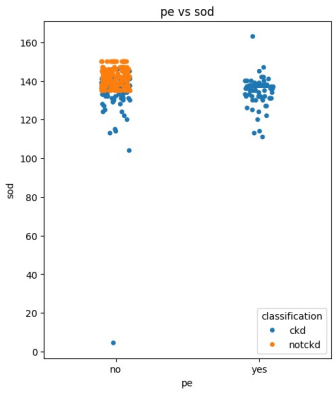

pe vs age
classification
ckd
notckd
age
pe
no
yes

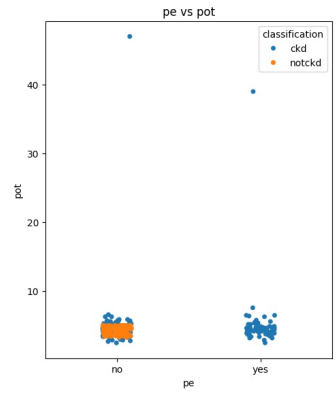

pe vs bp
classification
ckd
notckd
bp
pe
no
yes

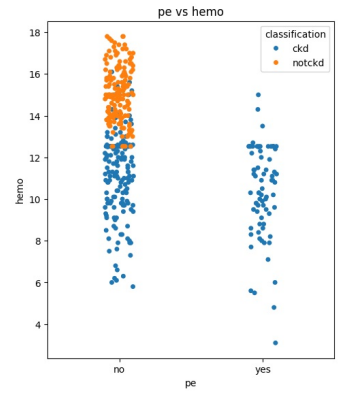

pe vs bgr
classification
ckd
notckd
bgr
pe
no
yes

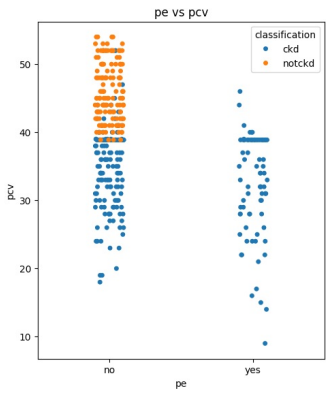

pe vs bu
classification
ckd
notckd
bu
pe
no
yes

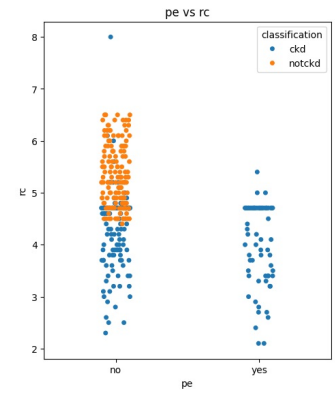

pe vs sc
classification
ckd
notckd
sc
pe
no
yes

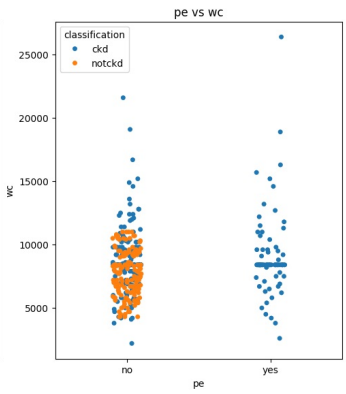

pe vs sod
classification
ckd
notckd
sod
pe
no
yes

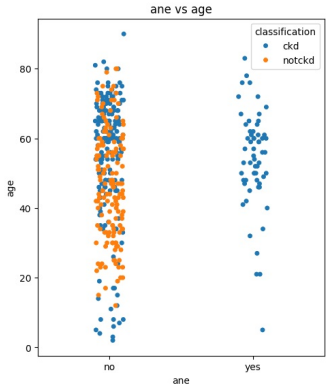

pe vs pot
classification
ckd
notckd
pot
pe
no
yes

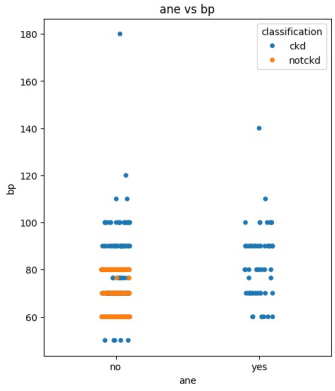

pe vs hemo
classification
ckd
notckd
hemo
pe
no
yes

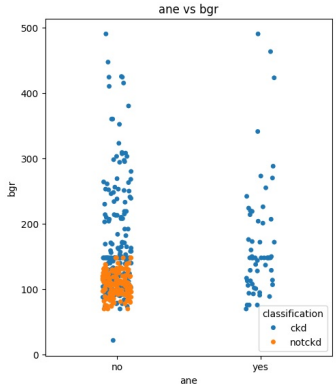

pe vs pcv
classification
ckd
notckd
pcv
pe
no
yes

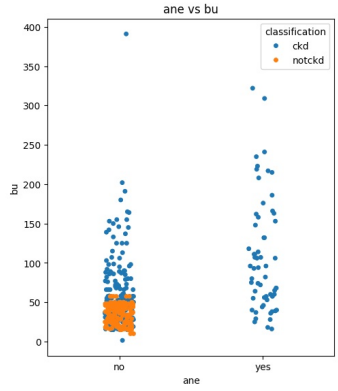

pe vs rc
classification
ckd
notckd
rc
pe
no
yes

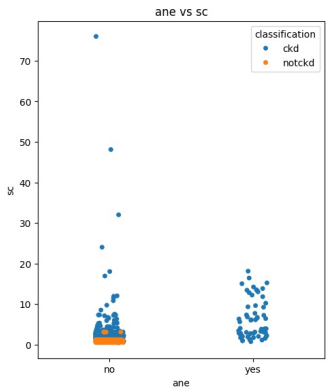

pe vs wc
classification
ckd
notckd
wc
pe
no
yes

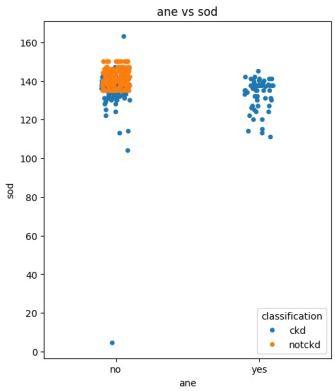

ane vs age
classification
ckd
notckd
age
ane
no
yes

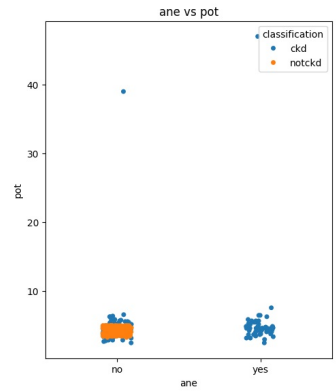

ane vs bp
classification
ckd
notckd
bp
ane
no
yes

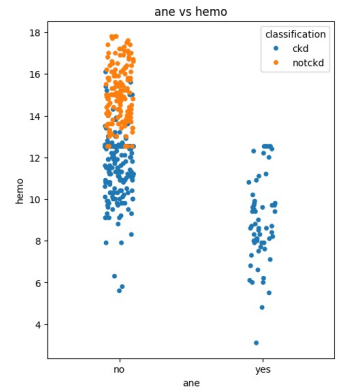

ane vs bgr
classification
ckd
notckd
bgr
ane
no
yes

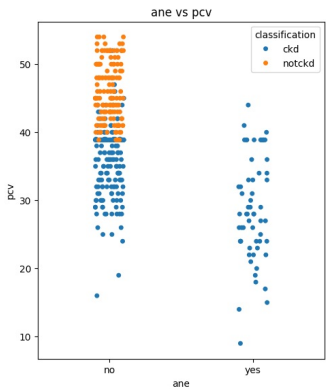

ane vs bu
classification
ckd
notckd
bu
ane
no
yes

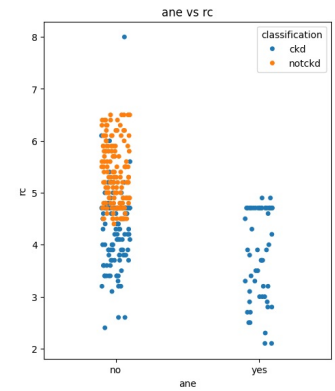

ane vs sc
classification
ckd
notckd
sc
ane
no
yes

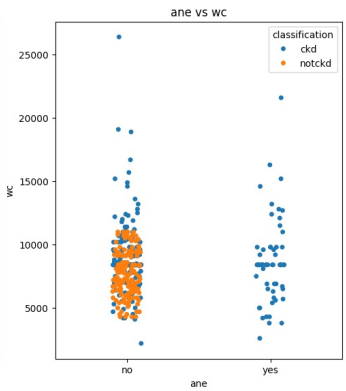

ane vs sod
classification
ckd
notckd
sod
ane
no
yes

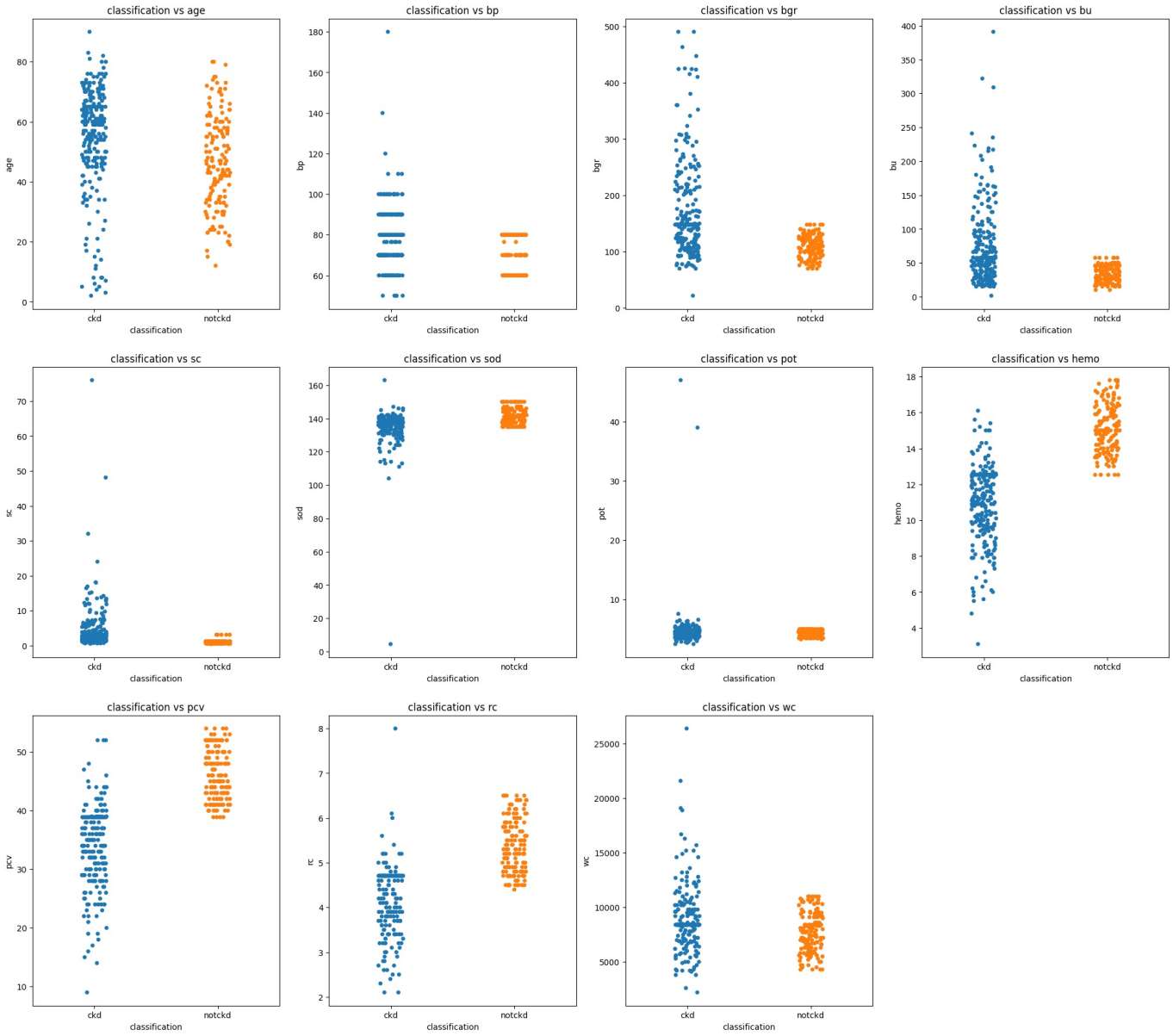

ane vs pot
classification
ckd
notckd
pot
ane
no
yes


ane vs hemo
classification
ckd
notckd
hemo
ane
no
yes


ane vs pcv
classification
ckd
notckd
pcv
ane
no
yes


ane vs rc
classification
ckd
notckd
rc
ane
no
yes


ane vs wc
classification
ckd
notckd
wc
ane
no
yes



Observations: The correlation coefficient of some *categorical* features with *continuous* features is very low

Note: Multivariate Analysis of dataset using heatmap is carried out after Label encoding(Step 7) since correlation demands all features to be of float type

Step: 7 Label Encoding of Categorical features

Encoding Categorical values into numerical values using `LabelEncoder`

In [ ]:
```
label_enc=LabelEncoder()
classes={}
for i in categorical.iloc[:]['Categorical Columns']:
    kidney_data[i] = label_enc.fit_transform(kidney_data[i])
    classes[f"{i}"]=label_enc.classes_
kidney_data.head()
```

Out[ ]:

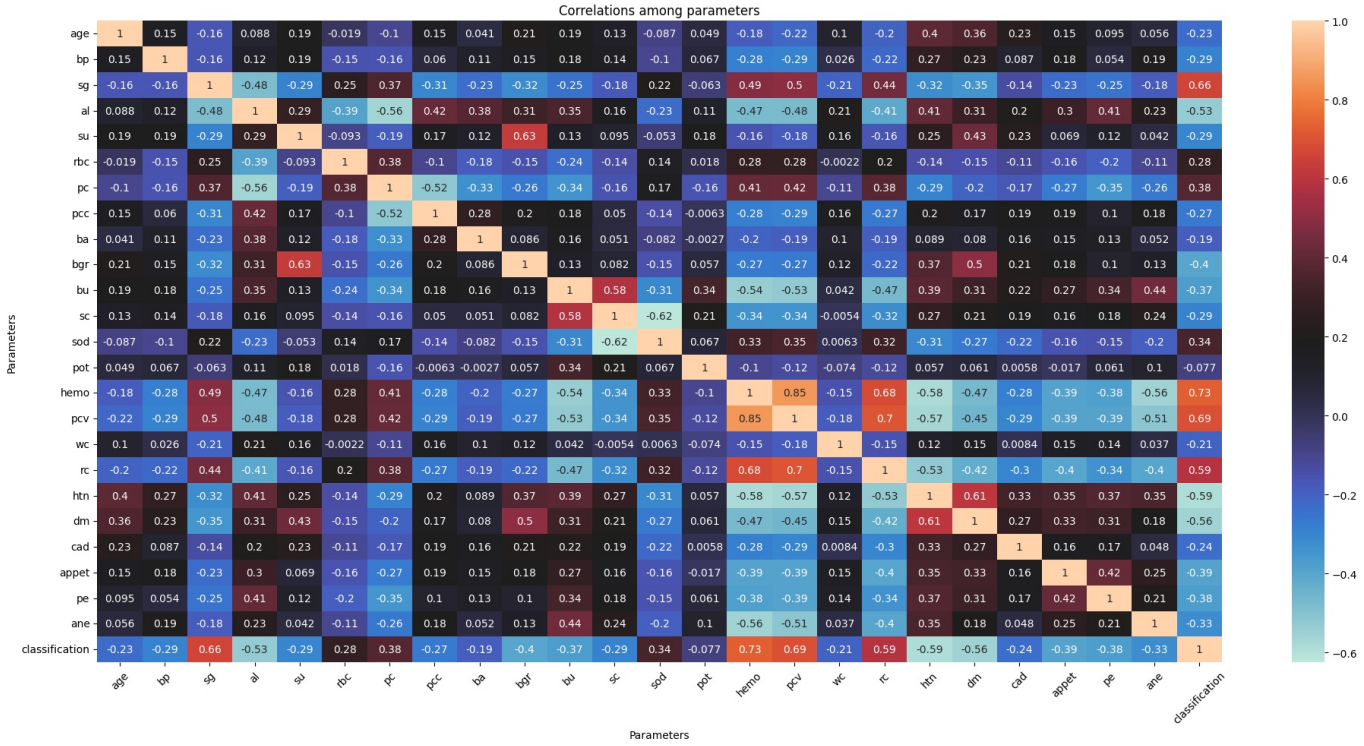
	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
0	48.0	80.0	3	1	0	1	1	0	0	121.000000	...	44.0	7800.0	5.200000	1	1	0	0	0	0	0
1	7.0	50.0	3	4	0	1	1	0	0	148.036517	...	38.0	6000.0	4.707435	0	0	0	0	0	0	0
2	62.0	80.0	1	2	3	1	1	0	0	423.000000	...	31.0	7500.0	4.707435	0	1	0	1	0	1	0
3	48.0	70.0	0	4	0	1	0	1	0	117.000000	...	32.0	6700.0	3.900000	1	0	0	1	1	1	0
4	51.0	80.0	1	2	0	1	1	0	0	106.000000	...	35.0	7300.0	4.600000	0	0	0	0	0	0	0

5 rows × 25 columns

Multivariate Analysis

In [ ]:
```
plt.figure(figsize=(20,10))
correlations=kidney_data.corr()
sns.heatmap(correlations,annot=True,cmap='icefire')
```

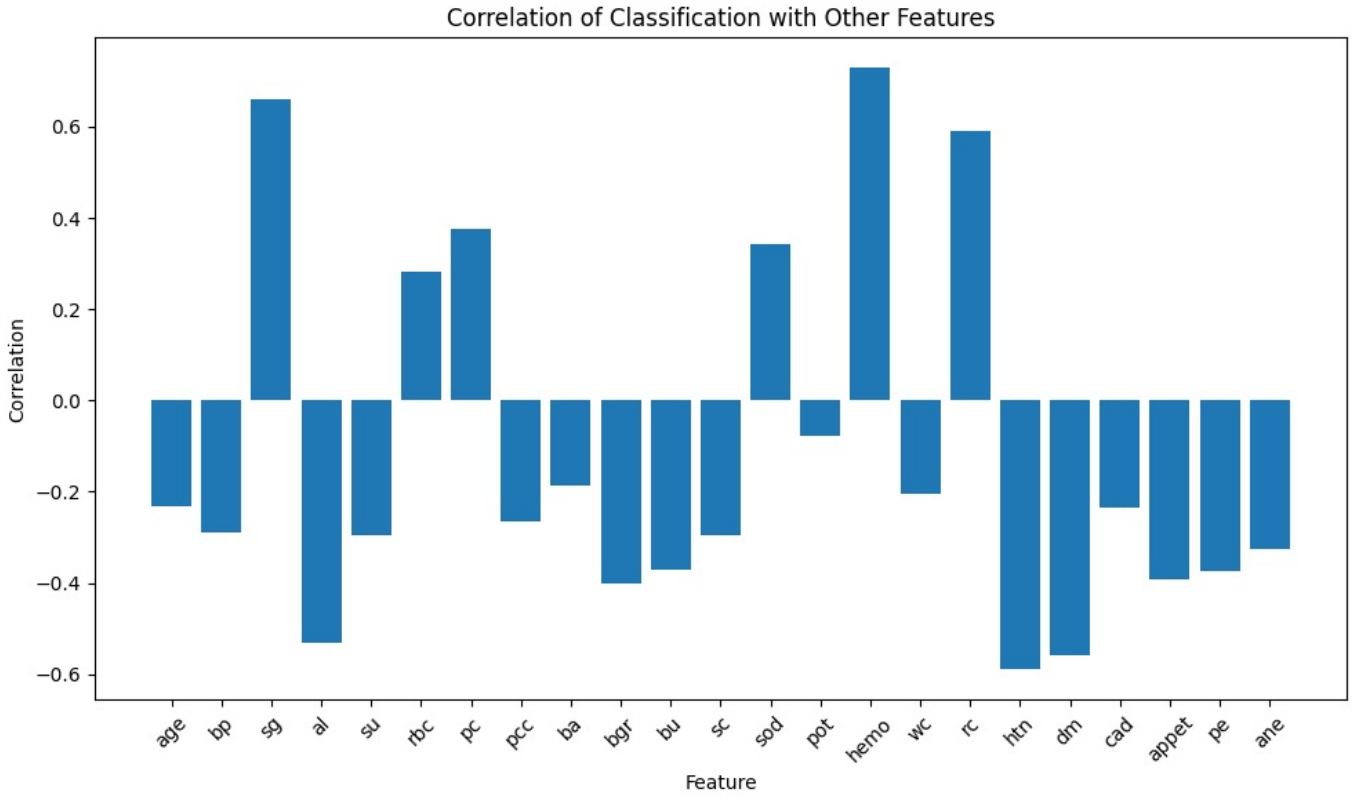
```
plt.xlabel("Parameters")
plt.ylabel("Parameters")
plt.title("Correlations among parameters")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Observations: From the heatmap, it is observed that `pcv` and `hemo` have 85% multicollinearity. Hence, we will drop `pcv` from the dataset

In [ ]:
```
kidney_data.drop(columns = ['pcv'],inplace=True)
```

In [ ]:
```
correlation_dict = {}
for col in kidney_data.drop(columns=['classification']).columns:
  correlation_dict[col] = kidney_data['classification'].corr(kidney_data[col])
correlation_values = list(correlation_dict.values())
column_names = list(correlation_dict.keys())
plt.figure(figsize=(10, 6))
plt.bar(column_names, correlation_values)
plt.xlabel('Feature')
plt.ylabel('Correlation')
plt.title('Correlation of Classification with Other Features')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



In [ ]:
```
kidney_data.to_csv('kidney_data_processed.csv')
```

Now that all the values are of numeric dtype, we will create a Synthethic dataset from the real dataset using `copulas` library to reduce overfitting to a certain extent

In [ ]:
```
copula = GaussianMultivariate()
copula.fit(kidney_data)
synthetic_data = copula.sample(200)
```

Altering value types in `synthetic_data` to match `kidney_data`

In [ ]:
```
synthetic_data.age = round(abs(synthetic_data.age)).astype('float64')
synthetic_data.bp =round(abs(synthetic_data.bp)).astype('float64')
synthetic_data.sg =round(abs(synthetic_data.sg)).astype('int64')
synthetic_data.su =round(abs(synthetic_data.su)).astype('int64')
synthetic_data.al =round(abs(synthetic_data.al)).astype('int64')
synthetic_data.pcc =round(abs(synthetic_data.pcc)).astype('int64')
synthetic_data.rbc =round(abs(synthetic_data.rbc)).astype('int64')
synthetic_data.pc =round(abs(synthetic_data.pc)).astype('int64')
synthetic_data.ba =round(abs(synthetic_data.ba)).astype('int64')
synthetic_data.bgr =round(abs(synthetic_data.bgr)).astype('float64')
synthetic_data.bu =round(abs(synthetic_data.bu)).astype('float64')
synthetic_data.sc =round(abs(synthetic_data.sc)).astype('float64')
synthetic_data.sod =round(abs(synthetic_data.sod)).astype('float64')
synthetic_data.pot =round(abs(synthetic_data.pot)).astype('float64')
synthetic_data.hemo =round(abs(synthetic_data.hemo)).astype('float64')
synthetic_data.wc = round(abs(synthetic_data.wc)).astype('float64')
synthetic_data.rc =round(abs(synthetic_data.rc)).astype('float64')
synthetic_data.htn =round(abs(synthetic_data.htn)).astype('int64')
synthetic_data.dm =round(abs(synthetic_data.dm)).astype('int64')
synthetic_data.cad =round(abs(synthetic_data.cad)).astype('int64')
synthetic_data.appet =round(abs(synthetic_data.appet)).astype('int64')
synthetic_data.pe =round(abs(synthetic_data.pe)).astype('int64')
synthetic_data.ane =round(abs(synthetic_data.ane)).astype('int64')
synthetic_data.classification =round(abs(synthetic_data.classification)).astype('int64')
```

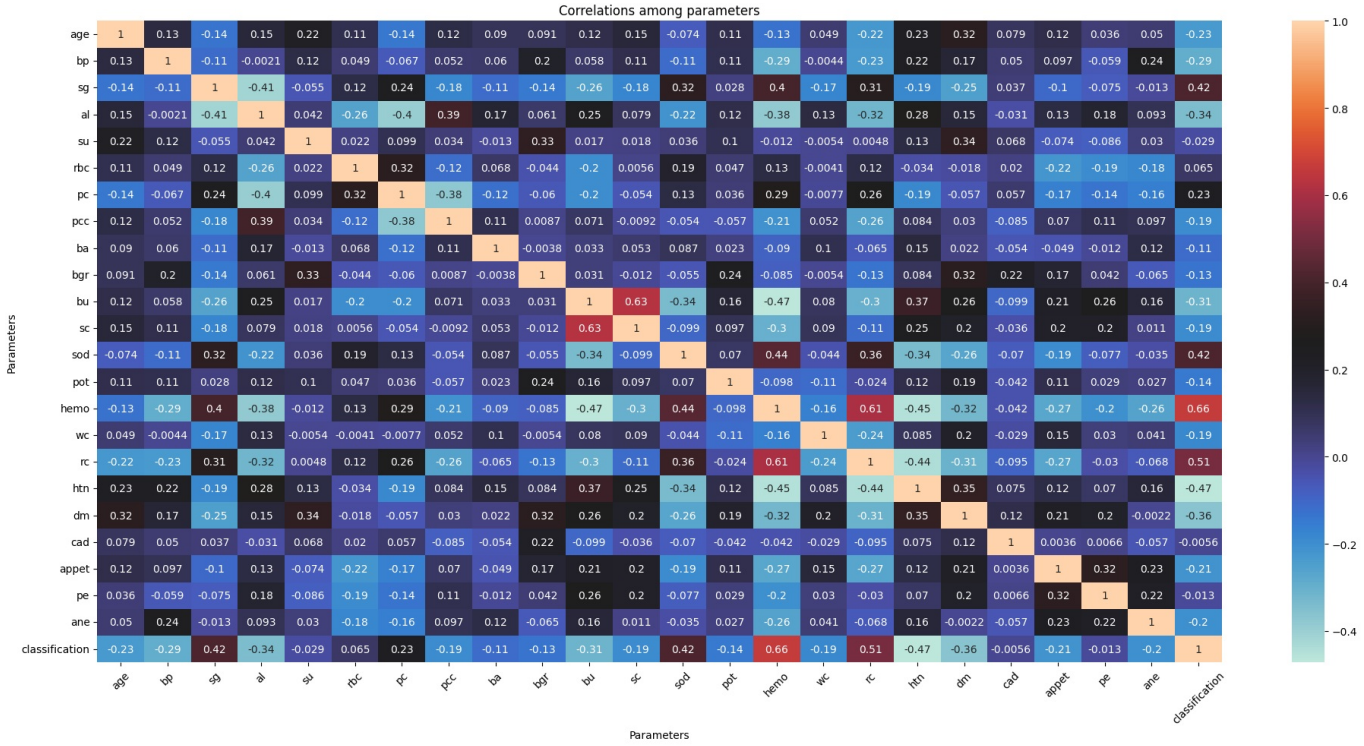
In [ ]:
```
synthetic_data.to_csv("synthetic_data.csv",index=False) #Saving synthetic data once a good quality synthetic da
```

In [ ]:
```
#synthetic_data = pd.read_csv('synthetic_data.csv')  #Reading only when good quality synthetic data is generate
```

Heat Map of Synthetic Dataset

In [ ]:
```
plt.figure(figsize=(20,10))
correlations=synthetic_data.corr()
sns.heatmap(correlations,annot=True,cmap='icefire')
plt.xlabel("Parameters")
plt.ylabel("Parameters")
```

```
plt.title("Correlations among parameters")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Step: 8 Splitting the Dataset into Independent and Dependent Variable

X - Independant Variables

Y - Target/Dependent Variable

In [ ]:
```
columns = list(kidney_data.columns)
columns.remove('classification')
```

Splitting Training and Testing dataset

`merged_kidney_data` will be used for training the model using cross-validation technique and `test_data` will be the unseen data tested upon to assess the genralizing ability of the model.

In [ ]:
```
merged_kidney_data = pd.concat([kidney_data,synthetic_data])
test_size = int(0.2*(merged_kidney_data.shape[0]))
test_data = merged_kidney_data.sample(test_size,random_state=42)
merged_kidney_data.drop(test_data.index,inplace=True)
```

In [ ]:
```
Xtest = test_data.drop("classification",axis=1)
ytest = test_data['classification']
```

Handling imbalance issue of `merged_kidney_data` using SMOTE

In [ ]:
```
X = merged_kidney_data.drop("classification", axis=1)
y = merged_kidney_data["classification"]
smote = SMOTE(random_state=42)
X_resampled,y_resampled = smote.fit_resample(X, y)
```

In [ ]:
```
print(y.value_counts())
print(y_resampled.value_counts())
```

```
classification
0    239
1    179
Name: count, dtype: int64
classification
0    239
1    239
Name: count, dtype: int64
```

In [ ]:
```
print(X_resampled.shape)
print(y_resampled.shape)
```

```
(478, 23)
(478,)
```

In [ ]:
```
kf=KFold(n_splits=25,random_state=42,shuffle=True) #KFold object
Accuracy,All_prec,All_rec,test_accuracy,test_recall,test_precision = {},{},{},{},{},{} #Dictionaries to store c
```

STEP: 9 Model Developemnt

Model: Random Forest Classification

Initial Model

In [ ]:
```
rfc = RandomForestClassifier()
```

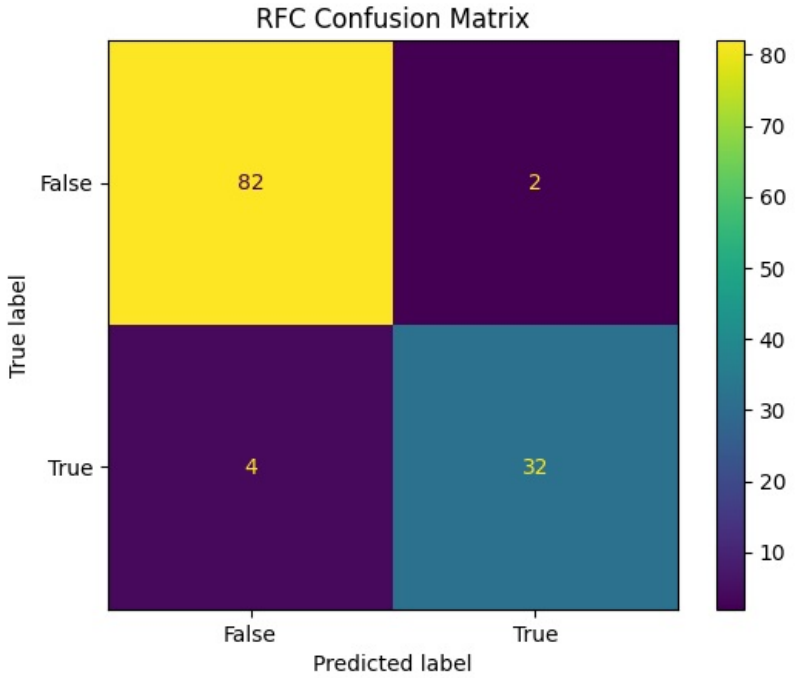
In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,train_size=0.8,random_state=42)
rfc.fit(X_train,y_train)
y_pred= rfc.predict(X_test)
print(f'Accuracy:{accuracy_score(y_test,y_pred)*100:.4f}%')
```

```
Accuracy:91.6667%
```

In [ ]:
```
y_pred = rfc.predict(Xtest)
print(f'Unseen Data Accuracy:{accuracy_score(ytest,y_pred)*100:.4f}%')
print(f'Unseen Data recall:{recall_score(ytest,y_pred)*100:.4f}%')
```

```
Unseen Data Accuracy:95.0000%
Unseen Data recall:88.8889%
```

In [ ]:
```
confusion_mat = confusion_matrix(ytest,y_pred)
cm_display = ConfusionMatrixDisplay(confusion_matrix = confusion_mat, display_labels = ['False', 'True'])
cm_display.plot()
plt.title("RFC Confusion Matrix")
plt.show()
```



In [ ]:
```
print(classification_report(ytest,y_pred))
```

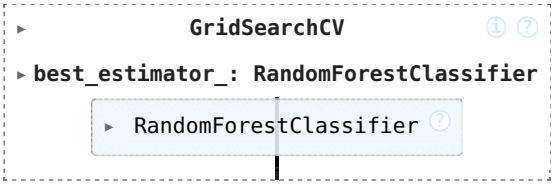
```
              precision    recall  f1-score   support

           0       0.95      0.98      0.96        84
           1       0.94      0.89      0.91        36

    accuracy                           0.95       120
   macro avg       0.95      0.93      0.94       120
weighted avg       0.95      0.95      0.95       120
```

Hyperparameter tuning using *GridSearchCV* (estimated run-time: 30-50 mins)

In [ ]:
```
param_dist = {
    'n_estimators': [10, 50, 100, 1000],
    'max_depth':[3, 5, 10, 15, 20, None],
    'min_samples_split': [2, 4, 8, 16],
    'min_samples_leaf': [1, 2, 4, 8],
    'max_features': ['auto', 'sqrt', 'log2']
}
```

In [ ]:
```
grid_search = GridSearchCV(estimator=rfc,param_grid=param_dist,scoring="accuracy",cv=10)
grid_search.fit(X_train, y_train)
```

Out[ ]:



Getting the best parameters for the model

In [ ]:
```
best_params = grid_search.best_params_
max_depth = best_params['max_depth']
n_estimators = best_params['n_estimators']
min_sample_split = best_params['min_samples_split']
min_sample_leaf = best_params['min_samples_leaf']
best_params
```

Out[ ]:
```
{'max_depth': 5,
 'max_features': 'log2',
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 50}
```

In [ ]:
```
best_rfc=RandomForestClassifier(n_estimators=n_estimators,min_samples_split=min_sample_split, min_samples_leaf=
rfe_rfc = RFECV(best_rfc,min_features_to_select=20,cv=kf,step=1)
```

In [ ]:
```
scores=[]
precision= []
recall = []
for train_index, test_index in kf.split(X_resampled,y_resampled):
  X_train, X_test = X_resampled.iloc[train_index], X_resampled.iloc[test_index]
  y_train, y_test = y_resampled.iloc[train_index], y_resampled.iloc[test_index]
  rfe_rfc.fit(X_train, y_train)
  y_pred=rfe_rfc.predict(X_test)
  prec = precision_score(y_test,y_pred)
  rec = recall_score(y_test,y_pred)
  score = accuracy_score(y_test,y_pred)
  precision.append(prec)
  recall.append(rec)
  scores.append(score)
  print(f'Fold Score: {score}')
```

```
Fold Score: 0.95
Fold Score: 0.95
Fold Score: 0.8
Fold Score: 0.9473684210526315
Fold Score: 0.8947368421052632
Fold Score: 1.0
Fold Score: 0.8947368421052632
Fold Score: 1.0
Fold Score: 0.8947368421052632
Fold Score: 1.0
Fold Score: 1.0
Fold Score: 1.0
Fold Score: 0.9473684210526315
Fold Score: 0.8947368421052632
Fold Score: 0.7894736842105263
Fold Score: 0.8947368421052632
Fold Score: 0.9473684210526315
Fold Score: 1.0
Fold Score: 1.0
Fold Score: 0.9473684210526315
Fold Score: 0.9473684210526315
Fold Score: 0.9473684210526315
Fold Score: 0.9473684210526315
Fold Score: 0.8947368421052632
Fold Score: 1.0
```

In [ ]:
```
average_score = sum(scores) / len(scores)
average_precision = sum(precision)/len(precision)
average_recall = sum(recall)/len(recall)
Accuracy['Random Forest'] = average_score
All_prec['Random Forest'] = average_precision
All_rec['Random Forest'] = average_recall
print(f'Average Cross-Validation Score: {average_score*100:.4f}')
print(f'Average Precision score: {average_precision*100:.4f}')
print(f'Average Recall score: {average_recall*100:.4f}')
```

```
Average Cross-Validation Score: 93.9579
Average Precision score: 94.5841
Average Recall score: 93.5202
```

In [ ]:
```
y_pred = rfe_rfc.predict([[45,90,1,0,1,1,0,0,0,230,57,0,140,6,20,3700,0,0,0,1,0,0,0]])
```

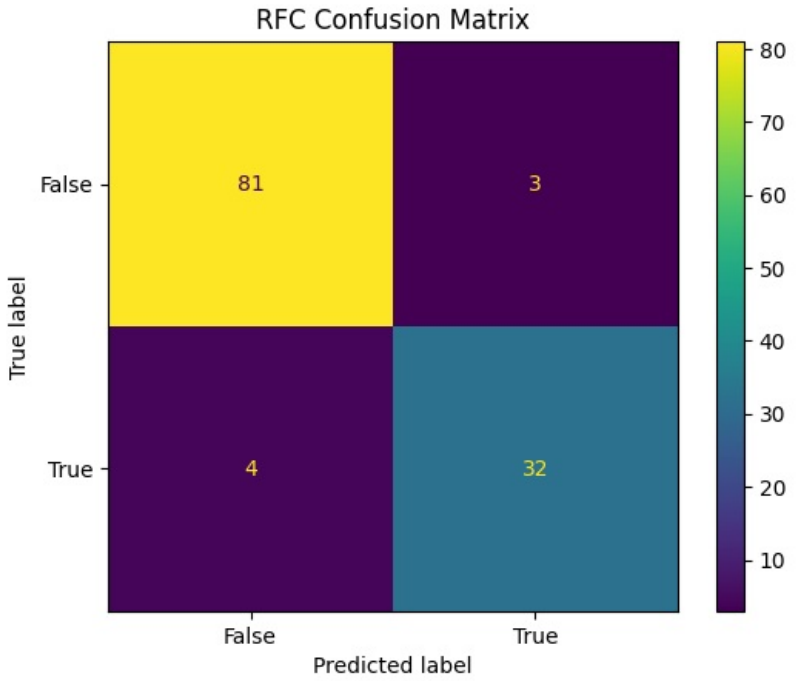
```
print(y_pred)
```

```
[0]
```

Testing optimized model on unseen data

In [ ]:
```
y_pred=rfe_rfc.predict(Xtest)
test_accuracy['Random Forest'] = accuracy_score(ytest,y_pred)
test_recall['Random Forest'] = recall_score(ytest,y_pred)
test_precision['Random Forest'] = precision_score(ytest,y_pred)
```

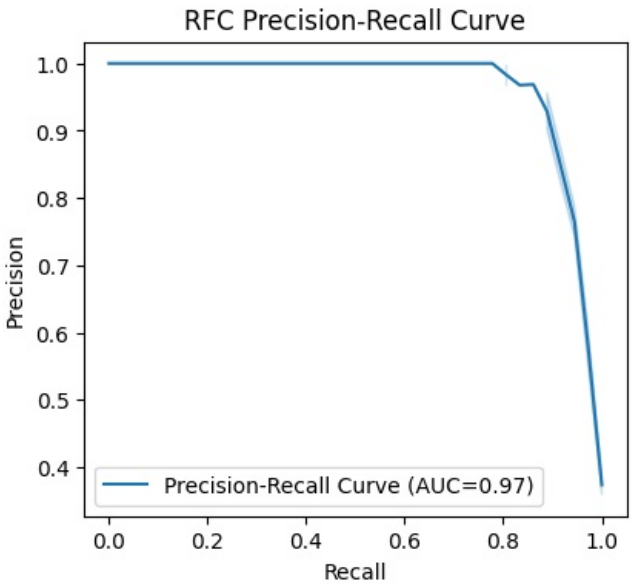
In [ ]:
```
confusion_mat=confusion_matrix(ytest,y_pred)
cm_display = ConfusionMatrixDisplay(confusion_matrix = confusion_mat, display_labels = ['False', 'True'])
cm_display.plot()
plt.title('RFC Confusion Matrix')
plt.show()
```

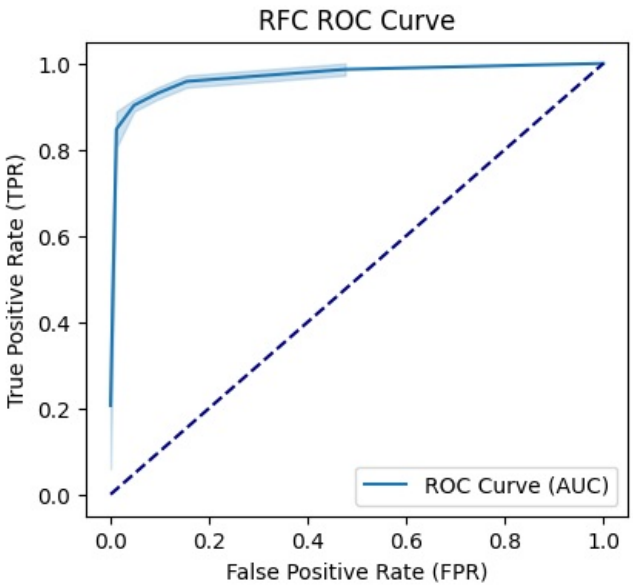


In [ ]:
```
print(classification_report(ytest,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.95      0.96      0.96        84
           1       0.91      0.89      0.90        36

    accuracy                           0.94       120
   macro avg       0.93      0.93      0.93       120
weighted avg       0.94      0.94      0.94       120
```

In [ ]:
```
yscore = rfe_rfc.predict_proba(Xtest)[:,1]
plt.figure(figsize=(10,4))
precision, recall, thresholds = precision_recall_curve(ytest, yscore)
auc_score=auc(recall,precision)
plt.subplot(1,2,1)
sns.lineplot(x=recall, y=precision, label=f'Precision-Recall Curve (AUC={auc_score:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('RFC Precision-Recall Curve')
fpr, tpr, thresholds = roc_curve(ytest, yscore)
plt.subplot(1,2,2)
sns.lineplot(x=fpr, y=tpr, label='ROC Curve (AUC)')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('RFC ROC Curve')
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.show()
```





Model: Logistic Regression

Initial Model

In [ ]:
```
lr = LogisticRegression()
```

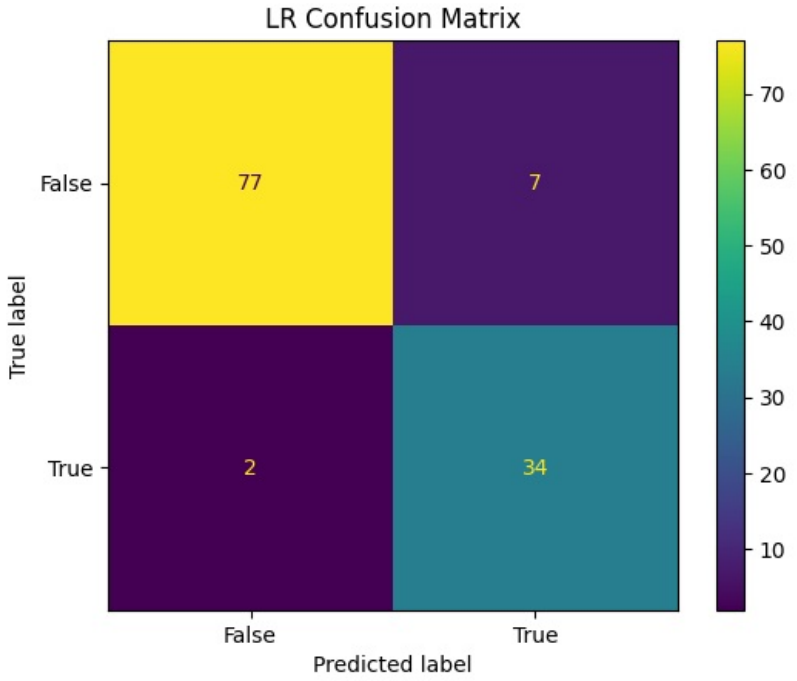
In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,test_size=0.2,random_state=42)
lr.fit(X_train,y_train)
y_pred= lr.predict(X_test)
print(f'Accuracy:{accuracy_score(y_test,y_pred)*100:.4f}%')
```

```
Accuracy:89.5833%
```

In [ ]:
```
y_pred = lr.predict(Xtest)
print(f'Unseen Data Accuracy:{accuracy_score(ytest,y_pred)*100:.4f}%')
print(f'Unseen Data Recall:{recall_score(ytest,y_pred)*100:.4f}%')
```

```
Unseen Data Accuracy:92.5000%
Unseen Data Recall:94.4444%
```

In [ ]:
```
confusion_mat = confusion_matrix(ytest,y_pred)
cm_display = ConfusionMatrixDisplay(confusion_matrix = confusion_mat, display_labels = ['False', 'True'])
cm_display.plot()
plt.title("LR Confusion Matrix")
plt.show()
```



In [ ]:
```
print(classification_report(ytest,y_pred))
```

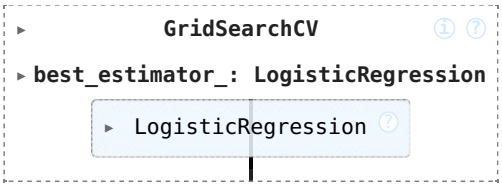
```
              precision    recall  f1-score   support

           0       0.97      0.92      0.94        84
           1       0.83      0.94      0.88        36

    accuracy                           0.93       120
   macro avg       0.90      0.93      0.91       120
weighted avg       0.93      0.93      0.93       120
```

Hyperparameter tuning using *GridSearchCV* (estimated run-time: 30 secs)

In [ ]:
```
param_dist = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'solver': ['liblinear', 'lbfgs']
}
```

In [ ]:
```
grid_search = GridSearchCV(estimator=lr,param_grid=param_dist,scoring='accuracy',cv=10)
grid_search.fit(X_train,y_train)
```

Out[ ]:



In [ ]:
```
best_params = grid_search.best_params_
C = best_params['C']
Solver = best_params['solver']
best_lr = LogisticRegression(C=C,solver=Solver)
rfe_lr = RFECV(best_lr,min_features_to_select=20,cv=kf,step=1)
```

In [ ]:
```
scores=[]
precision =[]
recall = []
for train_index, test_index in kf.split(X_resampled,y_resampled):
  X_train, X_test = X_resampled.iloc[train_index], X_resampled.iloc[test_index]
  y_train, y_test = y_resampled.iloc[train_index], y_resampled.iloc[test_index]
  rfe_lr.fit(X_train, y_train)
  y_pred=rfe_lr.predict(X_test)
  prec = precision_score(y_test,y_pred)
  rec = recall_score(y_test,y_pred)
  score = rfe_lr.score(X_test, y_test)
  scores.append(score)
  precision.append(prec)
  recall.append(rec)
  print(f'Fold Score: {score}')
```

```
Fold Score: 0.9
Fold Score: 0.8
Fold Score: 0.85
Fold Score: 0.9473684210526315
Fold Score: 0.8947368421052632
Fold Score: 1.0
Fold Score: 0.9473684210526315
Fold Score: 1.0
Fold Score: 0.9473684210526315
Fold Score: 0.9473684210526315
Fold Score: 1.0
Fold Score: 1.0
Fold Score: 0.8947368421052632
Fold Score: 0.8947368421052632
Fold Score: 0.8421052631578947
Fold Score: 0.8947368421052632
Fold Score: 0.9473684210526315
Fold Score: 0.9473684210526315
Fold Score: 0.9473684210526315
Fold Score: 1.0
Fold Score: 0.9473684210526315
Fold Score: 0.9473684210526315
Fold Score: 0.8947368421052632
Fold Score: 0.9473684210526315
Fold Score: 0.8947368421052632
```

In [ ]:
```
average_score = sum(scores) / len(scores)
average_precision = sum(precision)/len(precision)
average_recall = sum(recall)/len(recall)
Accuracy['Linear Regression'] = average_score
All_prec['Linear Regression'] = average_precision
All_rec['Linear Regression '] = average_recall
print(f'Average Cross-Validation Score: {average_score*100:.4f}')
```

```
print(f'Average Precision score: {average_precision*100:.4f}')
print(f'Average Recall score: {average_recall*100:.4f}')
```

```
Average Cross-Validation Score: 92.9368
Average Precision score: 91.9162
Average Recall score: 94.6172
```

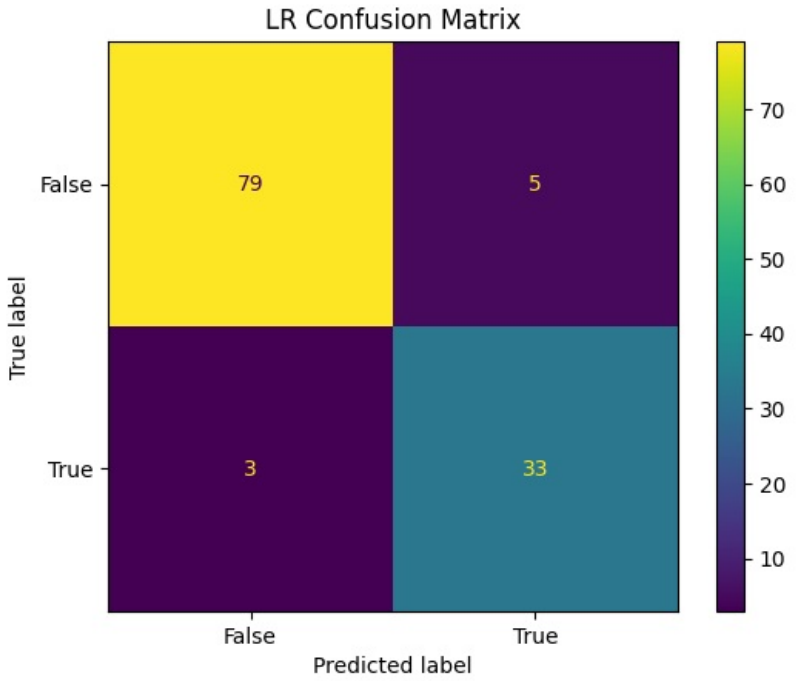
In [ ]:
```
y_pred = rfe_lr.predict([[45,90,1,0,1,1,0,0,0,230,57,0,140,6,20,3700,0,0,0,1,0,0,0]])
print(y_pred)
```

```
[1]
```

Testing optimized model on unseen data

In [ ]:
```
y_pred= rfe_lr.predict(Xtest)
test_accuracy['Logistic Regression'] = accuracy_score(ytest,y_pred)
test_recall['Logistic Regression'] = recall_score(ytest,y_pred)
test_precision['Logistic Regression'] = precision_score(ytest,y_pred)
```

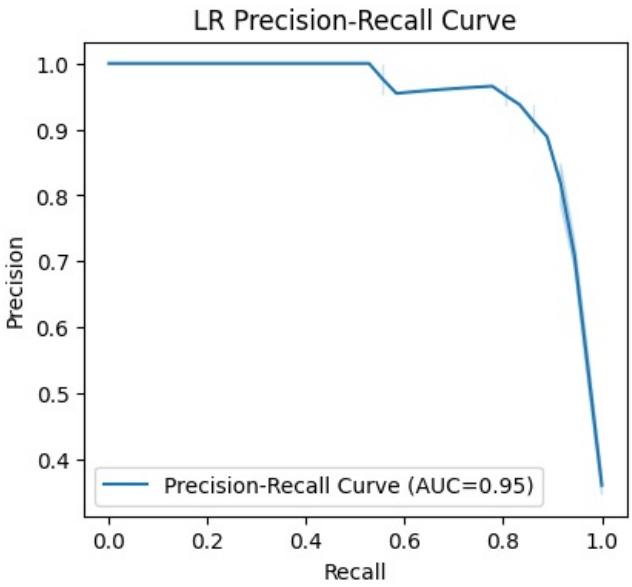
In [ ]:
```
confusion_mat=confusion_matrix(y_pred=y_pred,y_true=ytest)
cm_display = ConfusionMatrixDisplay(confusion_matrix = confusion_mat, display_labels = ['False', 'True'])
cm_display.plot()
plt.title('LR Confusion Matrix')
plt.show()
```

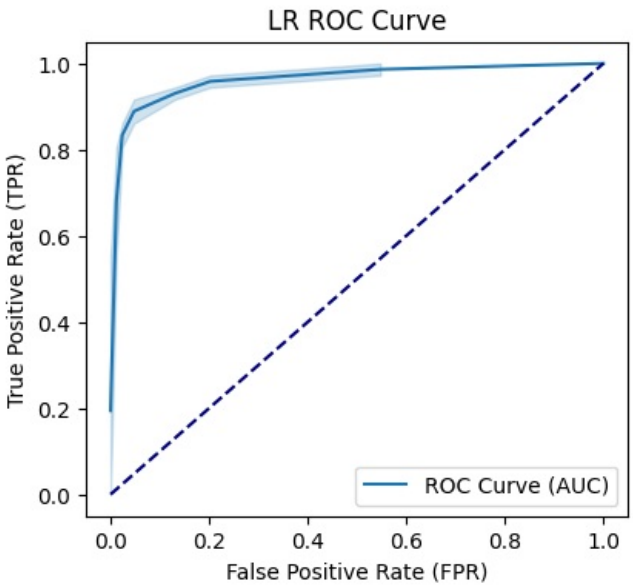


In [ ]:
```
print(classification_report(y_pred=y_pred,y_true=ytest))
```

```
              precision    recall  f1-score   support

           0       0.96      0.94      0.95        84
           1       0.87      0.92      0.89        36

    accuracy                           0.93       120
   macro avg       0.92      0.93      0.92       120
weighted avg       0.93      0.93      0.93       120
```

In [ ]:
```
yscore = rfe_lr.predict_proba(Xtest)[:,1]
plt.figure(figsize=(10,4))
precision, recall, thresholds = precision_recall_curve(ytest, yscore)
auc_score = auc(recall,precision)
plt.subplot(1,2,1)
sns.lineplot(x=recall, y=precision, label=f'Precision-Recall Curve (AUC={auc_score:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('LR Precision-Recall Curve')
fpr, tpr, thresholds = roc_curve(ytest, yscore)
plt.subplot(1,2,2)
sns.lineplot(x=fpr, y=tpr, label='ROC Curve (AUC)')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('LR ROC Curve')
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.show()
```





Model: Decision Tree Classifier

Initial Model

In [ ]:
```
dt= DecisionTreeClassifier()
```

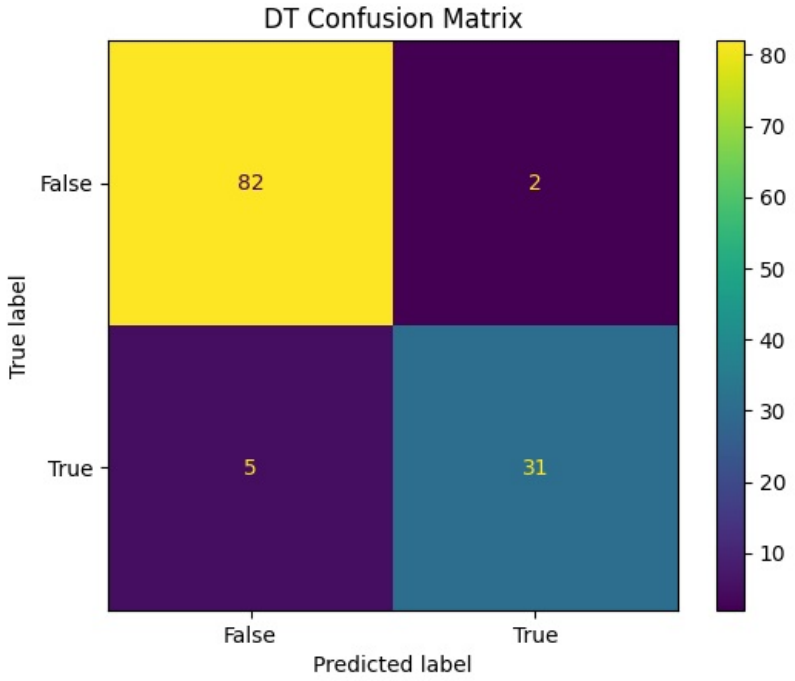
In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,test_size=0.2,random_state=42)
dt.fit(X_train,y_train)
y_pred= dt.predict(X_test)
print(f'Accuracy:{accuracy_score(y_test,y_pred)*100:.4f}%')
```

```
Accuracy:87.5000%
```

In [ ]:
```
y_pred = dt.predict(Xtest)
print(f'Unseen Data Accuracy:{accuracy_score(ytest,y_pred)*100:.4f}%')
print(f'Unseen Data Recall:{recall_score(ytest,y_pred)*100:.4f}%')
```

```
Unseen Data Accuracy:94.1667%
Unseen Data Recall:86.1111%
```

In [ ]:
```
confusion_mat = confusion_matrix(ytest,y_pred)
cm_display = ConfusionMatrixDisplay(confusion_matrix = confusion_mat, display_labels = ['False', 'True'])
cm_display.plot()
plt.title("DT Confusion Matrix")
plt.show()
```



In [ ]:
```
print(classification_report(ytest,y_pred))
```

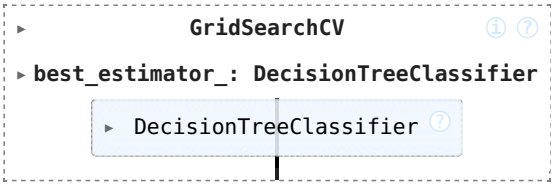
```
              precision    recall  f1-score   support

           0       0.94      0.98      0.96        84
           1       0.94      0.86      0.90        36

    accuracy                           0.94       120
   macro avg       0.94      0.92      0.93       120
weighted avg       0.94      0.94      0.94       120
```

Hyperparameter tuning using *GridSearchCV* (estimated run-time: 30 secs)

In [ ]:
```
param_dist = {
    'max_depth': [3, 5, 10, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2']
}
```

In [ ]:
```
grid_search = GridSearchCV(estimator=dt,param_grid=param_dist,scoring="accuracy",cv=10)
grid_search.fit(X_train,y_train)
```

Out[ ]:



In [ ]:
```
best_params = grid_search.best_params_
min_sample_split = best_params['min_samples_split']
min_sample_leaf = best_params['min_samples_leaf']
max_features = best_params['max_features']
max_depth=best_params['max_depth']
best_dt=DecisionTreeClassifier(min_samples_split=min_sample_split, min_samples_leaf=min_sample_leaf,max_features
rfe_dt = RFECV(best_dt,min_features_to_select=20,cv=kf,step=1)
```

In [ ]:
```
scores=[]
precision = []
recall=[]
for train_index, test_index in kf.split(X_resampled,y_resampled):
  X_train, X_test = X_resampled.iloc[train_index], X_resampled.iloc[test_index]
  y_train, y_test = y_resampled.iloc[train_index], y_resampled.iloc[test_index]
  rfe_dt.fit(X_train, y_train)
  y_pred=rfe_dt.predict(X_test)
  score = accuracy_score(y_test,y_pred)
  prec = precision_score(y_test,y_pred)
  rec =recall_score(y_test,y_pred)
  precision.append(prec)
  recall.append(rec)
  scores.append(score)
  print(f'Fold Score: {score}')
```

```
Fold Score: 0.9
Fold Score: 0.9
Fold Score: 0.8
Fold Score: 0.9473684210526315
Fold Score: 0.7894736842105263
Fold Score: 0.9473684210526315
Fold Score: 1.0
Fold Score: 0.9473684210526315
Fold Score: 0.8421052631578947
Fold Score: 0.8421052631578947
Fold Score: 1.0
Fold Score: 1.0
Fold Score: 0.8947368421052632
Fold Score: 1.0
Fold Score: 0.7894736842105263
Fold Score: 0.8947368421052632
Fold Score: 0.8947368421052632
Fold Score: 1.0
Fold Score: 1.0
Fold Score: 0.9473684210526315
Fold Score: 0.8947368421052632
Fold Score: 0.9473684210526315
Fold Score: 0.8947368421052632
Fold Score: 0.8421052631578947
Fold Score: 1.0
```

In [ ]:
```
average_score = sum(scores) / len(scores)
average_precision = sum(precision)/len(precision)
average_recall = sum(recall)/len(recall)
```

```
Accuracy['Decision Tree'] = average_score
All_prec['Decision Tree'] = average_precision
All_rec['Decision Tree'] = average_recall
print(f'Average Cross-Validation Score: {average_score*100:.4f}')
print(f'Average Precision score: {average_precision*100:.4f}')
print(f'Average Recall score: {average_recall*100:.4f}')
```

```
Average Cross-Validation Score: 91.6632
Average Precision score: 91.1676
Average Recall score: 91.7553
```

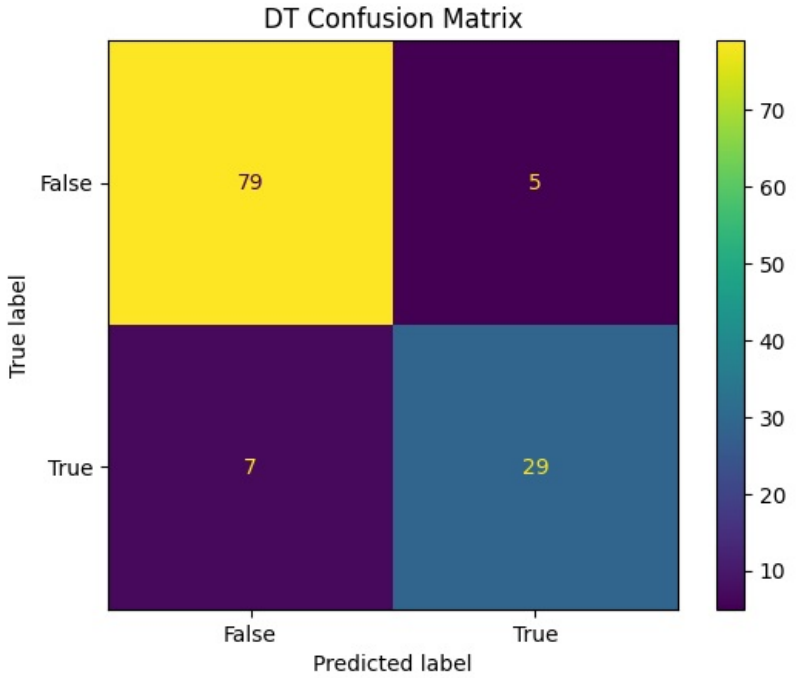
In [ ]:
```
y_pred = rfe_dt.predict([[45,90,1,0,1,1,0,0,0,230,57,0,140,6,20,3700,0,0,0,1,0,0,0]])
print(y_pred)
```

```
[0]
```

Testing optimized model on unseen data

In [ ]:
```
y_pred=rfe_dt.predict(Xtest)
test_accuracy['Decision Tree'] = accuracy_score(ytest,y_pred)
test_recall['Decision Tree'] = recall_score(ytest,y_pred)
test_precision['Decision Tree'] = precision_score(ytest,y_pred)
```

In [ ]:
```
confusion_mat=confusion_matrix(y_pred=y_pred,y_true=ytest)
cm_display = ConfusionMatrixDisplay(confusion_matrix = confusion_mat, display_labels = ['False', 'True'])
cm_display.plot()
plt.title('DT Confusion Matrix')
plt.show()
```

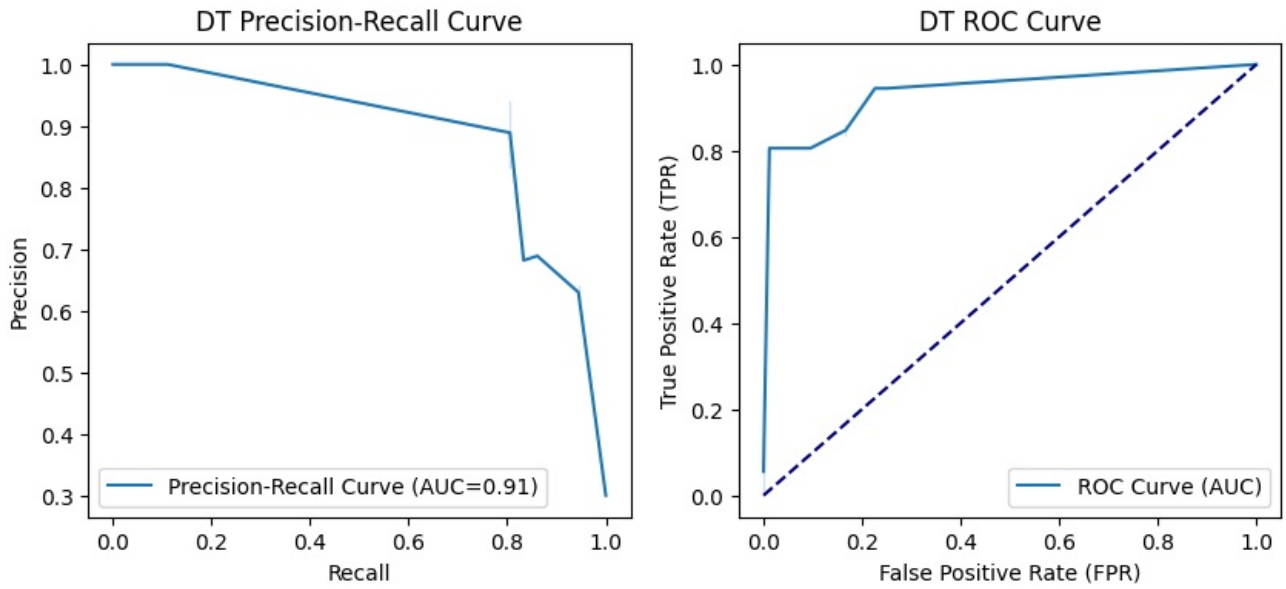


In [ ]:
```
print(classification_report(ytest,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      0.94      0.93        84
           1       0.85      0.81      0.83        36

    accuracy                           0.90       120
   macro avg       0.89      0.87      0.88       120
weighted avg       0.90      0.90      0.90       120
```

In [ ]:
```
yscore = rfe_dt.predict_proba(Xtest)[:,1]
plt.figure(figsize=(10,4))
precision, recall, thresholds = precision_recall_curve(ytest, yscore)
auc_score = auc(recall,precision)
plt.subplot(1,2,1)
sns.lineplot(x=recall, y=precision, label=f'Precision-Recall Curve (AUC={auc_score:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('DT Precision-Recall Curve')
fpr, tpr, thresholds = roc_curve(ytest, yscore)
plt.subplot(1,2,2)
sns.lineplot(x=fpr, y=tpr, label='ROC Curve (AUC)')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('DT ROC Curve')
```

```
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.show()
```



Model: XGBoost

Initial Model

In [ ]:
```
xgb_model = XGBClassifier()
```

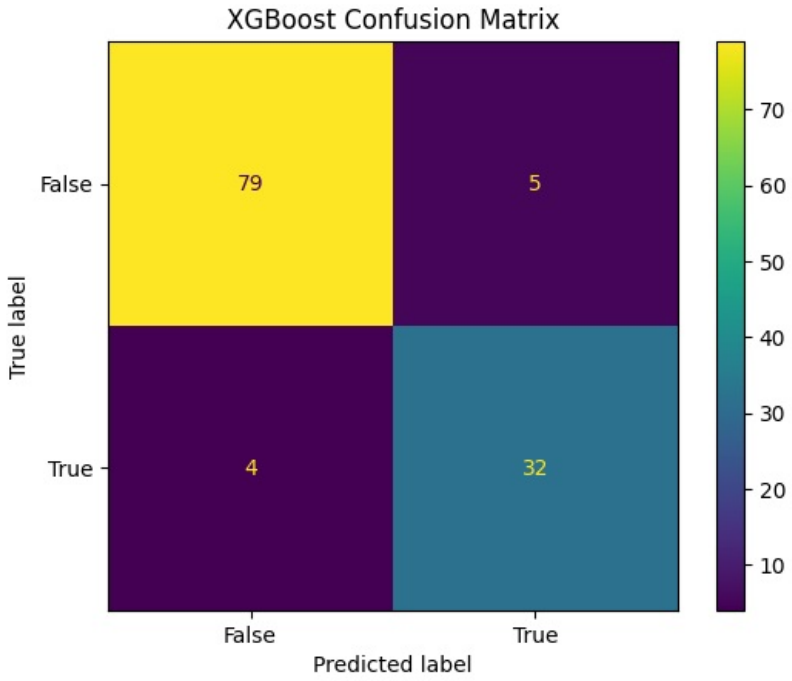
In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,test_size=0.2,random_state=42)
xgb_model.fit(X_train,y_train)
y_pred= xgb_model.predict(X_test)
print(f'Accuracy:{accuracy_score(y_test,y_pred)*100:.4f}%')
```

```
Accuracy:89.5833%
```

In [ ]:
```
y_pred = xgb_model.predict(Xtest)
print(f'Unseen Data Accuracy:{accuracy_score(ytest,y_pred)*100:.4f}%')
print(f'Unseen Data Recall:{recall_score(ytest,y_pred)*100:.4f}%')
```

```
Unseen Data Accuracy:92.5000%
Unseen Data Recall:88.8889%
```

In [ ]:
```
confusion_mat = confusion_matrix(ytest,y_pred)
cm_display = ConfusionMatrixDisplay(confusion_matrix = confusion_mat, display_labels = ['False', 'True'])
cm_display.plot()
plt.title("XGBoost Confusion Matrix")
plt.show()
```



In [ ]:
```
print(classification_report(ytest,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.95      0.94      0.95        84
           1       0.86      0.89      0.88        36

    accuracy                           0.93       120
   macro avg       0.91      0.91      0.91       120
weighted avg       0.93      0.93      0.93       120
```

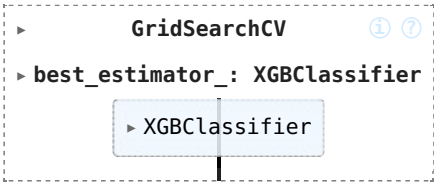
Hyperparameter tuning using *GridSearchCV* (estimated run-time: 1 hour)

In [ ]:
```
param_dist = {
    'learning_rate': [0.001,0.01],
    'n_estimators': [10, 100, 1000],
    'min_child_weight': [1, 3, 5],
    'gamma': [0, 0.1, 0.3],
    'reg_lambda':[1,10,100],
    'reg_alpha':[0.01,0.05]
}
```

Hyperparameter tuning using *GridSearchCV* (estimated run-time: 15-20 mins)

In [ ]:
```
grid_search = GridSearchCV(estimator=xgb_model,param_grid=param_dist,scoring="accuracy",cv=10)
grid_search.fit(X_train,y_train)
```

Out[ ]:



In [ ]:
```
best_params = grid_search.best_params_
learning_rate = best_params['learning_rate']
n_estimators = best_params['n_estimators']
min_child_weight = best_params['min_child_weight']
gamma = best_params['gamma']
reg_lambda = best_params['reg_lambda']
reg_alpha = best_params['reg_alpha']
```

In [ ]:
```
best_xgb=XGBClassifier(learning_rate=learning_rate, n_estimators=n_estimators,min_child_weight=min_child_weight
rfe_xgb = RFECV(best_xgb,min_features_to_select=20,cv=kf,step=1)
```

In [ ]:
```
scores=[]
precision = []
recall = []
for train_index, test_index in kf.split(X_resampled,y_resampled):
  X_train, X_test = X_resampled.iloc[train_index], X_resampled.iloc[test_index]
  y_train, y_test = y_resampled.iloc[train_index], y_resampled.iloc[test_index]
  rfe_xgb.fit(X_train, y_train)
  y_pred = rfe_xgb.predict(X_test)
  score = accuracy_score(y_test,y_pred)
  rec = recall_score(y_test,y_pred)
  prec = precision_score(y_test,y_pred)
  scores.append(score)
  recall.append(rec)
  precision.append(prec)
  print(f'Fold Score: {score}')
```

```
Fold Score: 0.95
Fold Score: 0.9
Fold Score: 0.8
Fold Score: 1.0
Fold Score: 0.9473684210526315
Fold Score: 1.0
Fold Score: 0.9473684210526315
Fold Score: 1.0
Fold Score: 0.9473684210526315
Fold Score: 1.0
Fold Score: 1.0
Fold Score: 1.0
Fold Score: 0.9473684210526315
Fold Score: 0.8947368421052632
Fold Score: 0.8421052631578947
Fold Score: 0.8947368421052632
Fold Score: 0.9473684210526315
Fold Score: 1.0
Fold Score: 1.0
Fold Score: 0.9473684210526315
Fold Score: 0.9473684210526315
Fold Score: 0.9473684210526315
Fold Score: 0.9473684210526315
Fold Score: 0.7894736842105263
Fold Score: 0.9473684210526315
```

In [ ]:
```
average_score = sum(scores) / len(scores)
average_precision = sum(precision)/len(precision)
average_recall = sum(recall)/len(recall)
Accuracy['XGBoost'] = average_score
All_prec['XGBoost'] = average_precision
All_rec['XGBoost'] = average_recall
print(f'Average Cross-Validation Score: {average_score*100:.4f}')
print(f'Average Precision score: {average_precision*100:.4f}')
print(f'Average Recall score: {average_recall*100:.4f}')
```

```
Average Cross-Validation Score: 94.1789
Average Precision score: 93.8496
Average Recall score: 94.6838
```

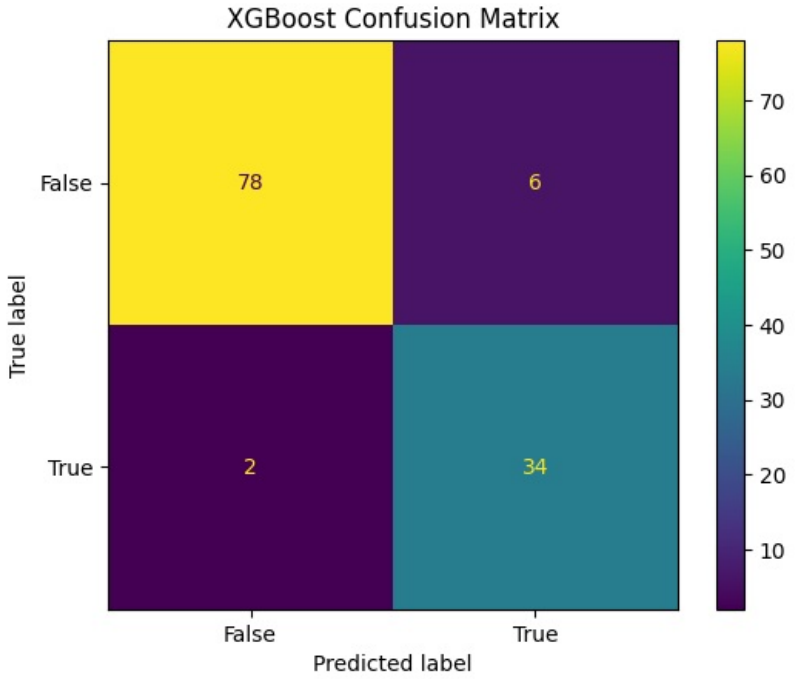
In [ ]:
```
y_pred = rfe_xgb.predict([[45,90,1,0,1,1,0,0,0,230,57,0,140,6,20,3700,0,0,0,1,0,0,0]])
print(y_pred)
```

```
[1]
```

Testing optimized model on unseen data

In [ ]:
```
y_pred = rfe_xgb.predict(Xtest)
test_accuracy['XGBoost'] = accuracy_score(ytest,y_pred)
test_recall['XGBoost'] = recall_score(ytest,y_pred)
test_precision['XGBoost'] = precision_score(ytest,y_pred)
```

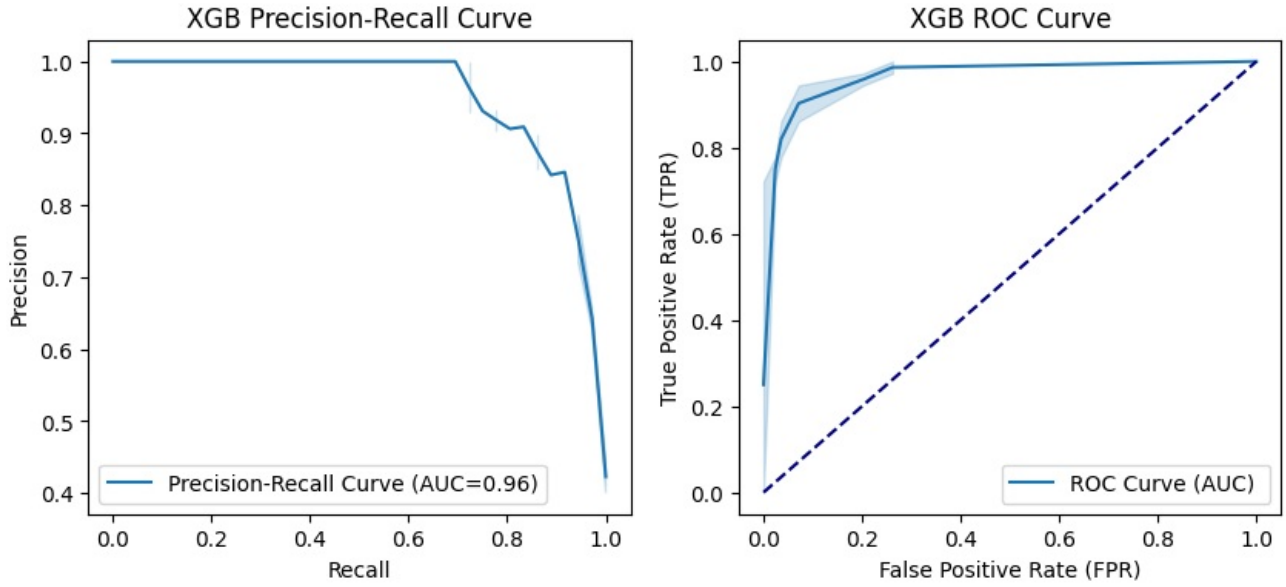
In [ ]:
```
confusion_mat=confusion_matrix(y_pred=y_pred,y_true=ytest)
cm_display = ConfusionMatrixDisplay(confusion_matrix = confusion_mat, display_labels = ['False', 'True'])
cm_display.plot()
plt.title('XGBoost Confusion Matrix')
plt.show()
```



In [ ]:
```
print(classification_report(ytest,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.93      0.95        84
           1       0.85      0.94      0.89        36

    accuracy                           0.93       120
   macro avg       0.91      0.94      0.92       120
weighted avg       0.94      0.93      0.93       120
```

In [ ]:
```
yscore = rfe_xgb.predict_proba(Xtest)[:,1]
plt.figure(figsize=(10,4))
precision, recall, thresholds = precision_recall_curve(ytest, yscore)
auc_score = auc(recall, precision)
plt.subplot(1,2,1)
sns.lineplot(x=recall, y=precision, label=f'Precision-Recall Curve (AUC={auc_score:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('XGB Precision-Recall Curve')
fpr, tpr, thresholds = roc_curve(ytest, yscore)
plt.subplot(1,2,2)
sns.lineplot(x=fpr, y=tpr, label='ROC Curve (AUC) ')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('XGB ROC Curve')
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.show()
```



Saving the Model

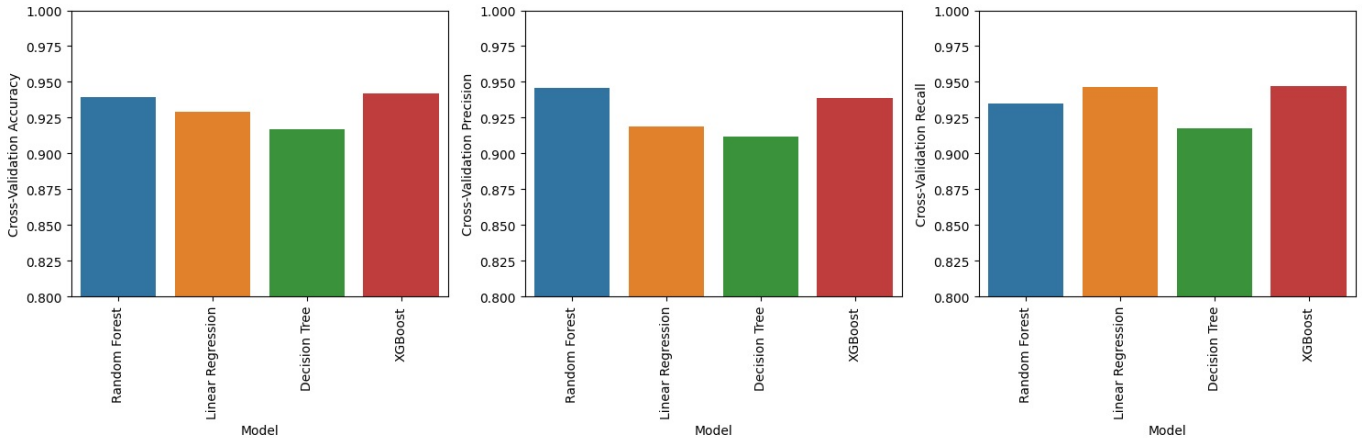
An appropriate model for Early prediction of Chronic Kidney Disease must have a ***high recall score*** i.e. the model is correctly diagnosing a person for CKD when they truly have CKD.
Although precision is not relatively crucial in comparison to recall, a ***reasonable precision*** score i.e. the number of cases which the model correctly predicted a person had CKD out of all the predicted CKD diagnosis.
The final factor to consider will be the accuracy of the model on unseen the unseen test dataset.
Once all the factors are considered, the model can finally be saved.

In [ ]:
```
model_name = [key for key in Accuracy.keys()]
model_accuracy = [acc for acc in Accuracy.values()]
model_precision = [prec for prec in All_prec.values()]
model_recall = [rec for rec in All_rec.values()]
model_data = pd.DataFrame([model_accuracy,model_precision,model_recall],index=['Accuracy','Precision','Recall']
model_data
```

Out[ ]:

	Random Forest	Linear Regression	Decision Tree	XGBoost
Accuracy	0.939579	0.929368	0.916632	0.941789
Precision	0.945841	0.919162	0.911676	0.938496
Recall	0.935202	0.946172	0.917553	0.946838

In [ ]:
```
plt.figure(figsize=(15,5))
plt.subplot(1,3,1)
sns.barplot(x=model_name,y=model_accuracy,hue=model_name)
plt.ylim(0.8,1)
plt.xlabel('Model')
plt.ylabel('Cross-Validation Accuracy')
plt.xticks(rotation=90)
plt.subplot(1,3,2)
sns.barplot(x=model_name,y=model_precision,hue=model_name)
plt.ylim(0.8,1)
plt.xticks(rotation=90)
plt.xlabel('Model')
plt.ylabel('Cross-Validation Precision')
plt.subplot(1,3,3)
sns.barplot(x=model_name,y=model_recall,hue=model_name)
plt.ylim(0.8,1)
plt.xticks(rotation=90)
plt.xlabel('Model')
plt.ylabel('Cross-Validation Recall')
plt.tight_layout()
plt.show()
```



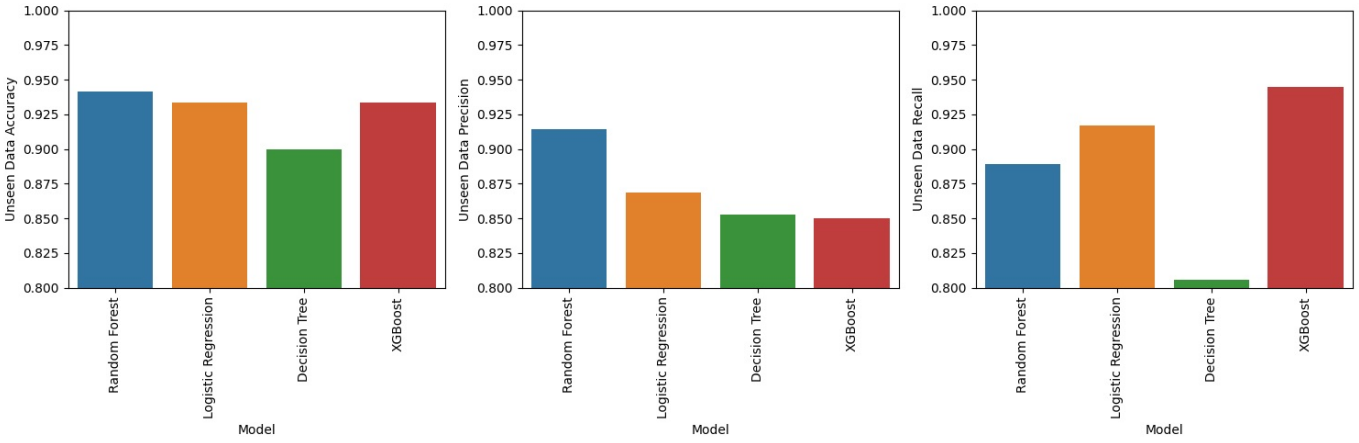
From the above visualisation, `RandomForest Classifier`, `Logistic Regression`, and `XGBoost Classifier` looks promising. `DecisionTree Classifier` overall underperforms in all parameters.

In [ ]:
```
test_name = ['Random Forest','Logistic Regression','Decision Tree','XGBoost']
test_accuracy = [acc for acc in test_accuracy.values()]
test_precision = [prec for prec in test_precision.values()]
test_recall = [rec for rec in test_recall.values()]
test_data = pd.DataFrame([test_accuracy,test_precision,test_recall],index=['Accuracy','Precision','Recall'],colu
test_data
```

Out[ ]:

	Random Forest	Logistic Regression	Decision Tree	XGBoost
Accuracy	0.941667	0.933333	0.900000	0.933333
Precision	0.914286	0.868421	0.852941	0.850000
Recall	0.888889	0.916667	0.805556	0.944444

In [ ]:
```
plt.figure(figsize=(15,5))
plt.subplot(1,3,1)
sns.barplot(x=test_name,y=test_accuracy,hue=test_name)
plt.ylim(0.8,1)
plt.xlabel('Model')
plt.ylabel('Unseen Data Accuracy')
plt.xticks(rotation=90)
plt.subplot(1,3,2)
sns.barplot(x=test_name,y=test_precision,hue=test_name)
plt.ylim(0.8,1)
plt.xticks(rotation=90)
plt.xlabel('Model')
plt.ylabel('Unseen Data Precision')
plt.subplot(1,3,3)
sns.barplot(x=test_name,y=test_recall,hue=test_name)
plt.ylim(0.8,1)
plt.xticks(rotation=90)
plt.xlabel('Model')
plt.ylabel('Unseen Data Recall')
plt.tight_layout()
plt.show()
```



From the above visualisation, it can be observed that `XGBoost Classifier` overshadows other models in *Recall score* while also providing decent *Precision score* and *Accuracy*
Hence, we will go forward with Random Forest Classifier and save it.

In [ ]:
```
dump(rfe_xgb,open('CKD.pkl','wb'))
dump(label_enc,open('Label_Encoder.pkl','wb')) #dumping label encoder to use it to encode the user inserted deta
```

Throughout the training the model uses RFE (Recursive Feature Elimination) to eliminate features that have considerably lower contribution towards accuracy. The following columns were retained from the dataset in the final RFE model.

In [ ]:
```
retained_features = X.columns[rfe_xgb.support_]
for i in retained_features:
    print(features[i])
```

```
Age
Blood_Pressure
Specific_gravity
Albumin
Sugar
Red_blood_cells
Pus_cell
Pus_cell_clumps
Bacteria
Blood_Glucose_Random
Blood_urea
Serum_creatinine
Sodium
Potassium
Haemoglobin
White_blood_cell_count
Red_blood_cell_count
Hypertension
Diabetes_mellitus
Coronary_artery_disease
```

In [ ]:
```
dump(retained_features,open("Retained_Features.pkl","wb")) #dumping retained features to discard features that
```
