CKD_Predict_notebook

February 10, 2023

[203]:
```
# Project 2 CKD Prediction
# Christine Orosco 1/30/2023
'''
Predict the probabiity of a person being diagnosed with CKD

'''
```

[203]:
```
'\nPredict the probabiity of a person being diagnosed with CKD\n\n'
```

[8]:
```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import numpy as np
```

1 Perform Data Cleaning

[9]:
```
# Read in data set
```

[72]:
```
# The UCI dataset

df = pd.read_csv('ckd_dataset.csv', header=0)
```

[73]:
```
df.head()
```

[73]:

	age	bp	sg	al	su	rbc	pc	ba	bgr	bu	…	wbcc	rbcc	\
0	48	80	1.020	1	0	?	normal	notpresent	121	36	…	7800	5.2	
1	7	50	1.020	4	0	?	normal	notpresent	?	18	…	6000	?	
2	62	80	1.010	2	3	normal	normal	notpresent	423	53	…	7500	?	
3	48	70	1.005	4	0	normal	abnormal	notpresent	117	56	…	6700	3.9	
4	51	80	1.010	2	0	normal	normal	notpresent	106	26	…	7300	4.6	

	htn	dm	cad	appet	pe	ane	class	Unnamed: 24
0	yes	yes	no	good	no	no	ckd	NaN
1	no	no	no	good	no	no	ckd	NaN
2	no	yes	no	poor	no	yes	ckd	NaN

```
3  yes   no  no  poor  yes  yes   ckd         NaN
4   no   no  no  good   no   no   ckd         NaN

[5 rows x 25 columns]
```

[74]:
```
#Define column names with abreviations

labels = {
    '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': 'hemoglobin',
    'pcv': 'packed_cell_volume',
    'wbcc': 'white_blood_cell_count',
    'rbcc': 'red_blood_cell_count',
    'htn': 'hypertension',
    'dm': 'diabetes_mellitus',
    'cad': 'coronary_artery_disease',
    'appet': 'appetite',
    'pe': 'pedal_edema',
    'ane': 'anemia',
    'class': 'class'}
```

[75]:
```
#Clean the dataset

# Remove bogus column Unamed: 24
#df.drop(['Unnamed: 24'], axis=1, inplace=True)
df
```

[75]:
```
     age  bp     sg al su     rbc        pc          ba  bgr  bu  …  wbcc  \
0     48  80  1.020  1  0       ?    normal  notpresent  121  36  …  7800
1      7  50  1.020  4  0       ?    normal  notpresent    ?  18  …  6000
2     62  80  1.010  2  3  normal    normal  notpresent  423  53  …  7500
3     48  70  1.005  4  0  normal  abnormal  notpresent  117  56  …  6700
4     51  80  1.010  2  0  normal    normal  notpresent  106  26  …  7300
..    ..  ..    … .. ..     …         …          …   …  ..  …   …
```

```
395  55  80  1.020  0  0  normal    normal  notpresent  140  49  …  6700
396  42  70  1.025  0  0  normal    normal  notpresent   75  31  …  7800
397  12  80  1.020  0  0  normal    normal  notpresent  100  26  …  6600
398  17  60  1.025  0  0  normal    normal  notpresent  114  50  …  7200
399  58  80  1.025  0  0  normal    normal  notpresent  131  18  …  6800

    rbcc  htn   dm cad appet   pe  ane   class Unnamed: 24
0    5.2  yes  yes  no  good   no   no     ckd         NaN
1      ?   no   no  no  good   no   no     ckd         NaN
2      ?   no  yes  no  poor   no  yes     ckd         NaN
3    3.9  yes   no  no  poor  yes  yes     ckd         NaN
4    4.6   no   no  no  good   no   no     ckd         NaN
..   …  …  …  ..   …  …  …     …          …
395  4.9   no   no  no  good   no   no  notckd         NaN
396  6.2   no   no  no  good   no   no  notckd         NaN
397  5.4   no   no  no  good   no   no  notckd         NaN
398  5.9   no   no  no  good   no   no  notckd         NaN
399  6.1   no   no  no  good   no   no  notckd         NaN

[400 rows x 25 columns]
```

[76]:
```
#Check the col defs
df.dtypes
```

[76]:
```
age            object
bp             object
sg             object
al             object
su             object
rbc            object
pc             object
ba             object
bgr            object
bu             object
sc             object
sod            object
pot            object
hemo           object
pcv            object
wbcc           object
rbcc           object
htn            object
dm             object
cad            object
appet          object
pe             object
ane            object
```

```
class           object
Unnamed: 24     object
dtype: object
```

[77]:
```
# Change unknown values with ?
# Take the average or use the lowest common values.
# First find any NAN values

df.isnull().sum()
```

[77]:
```
age              0
bp               0
sg               0
al               0
su               0
rbc              0
pc               0
ba               0
bgr              0
bu               0
sc               0
sod              0
pot              0
hemo             0
pcv              0
wbcc             0
rbcc             0
htn              0
dm               1
cad              0
appet            0
pe               0
ane              0
class            0
Unnamed: 24    399
dtype: int64
```

[78]:
```
# Look at the occurrence of the values
df['rbcc'].describe()
```

[78]:
```
count     400
unique     50
top         ?
freq      130
Name: rbcc, dtype: object
```

[79]:
```
df['su']
```

[79]:
```
0      0
1      0
2      3
3      0
4      0
      ..
395    0
396    0
397    0
398    0
399    0
Name: su, Length: 400, dtype: object
```

[80]:
```
#Fill NaN value
df['dm'].fillna('normal',inplace=True)
```

[81]:
```
# Replace the Question marks.
df['rbc'] = df['rbc'].replace({"?":"normal"})
```

[82]:
```
# Replace bgr
df['bgr'] = df['bgr'].replace({"?":150})
```

[83]:
```
# Replace rbcc
df['rbcc'] = df['rbcc'].replace({"?":5.0})
```

[84]:
```
# Look for other columns with ? mark
clean_lst = []
names = df.columns
for i in names:
    for j in df[i]:
        if j == "        ?":
            clean_lst.append(i)
            break
```

[85]:
```
clean_lst
```

[85]:
```
['pcv', 'wbcc', 'rbcc']
```

[86]:
```
# Replace pcv
df['pcv'] = df['pcv'].replace({"        ?":49})
```

[87]:
```
# Replace values
def re_value(col, new_val):
        df[col] = df[col].replace({"        ?":new_val})
        return
```

[88]:
```
col = 'rbcc'
new_val = 8.5
re_value(col, new_val)
```

[89]:
```
# Look at the occurrence of the values
df.describe()
```

[89]:
```
        age   bp     sg   al   su     rbc      pc          ba  bgr   bu  … \
count   400  400    400  400  400     400     400         400  400  400  …
unique   77   11      6    7    7       2       3           3  147  119  …
top      60   80  1.020    0    0  normal  normal  notpresent  150    ?  …
freq     19  116    106  199  290     353     259         374   44   19  …

       wbcc   rbcc  htn   dm  cad appet   pe  ane class Unnamed: 24
count   400  400.0  400  400  400   400  400  400   400           1
unique   91   50.0    3    7    5     4    4    3     4           1
top       ?    5.0   no   no   no  good   no   no   ckd      notckd
freq    105  130.0  251  257  362   316  322  339   248           1

[4 rows x 25 columns]
```

[118]:
```
# Look for other columns with ? mark
clean_lst = []
names = df.columns
for i in names:
    for j in df[i]:
        if j == "?":
            clean_lst.append(i)
            break
```

[122]:
```
# Look for other columns with ? mark
clean_lst2 = []
names = df.columns
for i in names:
    for j in df[i]:
        if j == "        ?":
            clean_lst2.append(i)
            break
```

[123]:
```
clean_lst2
```

[123]:
```
['wbcc']
```

[119]:
```
clean_lst
```

[119]:
```
['bu']
```

[92]:
```
# Replace values
def re_value(col, new_val):
        df[col] = df[col].replace({"?":new_val})
        return
```

[93]:
```
#{age:55, bp:80,sg:1.010, al:1,su:0,pc:normal(0)abnormal=1,ba:
 ↪0=present,1=notpresent
#sc:9.0,sod:135,pot:8.0,hemo:13.5,pcv:38,wbc:8000, htn:0=no,1=yes,dm:00=no,1=yes
#cad:00=no,1=yes,appet:poor=1good=0, pe00=no,1=yes, ane00=no,1=yes,rbc:
 ↪normal(0)abnormal=1

new_data = {'age':55, 'bp':80,'sg':1.010, 'al':1,'su':0,
            'sc':9.0,'sod':135,'pot':8.0,'hemo':13.5,'pcv':38,'wbcc':8000}

for k,v in new_data.items():
    col = k
    new_val = v
    new_col = re_value(col, new_val)
```

[94]:
```
df['bu'].head(15)
```

[94]:
```
0      36
1      18
2      53
3      56
4      26
5      25
6      54
7      31
8      60
9     107
10     55
11     60
12     72
13     86
14     90
Name: bu, dtype: object
```

[120]:
```
df['bu'] = df['bu'].replace({"?":90})
```

[124]:
```
df['wbcc'] = df['wbcc'].replace({"        ?":8000})
```

[95]:
```
# Define cat cols
cat_cols = ['htn', 'dm', 'cad', 'pe', 'ane', 'class', 'rbc', 'pc', 'appet',␣
 ↪'ba']
#cat_cols1 = ['htn', 'dm', 'cad', 'pe', 'ane']
#cat_cols2 = ['rbc', 'pc']
```

```
#cat_cols3 = ['appet']
#cat_cols4 = ['ba']
```

[96]:
```
# Replace cat values with binary values

for i in cat_cols:
    df[i] = df[i].replace({"?":0})
```

[97]:
```
#Replace other extraneous chars
df['pc'] = df['pc'].replace({"?":0})
df['class'] = df['class'].replace({"ckd":1, "notckd":0, "'ckd\t":1})
df['pe'] = df['pe'].replace({"good":0})
df['appet'] = df['appet'].replace({"no":0, "?":1})
df['ba'] = df['ba'].replace({"?":1})

df['dm'] = df['dm'].replace({" yes":1, "\tno":0,"\tyes":1,"normal":0})
```

[98]:
```
df['dm'] = df['dm'].replace({"yes":1, "no":0,})
```

[99]:
```
df['class'] = df['class'].replace({"ckd\t":1})
```

[115]:
```
# Check for other values in categorical columns
for i in cat_cols:
    print(f'{i} {df[i].unique()}')
```

```
htn [1 0]
dm [1 0]
cad [0 1]
pe [0 1]
ane [0 1]
class [1 0]
rbc [0 1]
pc [0 1]
appet [0 1]
ba [0 1]
```

[101]:
```
df['class'] = df['class'].replace({"no":0})
```

[114]:
```
df['htn'] = df['htn'].replace({"no":0, "yes":1})
```

[113]:
```
df['rbc'] = df['rbc'].replace({"normal":0, "abnormal":1})
```

[103]:
```
df['cad'] = df['cad'].replace({"no":0, "yes":1, "No":0, "\tno":0})
```

[104]:
```
df['ane'] = df['ane'].replace({"no":0, "yes":1})
```

[105]:
```
df['pe'] = df['pe'].replace({"no":0, "yes":1})
```

[106]:
```
df['htn'] = df['htn'].replace({"normal":0, "abnormal":1})
```

[107]:
```
df['pc'] = df['pc'].replace({"normal":0, "abnormal":1})
```

[108]:
```
df['appet'] = df['appet'].replace({"good":0, "poor":1})
```

[109]:
```
df['ba'] = df['ba'].replace({"notpresent":0, "present":1})
```

[116]:
```
df.dtypes
```

[116]:
```
age      object
bp       object
sg       object
al       object
su       object
rbc       int64
pc        int64
ba        int64
bgr      object
bu       object
sc       object
sod      object
pot      object
hemo     object
pcv      object
wbcc     object
rbcc     object
htn       int64
dm        int64
cad       int64
appet     int64
pe        int64
ane       int64
class     int64
dtype: object
```

[111]:
```
df.drop(columns=['Unnamed: 24'], axis=1, inplace=True)
```

[125]:
```
# Change df to numeric
df = df.apply(pd.to_numeric)
```

[126]:
```
df.dtypes
```

[126]:
```
age        int64
bp         int64
sg       float64
al         int64
```

```
su          int64
rbc         int64
pc          int64
ba          int64
bgr         int64
bu        float64
sc        float64
sod       float64
pot       float64
hemo      float64
pcv         int64
wbcc        int64
rbcc      float64
htn         int64
dm          int64
cad         int64
appet       int64
pe          int64
ane         int64
class       int64
dtype: object
```

[127]:
```
# Keep a copy just incase need to revert back to this
df_copy = df
df_copy.to_csv('new_ckd_data2.csv')
```

[128]:
```
#Read in the new dataset
df = pd.read_csv('new_ckd_data2.csv', header=0)
```

[129]:
```
df.drop("Unnamed: 0", axis=1, inplace=True)
```

[40]:
```
# Check values in certain columns
```

2 Perform Data Correlations and Relationship Analysis

[130]:
```
# Compute correlation between features
corr_df = df.corr()
corr_df
```

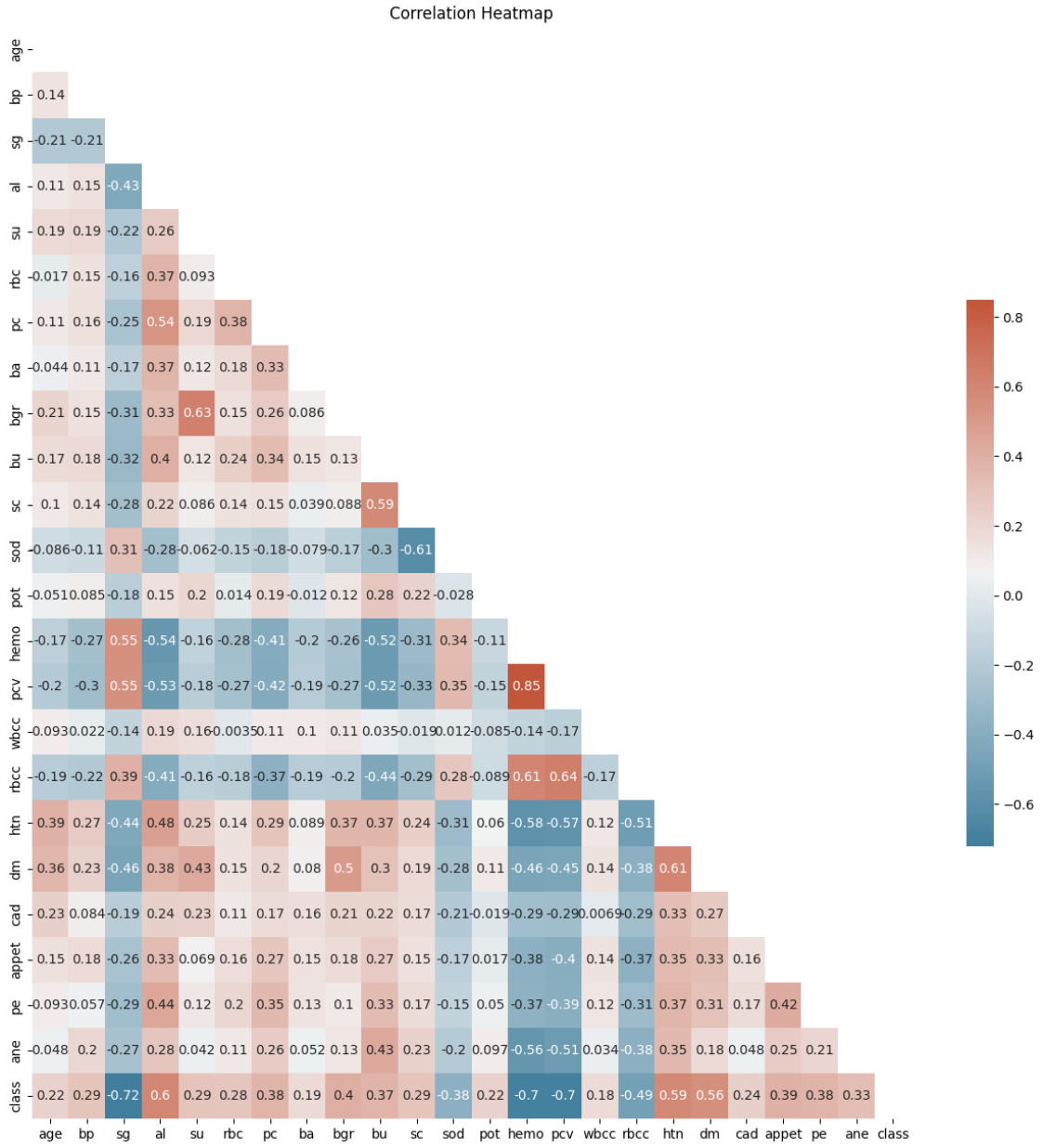
[130]:

	age	bp	sg	al	su	rbc	pc	\
age	1.000000	0.136121	-0.212145	0.111207	0.185521	0.016852	0.108413	
bp	0.136121	1.000000	-0.211619	0.147564	0.189561	0.150384	0.156231	
sg	-0.212145	-0.211619	1.000000	-0.427115	-0.222632	-0.161723	-0.251734	
al	0.111207	0.147564	-0.427115	1.000000	0.262564	0.374484	0.535895	
su	0.185521	0.189561	-0.222632	0.262564	1.000000	0.092940	0.190062	
rbc	0.016852	0.150384	-0.161723	0.374484	0.092940	1.000000	0.377394	

```
pc     0.108413  0.156231 -0.251734  0.535895  0.190062  0.377394  1.000000
ba     0.043633  0.110164 -0.165956  0.368222  0.119399  0.184402  0.330401
bgr    0.212837  0.149744 -0.311199  0.327390  0.629700  0.154056  0.263195
bu     0.170725  0.182174 -0.320948  0.396791  0.122439  0.237919  0.338142
sc     0.101705  0.139881 -0.283513  0.215704  0.085736  0.143396  0.152913
sod   -0.085662 -0.107184  0.311943 -0.281730 -0.062081 -0.147691 -0.183487
pot    0.051353  0.084987 -0.175673  0.152497  0.195494  0.014188  0.185147
hemo  -0.168518 -0.273970  0.550722 -0.543178 -0.155351 -0.279270 -0.410533
pcv   -0.204758 -0.295150  0.546819 -0.525709 -0.182275 -0.271486 -0.418691
wbcc   0.092800  0.022177 -0.141381  0.191492  0.156649 -0.003471  0.106733
rbcc  -0.190103 -0.215067  0.393612 -0.411869 -0.156812 -0.181685 -0.372117
htn    0.386340  0.266901 -0.438216  0.480675  0.254268  0.140538  0.291719
dm     0.355239  0.226489 -0.458341  0.377038  0.430514  0.145646  0.201032
cad    0.232499  0.084135 -0.186755  0.236254  0.229301  0.111493  0.172295
appet  0.150940  0.175054 -0.259634  0.330264  0.069216  0.160868  0.274985
pe     0.093422  0.056902 -0.289736  0.440612  0.116442  0.199285  0.350227
ane    0.048092  0.195134 -0.265119  0.281546  0.042464  0.107625  0.260566
class  0.222755  0.293693 -0.720910  0.598389  0.294555  0.282642  0.375154

              ba       bgr        bu  …       pcv      wbcc      rbcc  \
age    0.043633  0.212837  0.170725  … -0.204758  0.092800 -0.190103
bp     0.110164  0.149744  0.182174  … -0.295150  0.022177 -0.215067
sg    -0.165956 -0.311199 -0.320948  …  0.546819 -0.141381  0.393612
al     0.368222  0.327390  0.396791  … -0.525709  0.191492 -0.411869
su     0.119399  0.629700  0.122439  … -0.182275  0.156649 -0.156812
rbc    0.184402  0.154056  0.237919  … -0.271486 -0.003471 -0.181685
pc     0.330401  0.263195  0.338142  … -0.418691  0.106733 -0.372117
ba     1.000000  0.086393  0.149388  … -0.190211  0.104748 -0.188575
bgr    0.086393  1.000000  0.131373  … -0.269870  0.114906 -0.195127
bu     0.149388  0.131373  1.000000  … -0.519610  0.034946 -0.439877
sc     0.039119  0.087844  0.589841  … -0.333726 -0.018594 -0.285009
sod   -0.078859 -0.172496 -0.295783  …  0.350864  0.012154  0.277338
pot   -0.012290  0.119964  0.276549  … -0.147251 -0.084916 -0.089104
hemo  -0.202931 -0.263433 -0.523121  …  0.849999 -0.139315  0.610464
pcv   -0.190211 -0.269870 -0.519610  …  1.000000 -0.171419  0.643391
wbcc   0.104748  0.114906  0.034946  … -0.171419  1.000000 -0.165621
rbcc  -0.188575 -0.195127 -0.439877  …  0.643391 -0.165621  1.000000
htn    0.089046  0.369813  0.367661  … -0.567821  0.116827 -0.512260
dm     0.080070  0.500175  0.300813  … -0.451991  0.144101 -0.378652
cad    0.162395  0.212723  0.217758  … -0.292816  0.006923 -0.288533
appet  0.149126  0.175960  0.266698  … -0.398166  0.142714 -0.374219
pe     0.134732  0.101941  0.332424  … -0.391370  0.123381 -0.307671
ane    0.052208  0.127844  0.427274  … -0.513960  0.034059 -0.377516
class  0.186871  0.402785  0.372208  … -0.698160  0.177571 -0.488995

             htn        dm       cad     appet        pe       ane     class
age    0.386340  0.355239  0.232499  0.150940  0.093422  0.048092  0.222755
```

```
bp      0.266901  0.226489  0.084135  0.175054  0.056902  0.195134  0.293693
sg     -0.438216 -0.458341 -0.186755 -0.259634 -0.289736 -0.265119 -0.720910
al      0.480675  0.377038  0.236254  0.330264  0.440612  0.281546  0.598389
su      0.254268  0.430514  0.229301  0.069216  0.116442  0.042464  0.294555
rbc     0.140538  0.145646  0.111493  0.160868  0.199285  0.107625  0.282642
pc      0.291719  0.201032  0.172295  0.274985  0.350227  0.260566  0.375154
ba      0.089046  0.080070  0.162395  0.149126  0.134732  0.052208  0.186871
bgr     0.369813  0.500175  0.212723  0.175960  0.101941  0.127844  0.402785
bu      0.367661  0.300813  0.217758  0.266698  0.332424  0.427274  0.372208
sc      0.240687  0.185612  0.174665  0.154860  0.171187  0.226633  0.294956
sod    -0.306791 -0.277076 -0.210044 -0.167510 -0.148358 -0.199578 -0.379096
pot     0.059907  0.105091 -0.018690  0.017377  0.049738  0.096755  0.224031
hemo   -0.580489 -0.462453 -0.289364 -0.376544 -0.370653 -0.558788 -0.699533
pcv    -0.567821 -0.451991 -0.292816 -0.398166 -0.391370 -0.513960 -0.698160
wbcc    0.116827  0.144101  0.006923  0.142714  0.123381  0.034059  0.177571
rbcc   -0.512260 -0.378652 -0.288533 -0.374219 -0.307671 -0.377516 -0.488995
htn     1.000000  0.608118  0.325479  0.345070  0.371026  0.347802  0.590438
dm      0.608118  1.000000  0.271172  0.325134  0.308463  0.183686  0.559060
cad     0.325479  0.271172  1.000000  0.156104  0.172295  0.047700  0.236088
appet   0.345070  0.325134  0.156104  1.000000  0.417055  0.254942  0.393341
pe      0.371026  0.308463  0.172295  0.417055  1.000000  0.207025  0.375154
ane     0.347802  0.183686  0.047700  0.254942  0.207025  1.000000  0.325396
class   0.590438  0.559060  0.236088  0.393341  0.375154  0.325396  1.000000

[24 rows x 24 columns]
```

[131]:
```
# Define seaborn parameters

#Generate a cubehelix heatmap
cmap = sns.diverging_palette(230, 20, as_cmap=True)

#Create a mask to display the upper triangle
mask = np.triu(np.ones_like(corr_df, dtype=bool))

# Set up the figure bounds

f, ax = plt.subplots(figsize=(15,15))

sns.heatmap(corr_df, mask=mask, cmap=cmap,   cbar_kws={"shrink": .5},␣
 ↪annot=True)
plt.title('Correlation Heatmap')
plt.savefig("sns-heatmap.png")
plt.show()
```



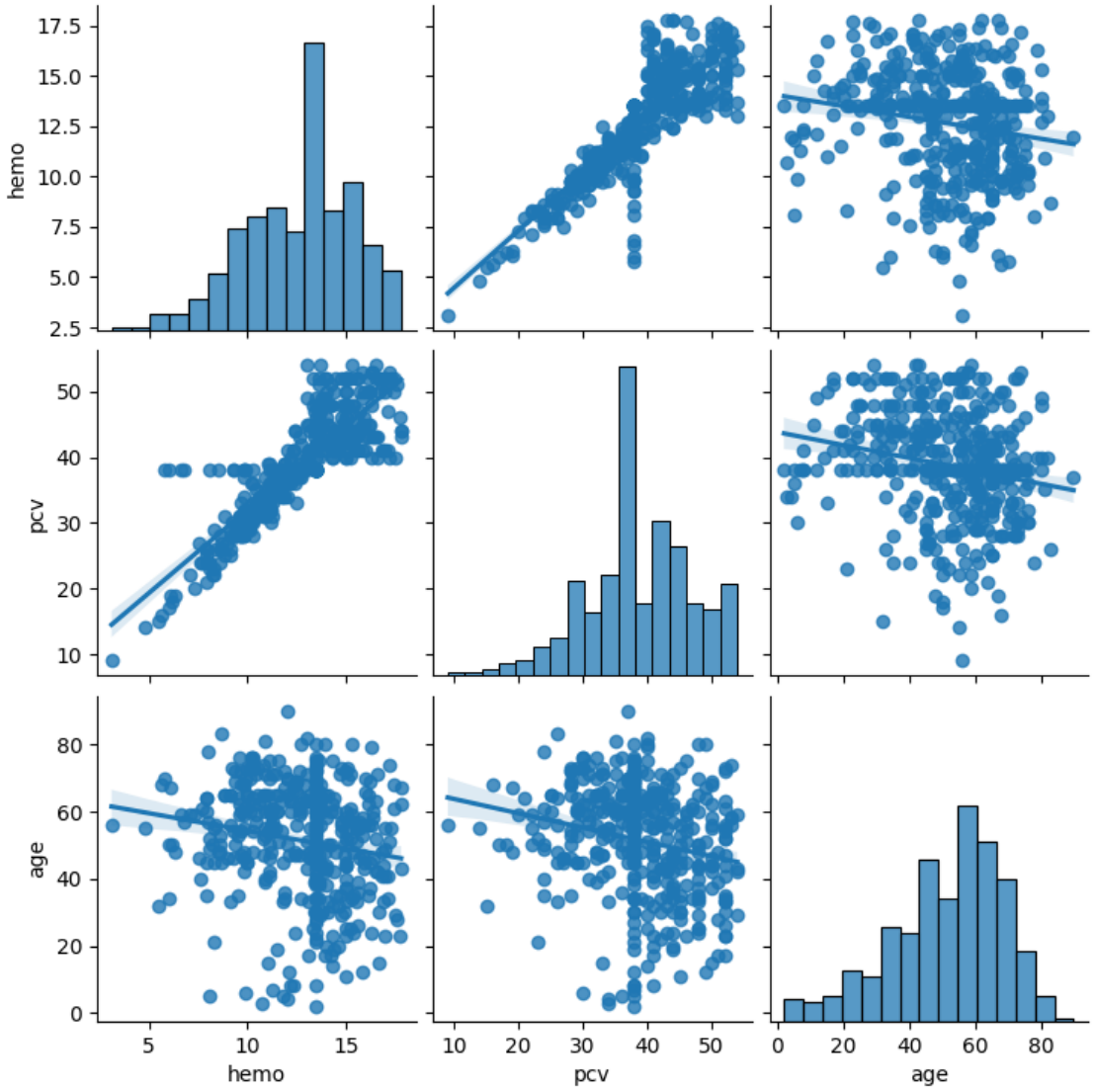
[43]:
```
# There doesn't seem to be many positive correlation between features
# Person with anemia is expected to have a lower hemoglobin and low red blood␣
 ↪cell counts
# Check the analysis
```

[44]:
```
#Look at some variables with seaborn pairplots
```

[134]:
```
plt.figure(figsize=(14,10))
```

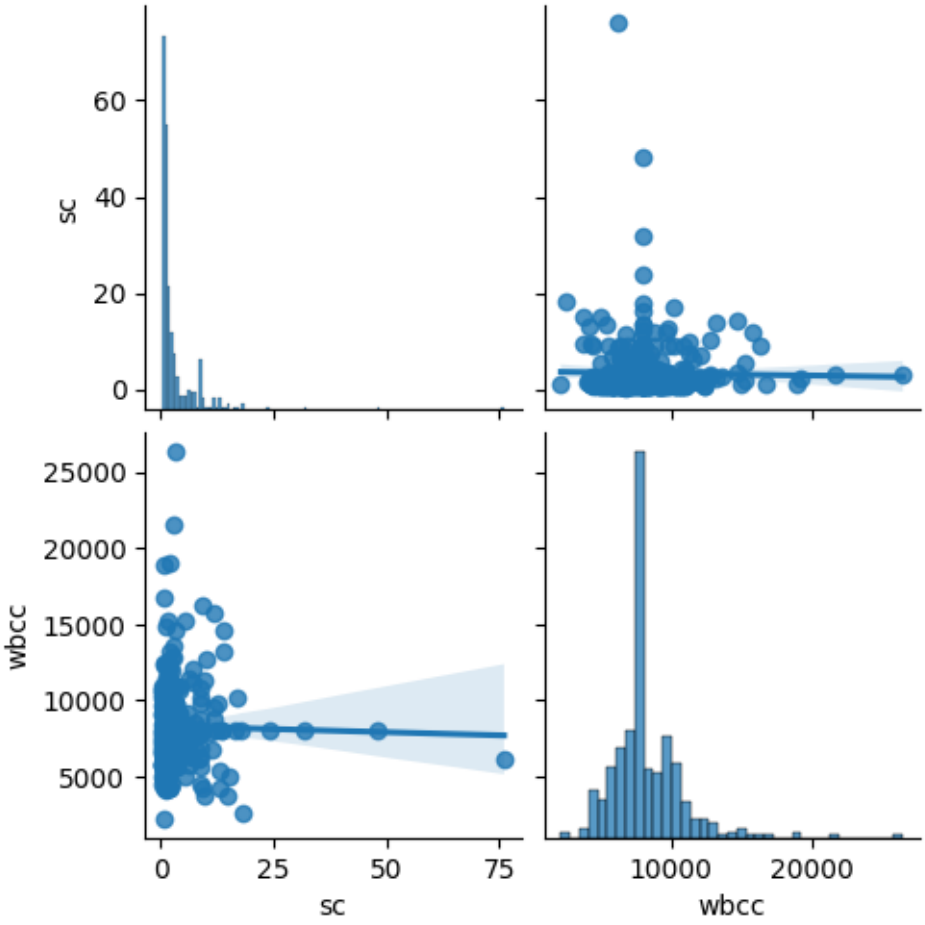
```
sns.pairplot(df[['hemo','pcv','age']], kind="reg")
plt.show()
```

```
<Figure size 1400x1000 with 0 Axes>
```



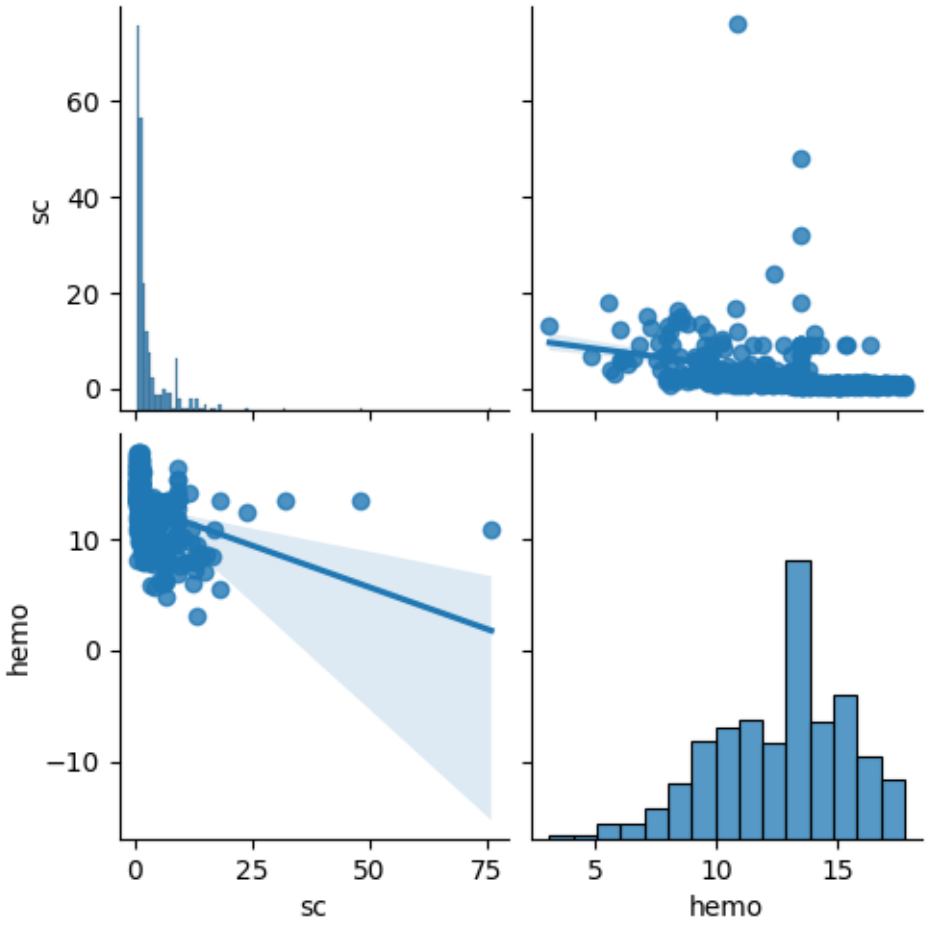
[135]:
```
# Look at serum creatinine and potassium and sodium levels
# There appears to be a linear relationship between sc and hemoglobin

plt.figure(figsize=(10,10))

sns.pairplot(df[['sc', 'wbcc']], kind="reg")
plt.show()
```

<Figure size 1000x1000 with 0 Axes>

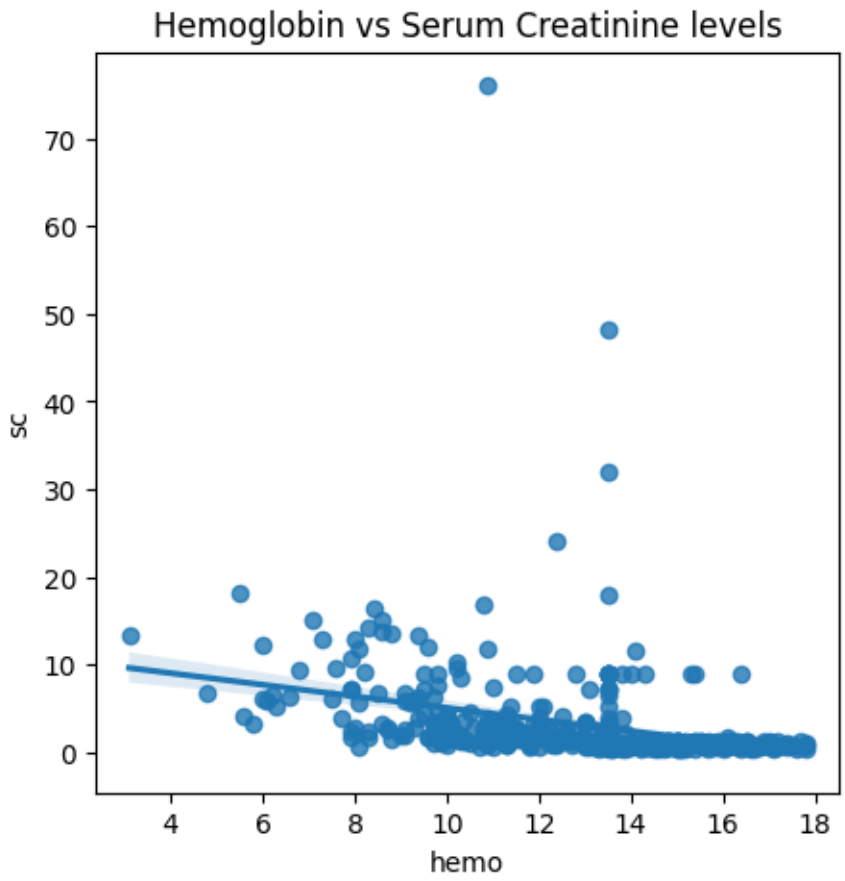


[136]:
```
# Look at serum creatinine and hemoglobin
# There appears to be a linear relationship between sc and hemoglobin

plt.figure(figsize=(14,10))
sns.pairplot(df[['sc', 'hemo']], kind="reg")
plt.show()
```

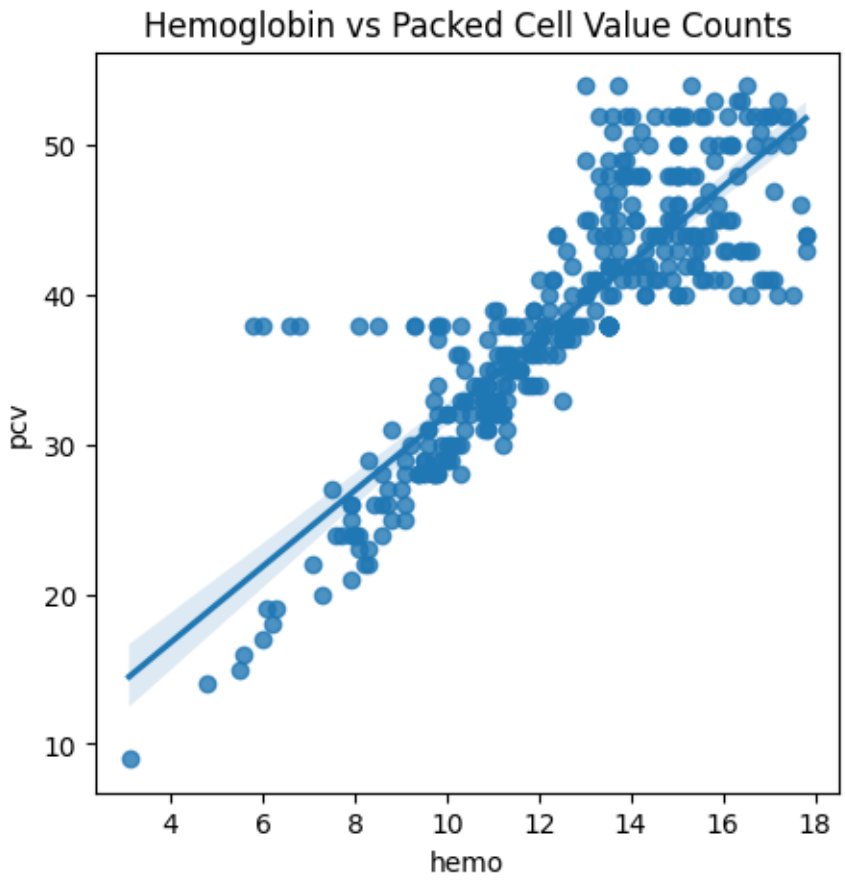
<Figure size 1400x1000 with 0 Axes>



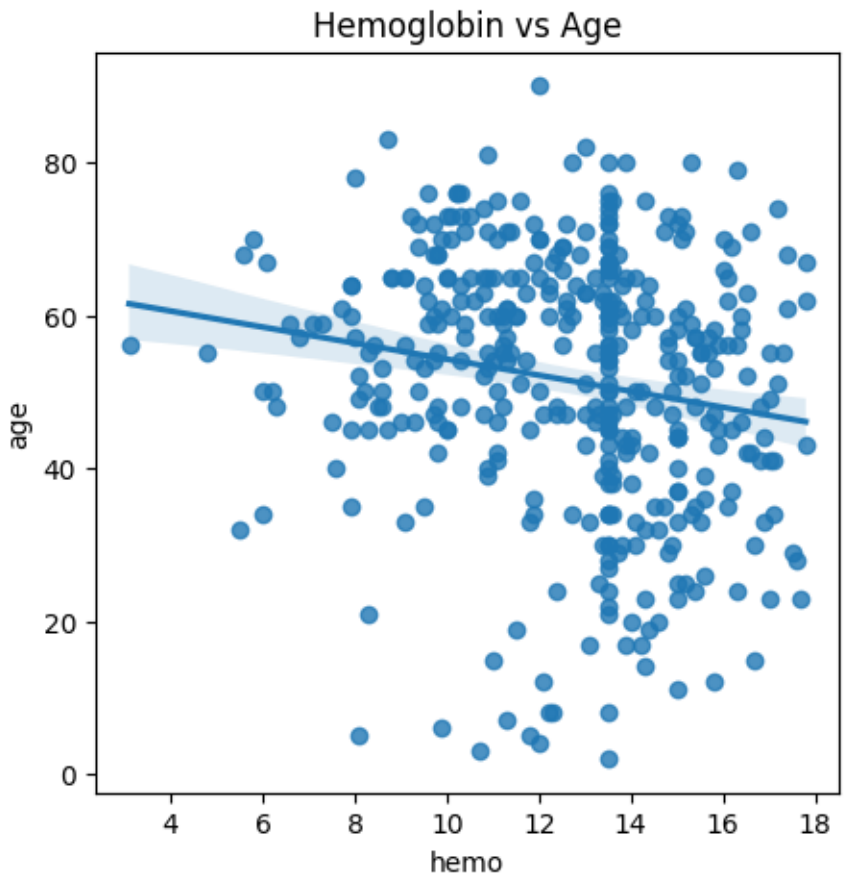
[137]:
```
# look at a specific pair
plt.figure(figsize=(5,5))
xlabel = "Hemogloben Levels"
ylabel = "Serum Creatinine Levels"
sns.regplot(x=df["hemo"], y=df["sc"])
plt.title('Hemoglobin vs Serum Creatinine levels')
plt.show()
```



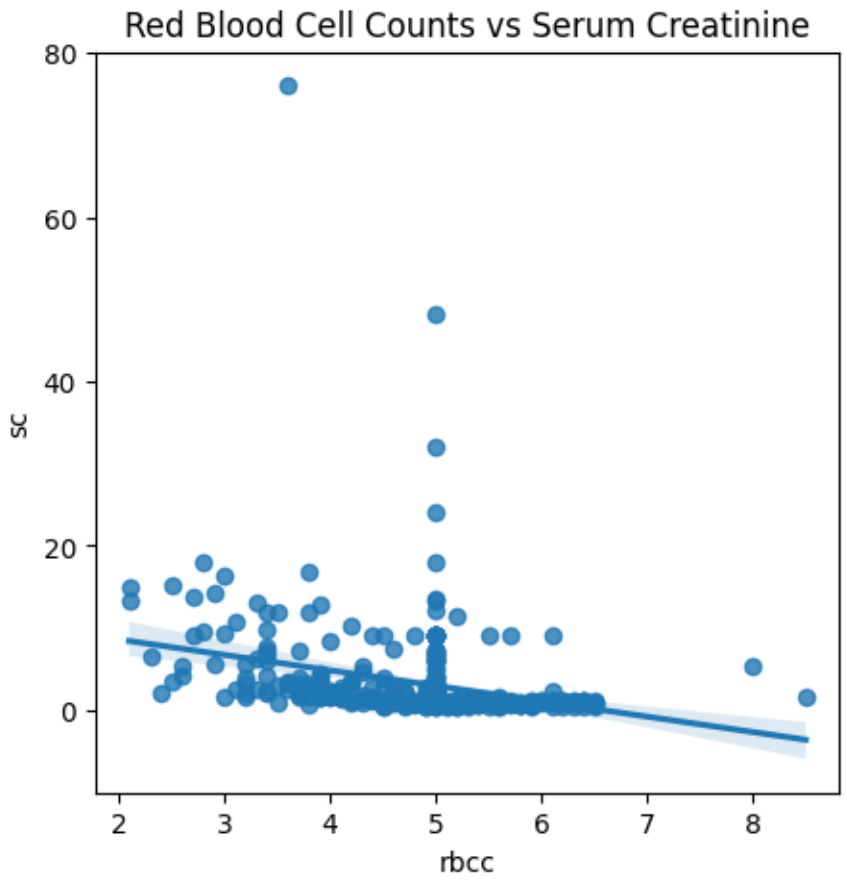
[138]:
```
# look at a specific pair
# This is expected as hemoglobin counts rise so do the number of red blood cells
# This is a natural occurence and in itself does not show propensity towards CKD
# Pvc can be deleted from the feature set for the model.

plt.figure(figsize=(5,5))
xlabel = "Hemogloben Levels"
ylabel = "Packed Cell Value Counts"
sns.regplot(x=df["hemo"], y=df["pcv"])
plt.title('Hemoglobin vs Packed Cell Value Counts')
plt.show()
```



[139]:
```
#Does age factor into the equation?

plt.figure(figsize=(5,5))
xlabel = "Hemoglobin Levels"
ylabel = "Age"
sns.regplot(x=df["hemo"], y=df["age"])
plt.title('Hemoglobin vs Age')
plt.show()
```



[140]:
```
# Appears to have a linear relationship
# As rbcc counts increase, serum creatinine decrease
# makes sense because higher creatinine levels are associated with
# lower hemoglobin counts which indicates anemia
plt.figure(figsize=(5,5))
xlabel = "Red Blood Cell Counts"
ylabel = "Serum Creatinine"
sns.regplot(x=df["rbcc"], y=df["sc"])
plt.title('Red Blood Cell Counts vs Serum Creatinine')
plt.show()
```

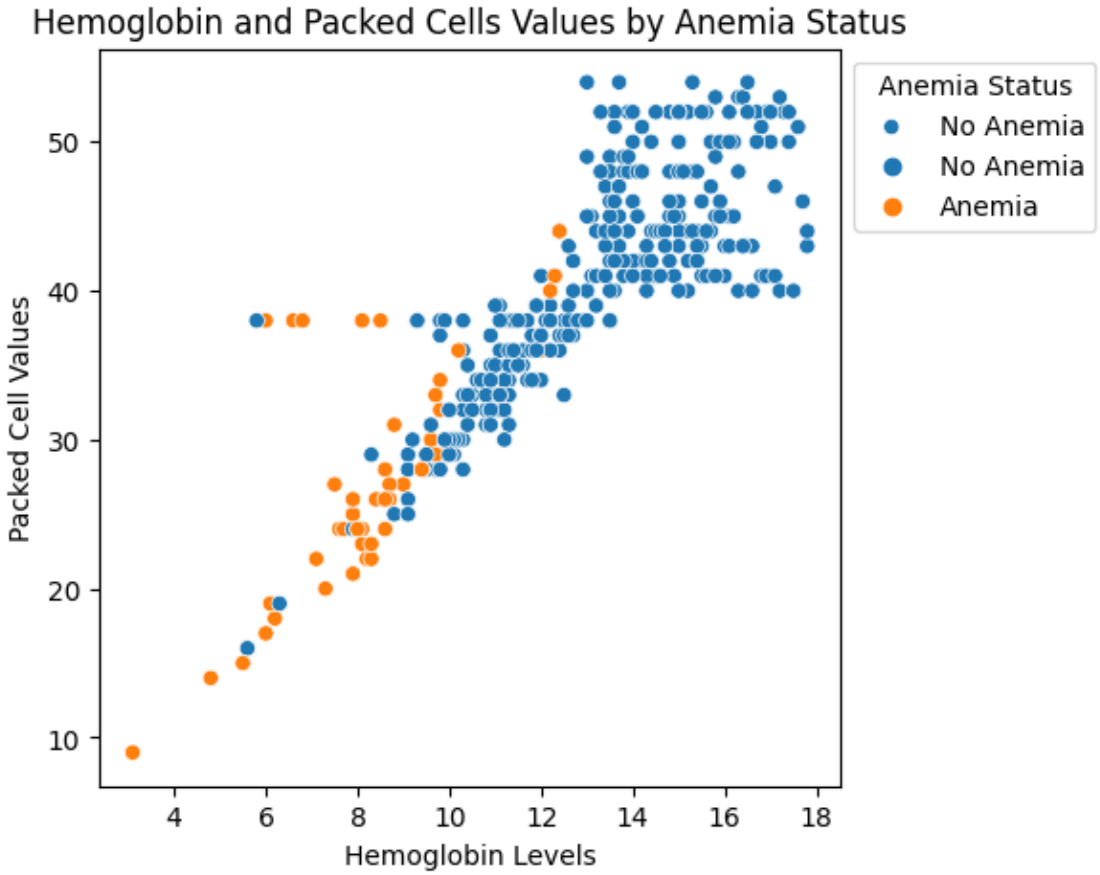


[52]:
```
# Analysis the data using Scatterplots to compare features
```

[142]:
```
#Create a scatter plot function using seaborn
def sns_scat_plt(X,Y,H, xlab, ylab, Title):
    plt.figure(figsize=(5,5))
    ax = sns.scatterplot(x=df[X],y=df[Y],hue=df[H])
    plt.xlabel(xlab)
    plt.ylabel(ylab)
    plt.title(Title)
    plt.legend()
    return (ax)
```

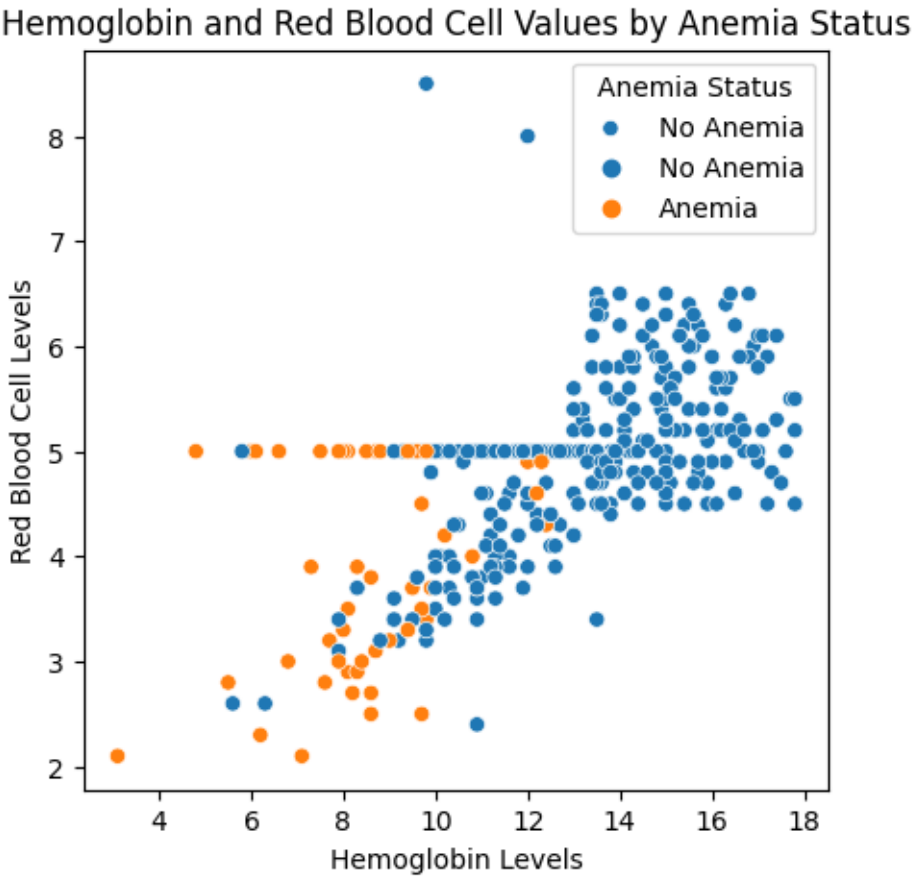
[145]:
```
#Relationship between hemoglobin counts and Packed Cell Volumes by Anemia Status
# Expect the lower the hemoglobin and packed cell values to indicate anemia
X='hemo'
Y='pcv'
H='ane'
xlab='Hemoglobin Levels'
ylab='Packed Cell Values'
```

```
Title = 'Hemoglobin and Packed Cells Values by Anemia Status'
ax = sns_scat_plt(X,Y,H, xlab, ylab, Title)
ax.legend(title='Anemia Status', bbox_to_anchor=(1, 1),labels=['No Anemia', 'No␣
 ↪Anemia', 'Anemia'] )
plt.show()
```

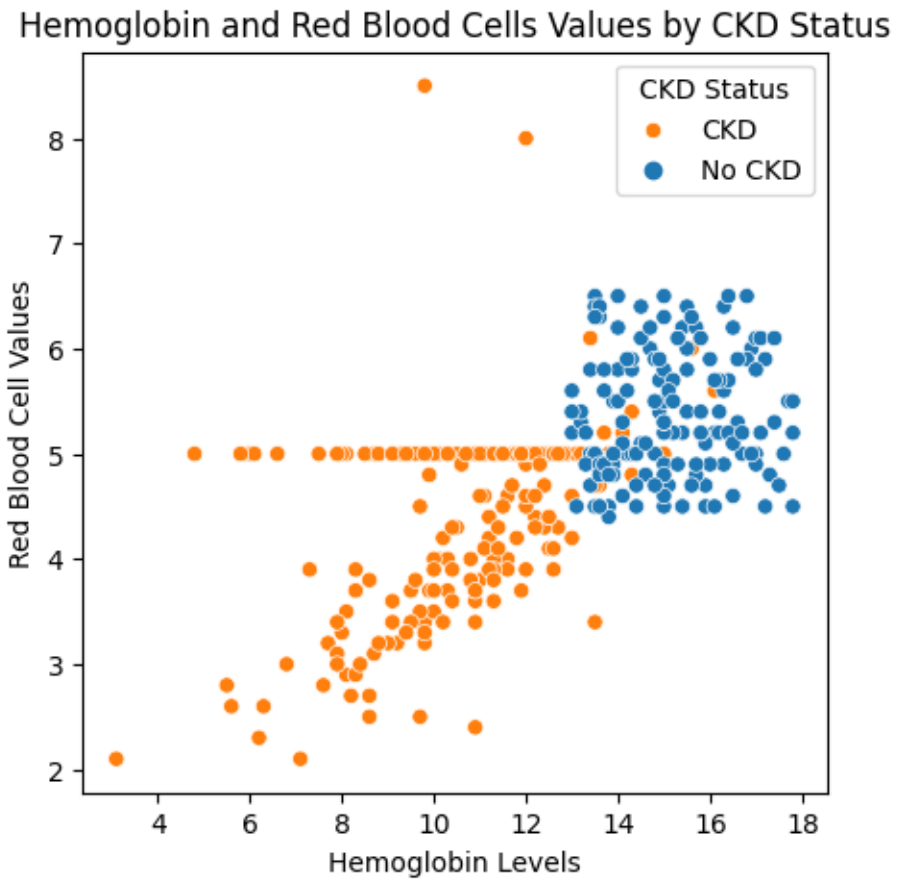


[146]:
```
#Relationship between hemoglobin counts and Red Blood cell  Volumes by Anemia␣
 ↪Status
# Expect the lower the hemoglobin and Red Blood cell values to indicate anemia

X='hemo'
Y='rbcc'
H='ane'
xlab='Hemoglobin Levels'
ylab='Red Blood Cell Levels'
Title = 'Hemoglobin and Red Blood Cell Values by Anemia Status'
ax = sns_scat_plt(X,Y,H, xlab, ylab, Title)
ax.legend(title='Anemia Status', bbox_to_anchor=(1, 1), labels=['No Anemia',␣
 ↪'No Anemia','Anemia'])
```

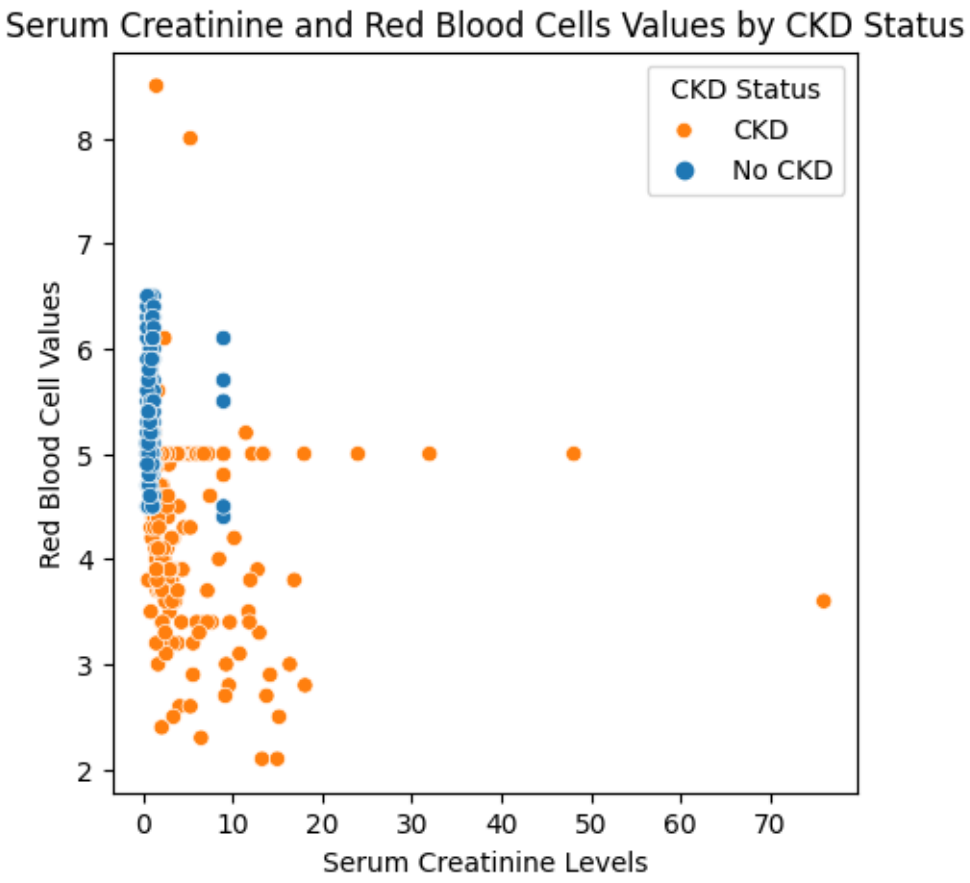
```
plt.show()
```



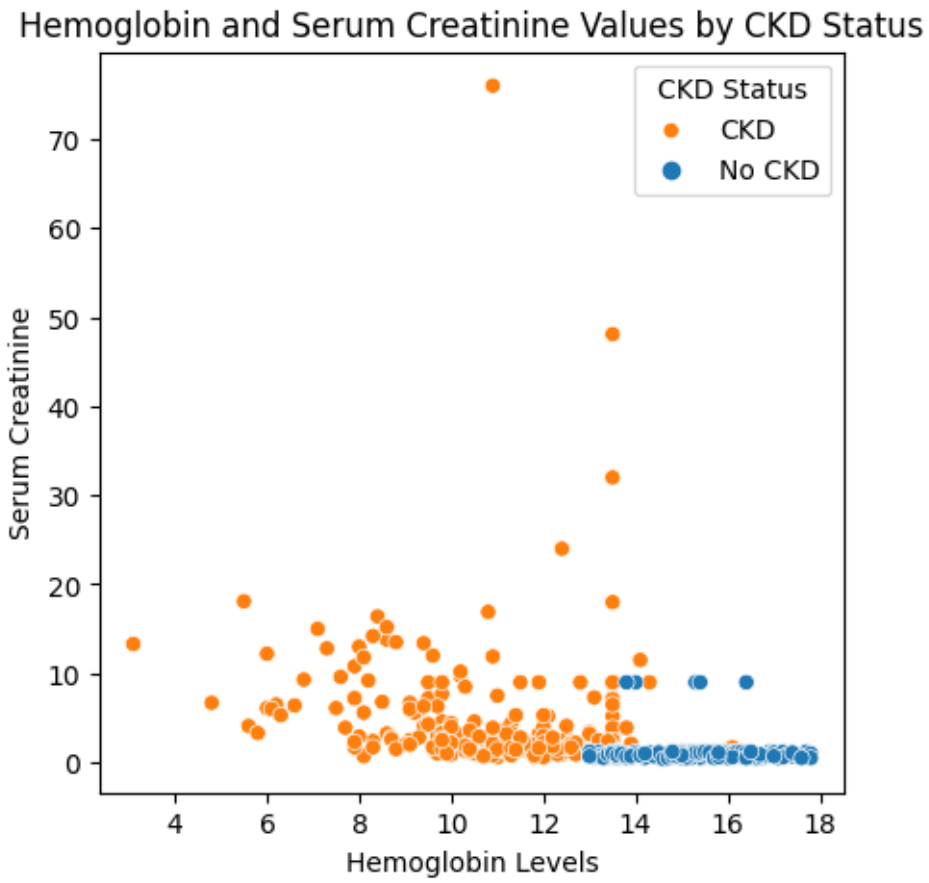
[147]:
```
# Relationship between hemoglobin counts and Red Blood Cell Values by CKD Status
# With CKD comes anemia, expect lower hemoglobin and red blood cell values.
X='hemo'
Y='rbcc'
H='class'
xlab='Hemoglobin Levels'
ylab='Red Blood Cell Values'
Title = 'Hemoglobin and Red Blood Cells Values by CKD Status'

ax = sns_scat_plt(X,Y,H, xlab, ylab, Title)
ax.legend(title='CKD Status', bbox_to_anchor=(1, 1), labels=['CKD', 'No CKD'])
plt.show()
```



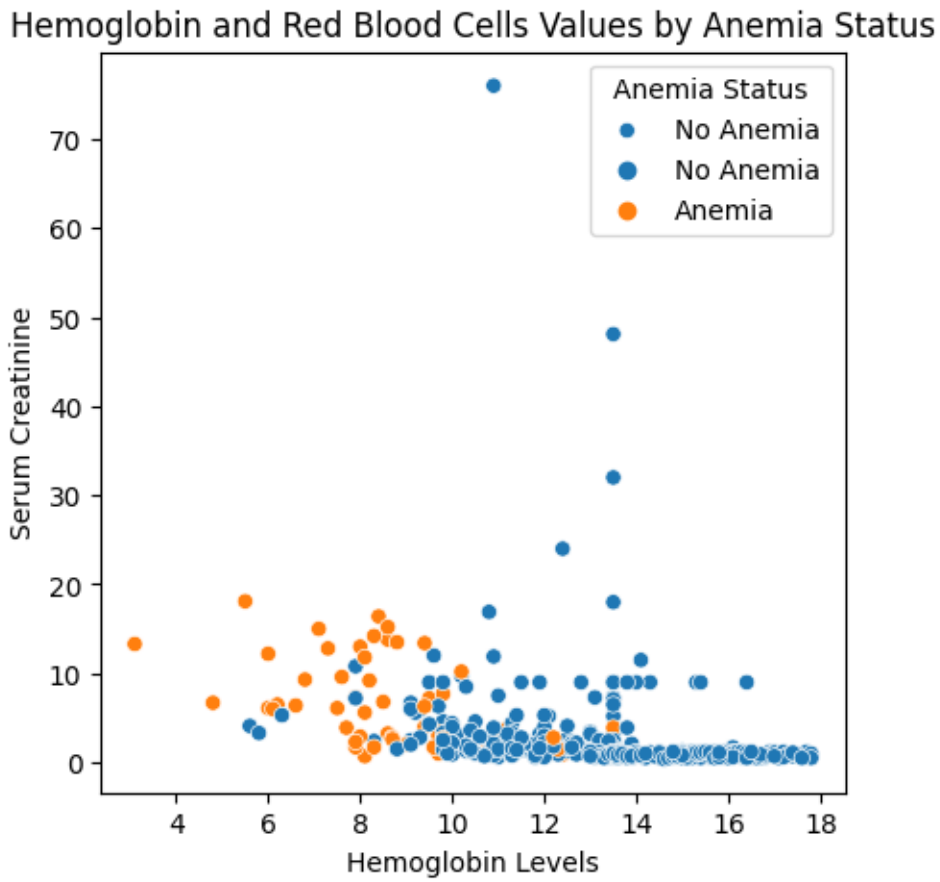
[281]:
```
# Relationship between Serum Creatinine and Red Blood Cell Values by CKD Status
# With CKD comes anemia, expect lower hemoglobin and red blood cell values.
X='sc'
Y='rbcc'
H='class'
xlab='Serum Creatinine Levels'
ylab='Red Blood Cell Values'
Title = 'Serum Creatinine and Red Blood Cells Values by CKD Status'

ax = sns_scat_plt(X,Y,H, xlab, ylab, Title)
ax.legend(title='CKD Status', bbox_to_anchor=(1, 1), labels=['CKD', 'No CKD'])
plt.show()
```



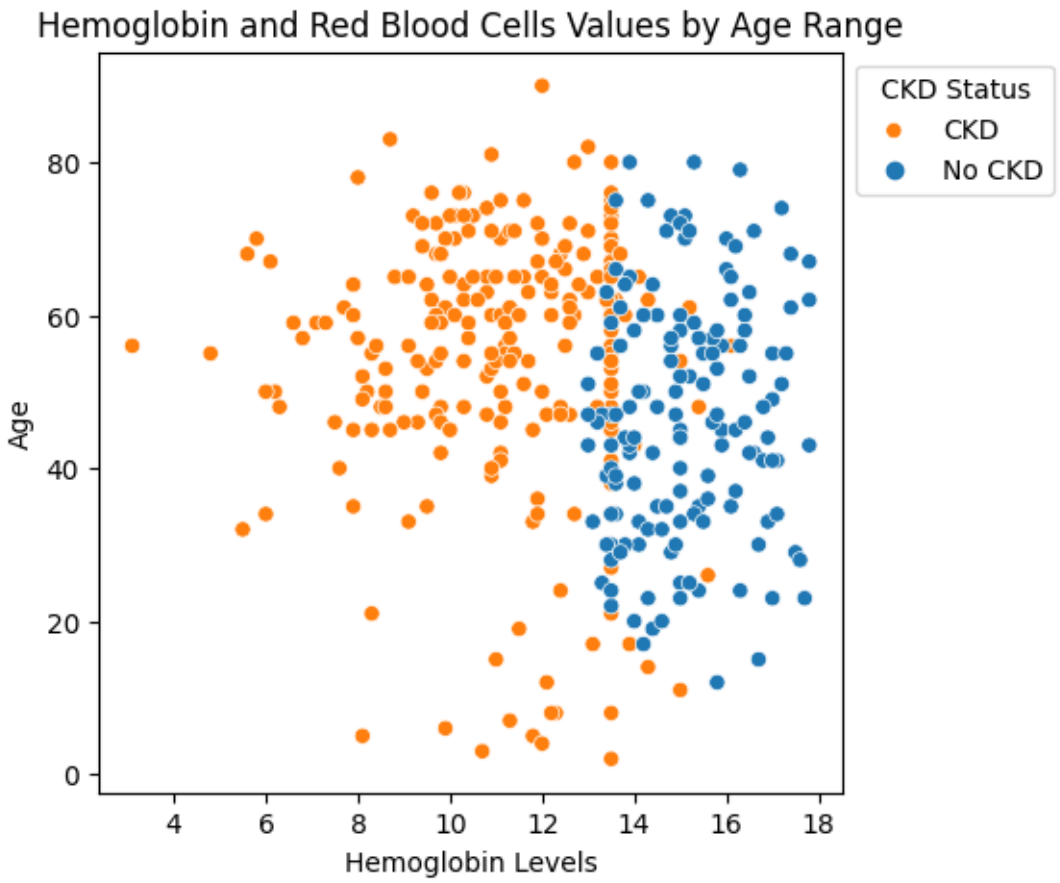
[148]:
```
# Relationship between Serum Creatinine and Red Blood Cell Values by CKD Status
# With CKD comes anemia, expect lower hemoglobin and red blood cell values.
X='hemo'
Y='sc'
H='class'
xlab='Hemoglobin Levels'
ylab='Serum Creatinine'
Title = 'Hemoglobin and Serum Creatinine Values by CKD Status'
ax = sns_scat_plt(X,Y,H, xlab, ylab, Title)
ax.legend(title='CKD Status', bbox_to_anchor=(1, 1), labels=['CKD', 'No CKD'])
plt.show()
```



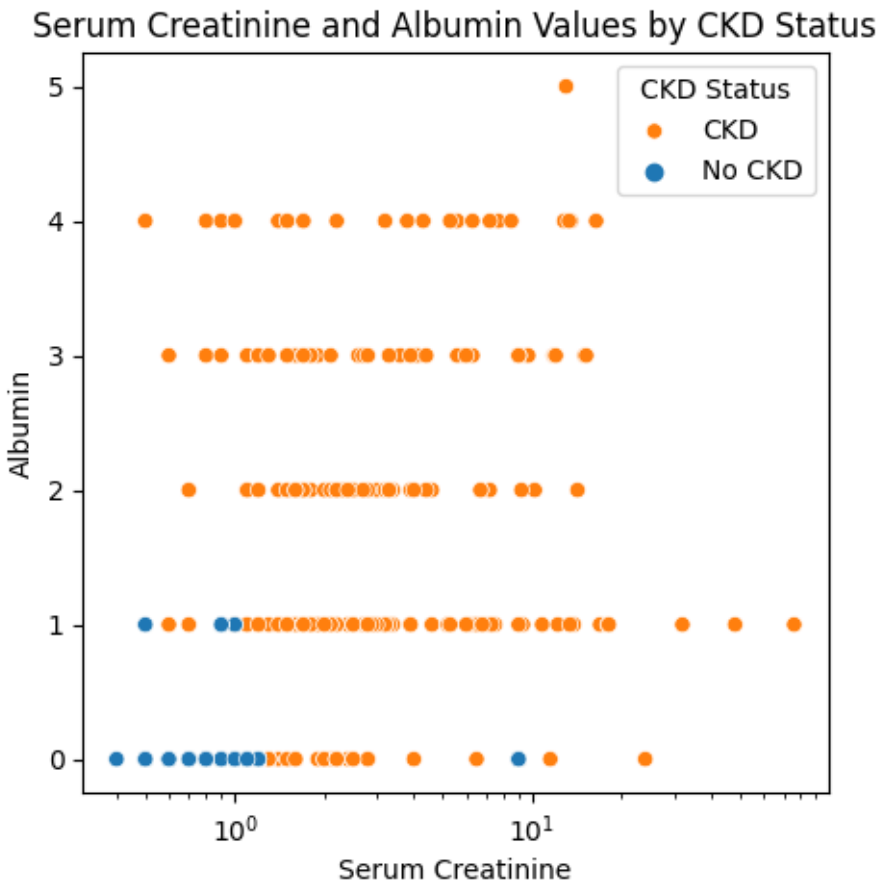
[149]:
```
# Relationship between Serum Creatinine and Red Blood Cell Values by CKD Status
# With CKD comes anemia, expect lower hemoglobin and red blood cell values.
X='hemo'
Y='sc'
H='ane'
xlab='Hemoglobin Levels'
ylab='Serum Creatinine'
Title = 'Hemoglobin and Red Blood Cells Values by Anemia Status'
ax = sns_scat_plt(X,Y,H, xlab, ylab, Title)
ax.legend(title='Anemia Status', bbox_to_anchor=(1, 1), labels=['No Anemia',␣
 ↪'No Anemia','Anemia'])
plt.show()
```



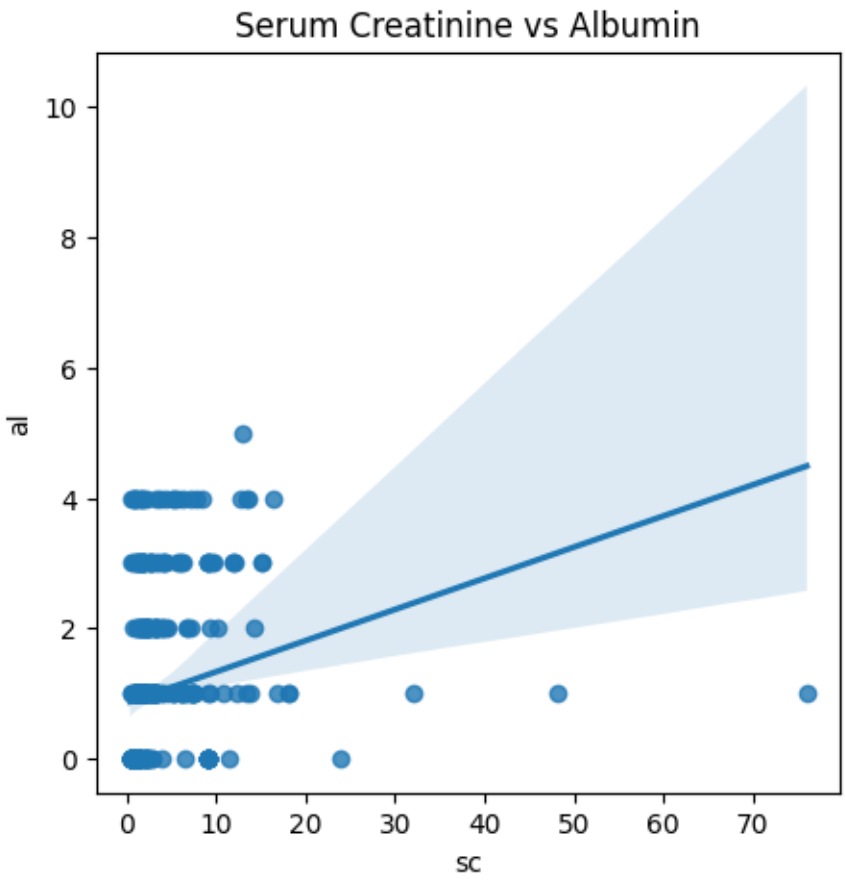
[150]:
```
# Check the relationships by age
# Relationship between age and anemia (low hemoglobin values and low rbcc)


X='hemo'
Y='age'
H='class'
xlab='Hemoglobin Levels'
ylab='Age'
Title = 'Hemoglobin and Red Blood Cells Values by Age Range'
ax = sns_scat_plt(X,Y,H, xlab, ylab, Title)
ax.legend(title='CKD Status', bbox_to_anchor=(1, 1), labels=['CKD', 'No CKD'])
plt.show()
```



[158]:
```
# Check the relationships by albumin
# Relationship between albumin and serum creatinine


X='sc'
Y='al'
H='class'
xlab='Serum Creatinine'
ylab='Albumin'
Title = 'Serum Creatinine and Albumin Values by CKD Status'
ax = sns_scat_plt(X,Y,H, xlab, ylab, Title)
ax.legend(title='CKD Status', bbox_to_anchor=(1, 1), labels=['CKD', 'No CKD'])
ax.set_xscale("log")
plt.show()
```

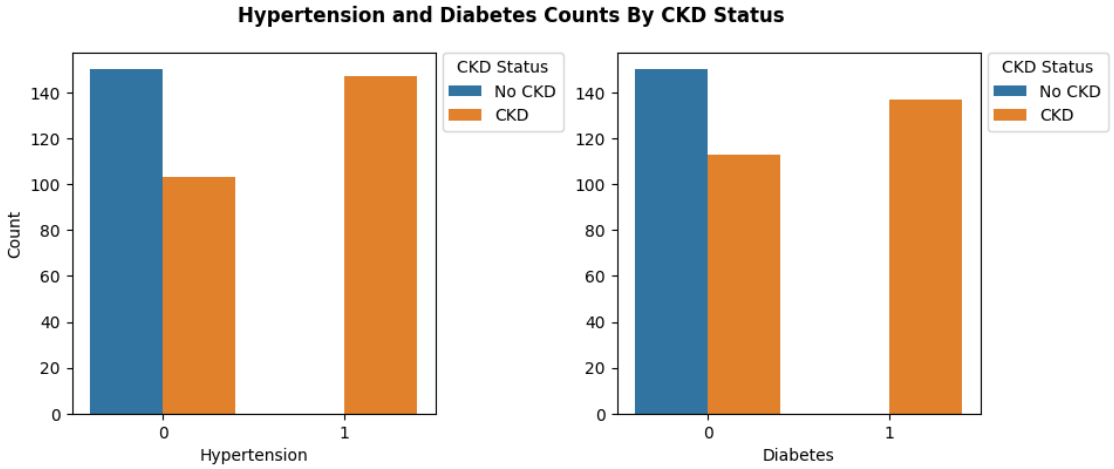


[161]:
```
plt.figure(figsize=(5,5))
xlabel = "Serum Creatinine"
ylabel = "Albumin"
sns.regplot(x=df["sc"], y=df["al"])
plt.title('Serum Creatinine vs Albumin')
plt.show()
```



[162]:
```
fig = plt.figure(figsize=(10,4))
title = fig.suptitle('Hypertension and Diabetes Counts By CKD Status',␣
 ↪fontsize=12, fontweight="bold")

fig.subplots_adjust(top=0.88, wspace=0.5)

# Countplot 1
ax1 = ax1 = fig.add_subplot(1,2,1)


ax1 = sns.countplot(x='htn',hue='class',data=df)
ax1.set_xlabel('Hypertension')
ax1.set_ylabel("Count")
plt.legend(title='CKD Status', labels=['No CKD', 'CKD'],bbox_to_anchor=(1.02,␣
 ↪1), loc='upper left', borderaxespad=0)

# Countplot 2
ax2 = ax2 = fig.add_subplot(1,2,2)
```

```
ax2 = sns.countplot(x='dm',hue='class',data=df)
ax2.set_xlabel('Diabetes')
ax2.set_ylabel("")
plt.legend(title='CKD Status', labels=['No CKD', 'CKD'],bbox_to_anchor=(1.02,␣
 ↪1), loc='upper left', borderaxespad=0)
plt.show()
```



3 Feature Selection

[167]:
```
#Create features and target sets
# Dependent variable is the CKD status or 'class'
# y is the class column, last column in the dataset
```

[168]:
```
X = df.iloc[: , :-1]  #features
y = df.iloc[:,-1:]    #target
```

[169]:
```
# Normalize the dataset
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_scale = scaler.fit_transform(X)
y_scale = scaler.fit_transform(y)
```

[170]:
```
# Create the training and test datasets
# Train Test Split
from sklearn.model_selection import train_test_split
```

[171]:
```
X_train, X_test, y_train, y_test = train_test_split(X_scale, y_scale, test_size␣
 ↪= 0.2, random_state = 0)
```

[172]:
```
# Only 400 obs in df, check the shape of X_train and X_test
X_train.shape, X_test.shape
```

[172]:
```
((320, 23), (80, 23))
```

[173]:
```
#Conduct Feature Engineering
```

[174]:
```
#Use RFE from SKlearn
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
```

[175]:
```
# Build model
rf = RandomForestRegressor()
RFE(estimator=LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1,␣
 ↪normalize=False),
  n_features_to_select=None, step=1, verbose=0)

#Fit the model
rf.fit(X_train,y_train.ravel())

#Check the score using R-squared
rf.score(X_test, y_test.ravel())
```

[175]:
```
0.9285989010989011
```

[176]:
```
# View the computed coefficients by viewing the .feature_importances object
#
rf_df = pd.DataFrame(
    zip(X.columns, abs(rf.feature_importances_)),
    columns=["feature", "weight"],
    ).sort_values("weight").reset_index(drop=True)

rf_df
```

[176]:

	feature	weight
0	ane	0.000000
1	cad	0.000000
2	ba	0.000000
3	pc	0.000118
4	su	0.000519
5	wbcc	0.001393
6	rbc	0.001446
7	age	0.001632
8	pe	0.002120
9	bp	0.002197
10	appet	0.004363

```
11     bgr  0.008341
12    rbcc  0.013321
13      bu  0.014456
14      dm  0.014624
15     pot  0.014970
16     htn  0.015105
17     sod  0.016455
18      sc  0.017323
19      al  0.050762
20      sg  0.173735
21    hemo  0.254775
22     pcv  0.392347
```

[177]:
```
# Feature Selection using RFECV
```

[178]:
```
from sklearn.feature_selection import RFECV
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import StratifiedKFold
```

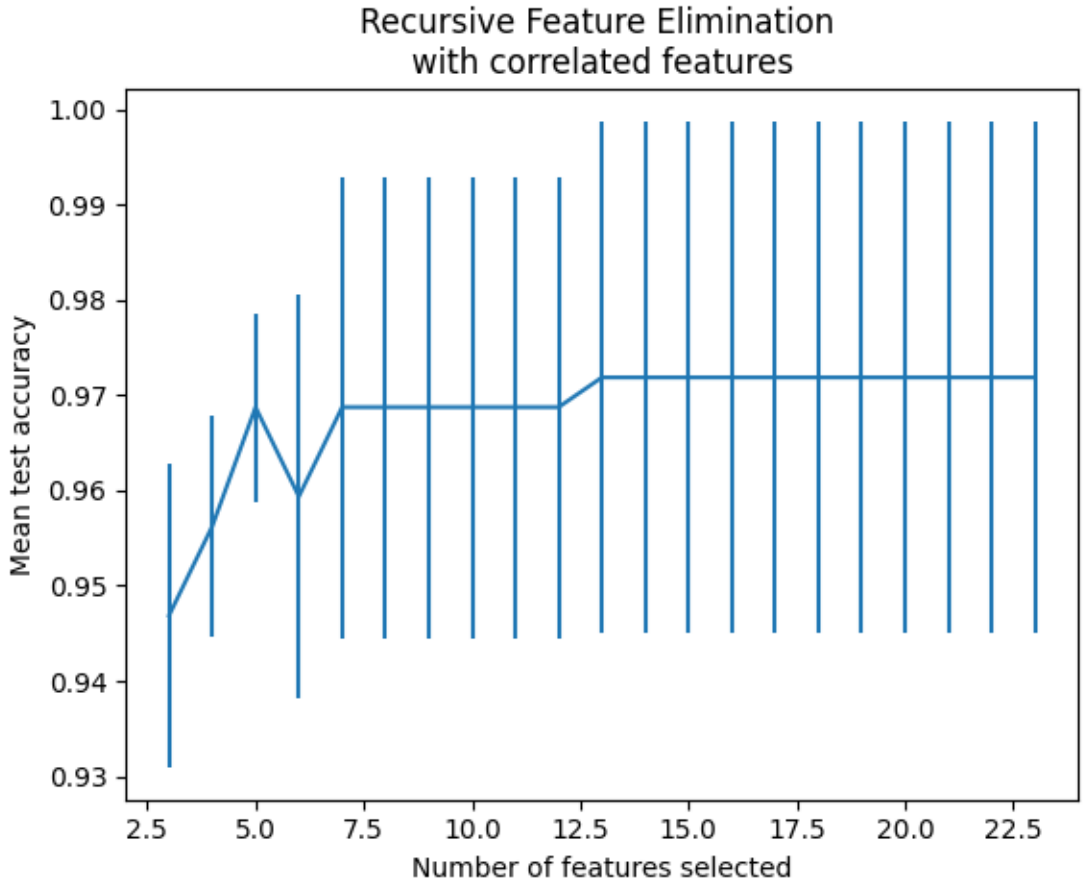
[179]:
```
#Build model and fit the model
```

[180]:
```
min_features_to_select = 3

clf = LogisticRegression()
cv = StratifiedKFold(5)

rfecv = RFECV(
    estimator=clf,
    step=1,
    cv=cv,
    scoring="accuracy",
    min_features_to_select=min_features_to_select,
    n_jobs=2,
)
rfecv.fit(X_train, y_train.ravel())

print(f"Optimal number of features: {rfecv.n_features_}")
```

```
Optimal number of features: 13
```

[181]:
```
#Plot the features vs cross-validation scores

n_scores = len(rfecv.cv_results_["mean_test_score"])
plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Mean test accuracy")
plt.errorbar(
```

```
    range(min_features_to_select, n_scores + min_features_to_select),
    rfecv.cv_results_["mean_test_score"],
    yerr=rfecv.cv_results_["std_test_score"],
)
plt.title("Recursive Feature Elimination \nwith correlated features")
plt.show()
```



[182]:
```
# View the selected features along with the weights from the RFE model
#
rfecv_df = pd.DataFrame(
    zip(X.columns, rfecv.support_, rf_df['weight']),
    columns=["feature", "mask", 'rf weights'],
    ).sort_values("rf weights").reset_index(drop=True)

rfecv_df
```

[182]:

	feature	mask	rf weights
0	age	False	0.000000
1	bp	True	0.000000

```
2        sg   True    0.000000
3        al   True    0.000118
4        su  False    0.000519
5       rbc   True    0.001393
6        pc   True    0.001446
7        ba  False    0.001632
8       bgr   True    0.002120
9        bu  False    0.002197
10       sc  False    0.004363
11      sod  False    0.008341
12      pot  False    0.013321
13     hemo   True    0.014456
14      pcv   True    0.014624
15     wbcc  False    0.014970
16     rbcc   True    0.015105
17      htn   True    0.016455
18       dm   True    0.017323
19      cad  False    0.050762
20    appet   True    0.173735
21       pe   True    0.254775
22      ane  False    0.392347
```

[183]:
```
# Create the feature dataset from the feature estimator model
# to input to the prediction models
```

[184]:
```
#Copy the old df with all the columns from both models
col = rfecv_df['feature'].to_list()
```

[185]:
```
new_df = df[col]
```

[186]:
```
#add the class column
new_df['class'] = df['class']
```

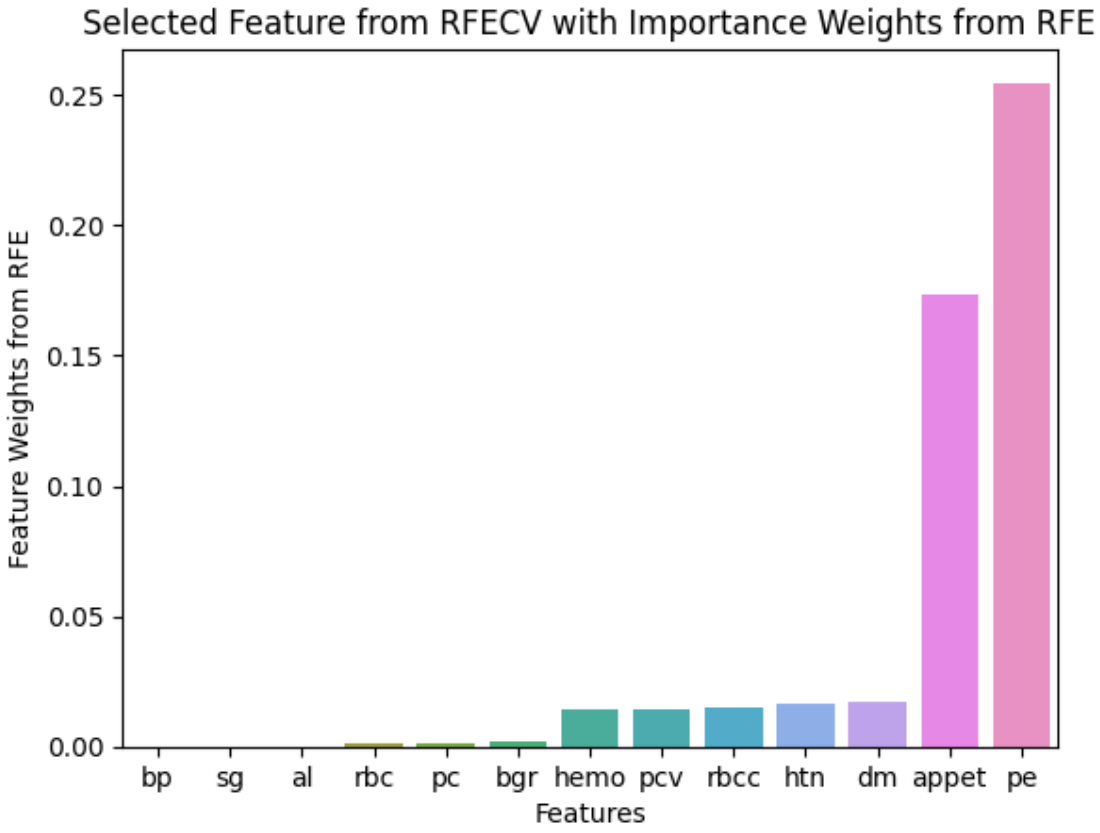
[187]:
```
# select only the feature columns selected by RFECV
rfecv_cols = rfecv_df[rfecv_df['mask'] == True]
```

[188]:
```
rfecv_cols
```

[188]:
```
   feature  mask  rf weights
1       bp  True    0.000000
2       sg  True    0.000000
3       al  True    0.000118
5      rbc  True    0.001393
6       pc  True    0.001446
8      bgr  True    0.002120
13    hemo  True    0.014456
14     pcv  True    0.014624
```

```
16    rbcc  True    0.015105
17     htn  True    0.016455
18      dm  True    0.017323
20   appet  True    0.173735
21      pe  True    0.254775
```

[189]:
```
#Plot the selected features by weight
#Use a barplot

sns.barplot(x=rfecv_cols['feature'], y = rfecv_cols['rf weights'])
plt.title('Selected Feature from RFECV with Importance Weights from RFE')
plt.xlabel('Features')
plt.ylabel('Feature Weights from RFE')
plt.show()
```



[ ]:

[190]:
```
#add class to the list of best features
best_feature_lst = rfecv_cols['feature'].to_list()
best_feature_lst.append('class')
```

[191]:
```
feature_df = new_df[best_feature_lst]
```

[192]:
```
# df for prediction models
feature_df
```

[192]:

	bp	sg	al	rbc	pc	bgr	hemo	pcv	rbcc	htn	dm	appet	pe	class
0	80	1.020	1	0	0	121	15.4	44	5.2	1	1	0	0	1
1	50	1.020	4	0	0	150	11.3	38	5.0	0	0	0	0	1
2	80	1.010	2	0	0	423	9.6	31	5.0	0	1	1	0	1
3	70	1.005	4	0	1	117	11.2	32	3.9	1	0	1	1	1
4	80	1.010	2	0	0	106	11.6	35	4.6	0	0	0	0	1
..	..	…	..	…	..	…	…	…	…	..	…	..	…	…	
395	80	1.020	0	0	0	140	15.7	47	4.9	0	0	0	0	0
396	70	1.025	0	0	0	75	16.5	54	6.2	0	0	0	0	0
397	80	1.020	0	0	0	100	15.8	49	5.4	0	0	0	0	0
398	60	1.025	0	0	0	114	14.2	51	5.9	0	0	0	0	0
399	80	1.025	0	0	0	131	15.8	53	6.1	0	0	0	0	0

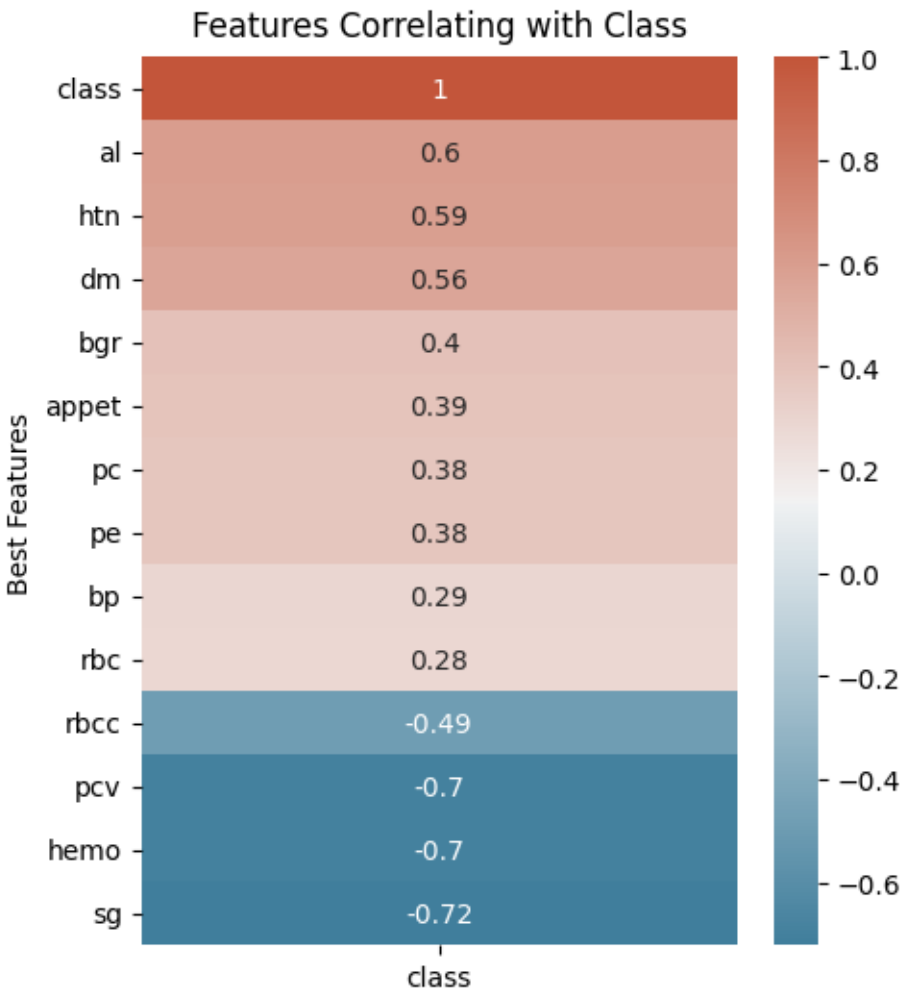
[400 rows x 14 columns]

[ ]:
```
#Add the serum-column column for an additional test
```

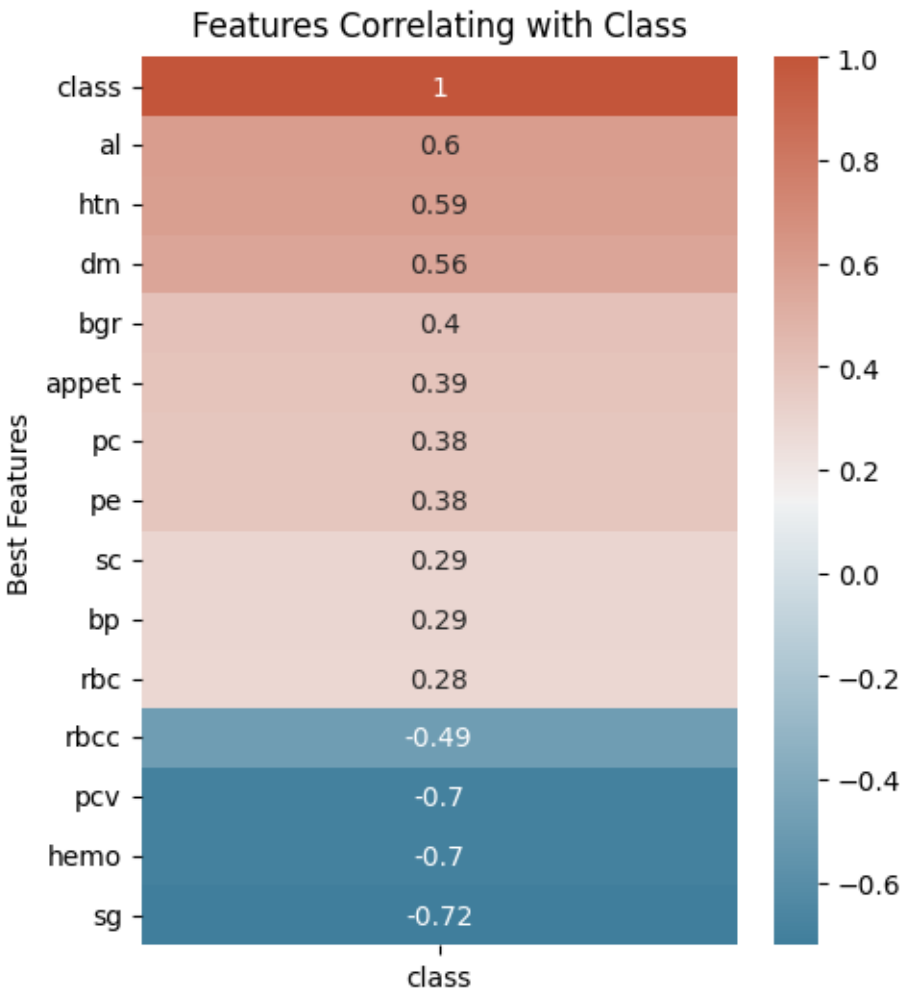
[296]:
```
new_test_data = new_df[best_feature_lst].copy()
```

[297]:
```
new_test_data['sc'] = new_df['sc'].copy()
```

[193]:
```
# Plot the correlation with CKD Status aka Class variable
# Compute the correlation matrix
corr = feature_df.corr()

plt.figure(figsize=(5, 6))

ax = sns.heatmap(feature_df.corr()[['class']].sort_values(by='class',␣
 ↪ascending=False), annot=True, cmap=cmap)
ax.set_title('Features Correlating with Class', fontdict={'fontsize':12}, pad=8)
plt.yticks(rotation=0)
plt.ylabel('Best Features')
plt.show()
```



[300]:
```
# Plot the correlation with CKD Status using the new_test_set with the sc values
# Compute the correlation matrix
corr = new_test_data.corr()

plt.figure(figsize=(5, 6))

ax = sns.heatmap(new_test_data.corr()[['class']].sort_values(by='class',␣
 ↪ascending=False), annot=True, cmap=cmap)
ax.set_title('Features Correlating with Class', fontdict={'fontsize':12}, pad=8)
plt.yticks(rotation=0)
plt.ylabel('Best Features')
plt.show()
```



4 Prepare the Training, Test, and Validation Sets

[194]:
```
# Read in dataset as validation set
val_df = pd.read_csv('val_set.csv', usecols= best_feature_lst, header=0)
```

[195]:
```
val_df.head()
```

[195]:

	bp	sg	al	rbc	pc	bgr	hemo	pcv	rbcc	htn	dm	appet	pe	\
0	80.0	1.025	0.0	0.0	0.0	85.0	15.6	44	6.3	0.0	0.0	1.0	0.0	
1	70.0	1.015	3.0	0.0	0.0	253.0	10.9	31	3.4	1.0	1.0	0.0	1.0	
2	80.0	1.025	0.0	0.0	0.0	119.0	13.9	49	5.0	0.0	0.0	1.0	0.0	
3	80.0	1.020	0.0	0.0	0.0	99.0	13.6	44	6.4	0.0	0.0	1.0	0.0	
4	80.0	1.025	0.0	0.0	0.0	118.0	14.8	45	4.7	0.0	0.0	1.0	0.0	

class

```
0    0.0
1    1.0
2    0.0
3    0.0
4    0.0
```

[196]:
```
# sample the validation set
val_set = val_df.sample(n = 60)
```

[303]:
```
X2 = feature_df.iloc[: , :-1]  #features
y2 = feature_df.iloc[:,-1:]    #target
val_set_test = val_set.iloc[:,:-1] #validation set
val_set_class = val_set.iloc[:,-1:] #validation class column
```

[198]:
```
#View the value counts for the test set
y2.value_counts()
```

[198]:
```
class
1        250
0        150
dtype: int64
```

[304]:
```
# Normalize the datasets

scaler = MinMaxScaler()
X_scale2 = scaler.fit_transform(X2)
y_scale2 = scaler.fit_transform(y2)
val_scale_test = scaler.fit_transform(val_set_test)
val_scale_class = scaler.fit_transform(val_set_class)
```

[200]:
```
#Create training and test sets
X_train2, X_test2, y_train2, y_test2 = train_test_split(X_scale2, y_scale2,␣
 ↪test_size = 0.2, random_state = 0)
```

[201]:
```
#View the obs per set

y_test2.ravel()
```

[201]:
```
array([1., 0., 0., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 0., 1., 1., 0.,
       1., 1., 1., 1., 0., 1., 0., 1., 0., 1., 1., 0., 1., 1., 0., 0., 1.,
       1., 1., 1., 1., 1., 0., 1., 0., 1., 1., 0., 1., 0., 0., 1., 1., 1.,
       1., 0., 1., 0., 0., 1., 1., 1., 1., 1., 0., 1., 0., 1., 0., 1., 1.,
       1., 0., 1., 0., 1., 1., 1., 1., 0., 0., 0., 0.])
```

5 Build the model the models

5.1 Random Forest

[202]:
```
from sklearn.ensemble import RandomForestClassifier
```

[203]:
```
# Use the Random Forest Classifier Model

# Create random forest classifier object
randomforest = RandomForestClassifier(random_state=0, n_jobs=-1,␣
 ↪class_weight="balanced")

#Train model
rf_model = randomforest.fit(X_train2, y_train2.ravel())
```

[204]:
```
# view model accuracy score
rf_model.score(X_test2, y_test2.ravel())
```

[204]: 0.9625

[205]:
```
# Random Forest appears to have a reasonable accuracy
#
```

[206]:
```
# Compute predictions

from sklearn.metrics import RocCurveDisplay, roc_auc_score,␣
 ↪average_precision_score, PrecisionRecallDisplay, ConfusionMatrixDisplay,\
                    precision_recall_curve, accuracy_score
```

[207]:
```
# compute predict probabilties on test set
# only need the probabilities for the positive class only

y_pred_probs = rf_model.predict_proba(X_test2)[:, 1]

# compute predictions on test set
y_preds = rf_model.predict(X_test2)
```

[208]:
```
#Check the accuracy
print(f'Accuracy:, {accuracy_score(y_test2.ravel(), y_preds)}')
```

```
Accuracy:, 0.9625
```

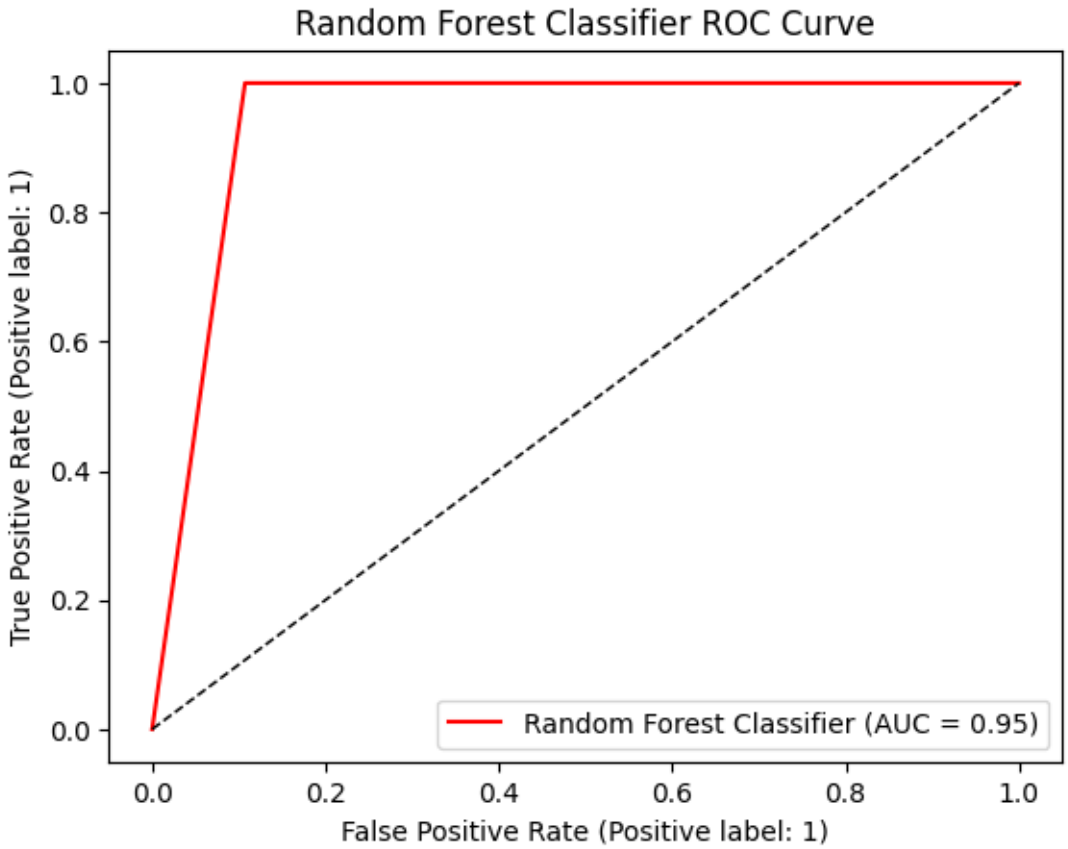
[209]:
```
# Compute AUC to validate accuracy
auc_score = roc_auc_score(y_test2.ravel(), y_pred_probs)
round(auc_score,4)
```

[209]: 0.999

[ ]:
```
rf_model_new.score(X_test2_new, y_test2_new.ravel())
```

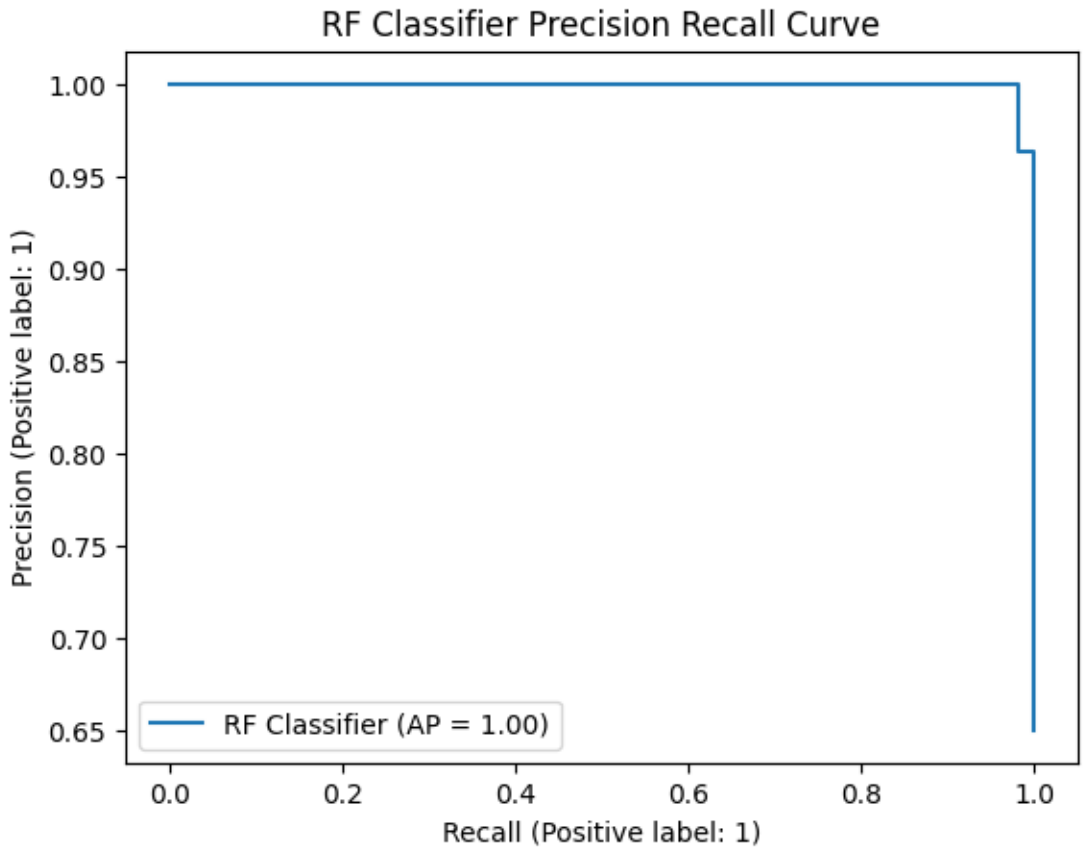
[210]:
```
# Plot the ROC Curve

def roc_curve_plt(test, preds, Name,Title):
    lw = 1
    RocCurveDisplay.from_predictions(test, preds,
                                     name = Name,
                                     color="red", pos_label=None)
    plt.plot([0, 1], [0, 1], 'k--', lw=lw)
    plt.title(Title)
    plt.show()
    return
```

[211]:
```
#Define parameters for ROC Curve plot
test = y_test2
preds = y_preds
Name = 'Random Forest Classifier'
Title = 'Random Forest Classifier ROC Curve'
roc_curve_plt(test, preds, Name,Title)
```



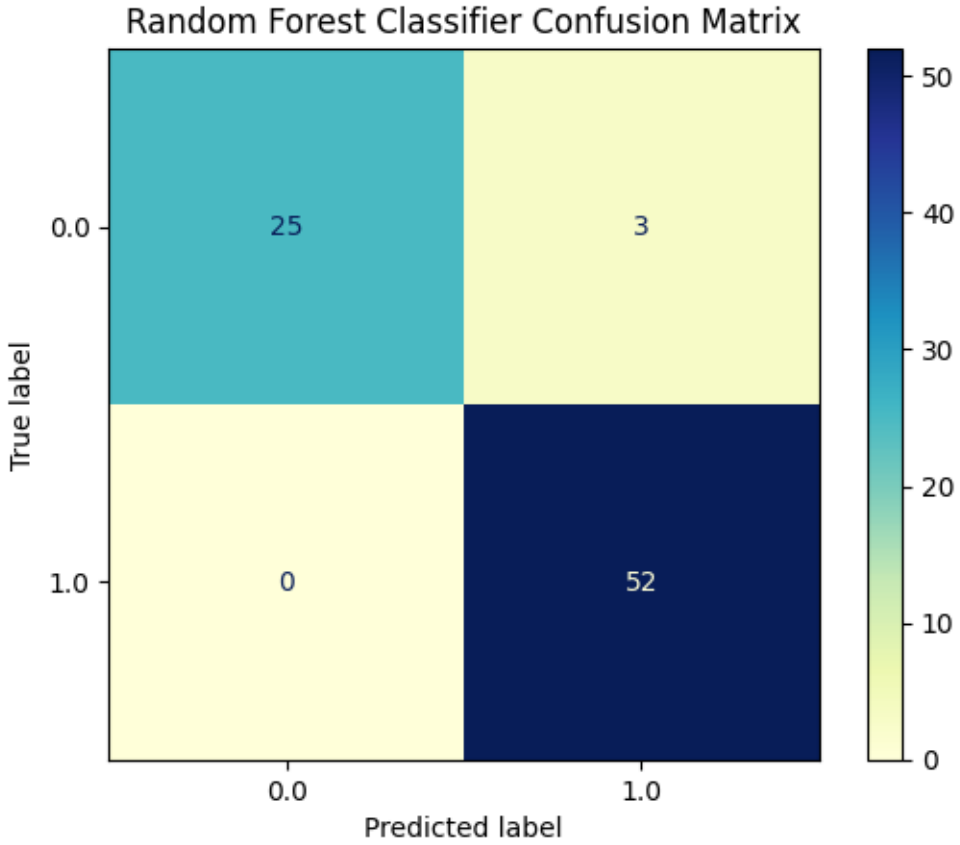
[212]:
```
# Compute average precision score
# Plot the precision recall curve

def prc_plt(test, pred_probs, Name, Title):
    PrecisionRecallDisplay.from_predictions(
         test, pred_probs, name = Name)
    plt.title(Title)
    plt.show()
    return
```

[213]:
```
#Define parameters for prc
test = y_test2
pred_probs = y_pred_probs
Name = "RF Classifier"
Title = "RF Classifier Precision Recall Curve"
prc_plt(test, pred_probs, Name, Title)
```



[216]:
```
# Create Confusion Matrix from predictions
# Plot predictions on the test set
from sklearn.metrics import confusion_matrix, classification_report
```

```
from sklearn.metrics import ConfusionMatrixDisplay

ConfusionMatrixDisplay.from_predictions(y_test2, y_preds, cmap= 'YlGnBu')

plt.title("Random Forest Classifier Confusion Matrix")
plt.show()
```



[217]:
```
# Build Classification report
print(classification_report(y_test2, y_preds))
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.89      0.94        28
         1.0       0.95      1.00      0.97        52

    accuracy                           0.96        80
   macro avg       0.97      0.95      0.96        80
weighted avg       0.96      0.96      0.96        80
```

5.2 Build Adaboost model

[ ]:

[218]:
```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import cross_val_score
```

[219]:
```
# Create adaboost tree classifier object
adaboost = AdaBoostClassifier(random_state=0)

# Train model on the important features data set
ab_model = adaboost.fit(X_train2, y_train2.ravel())
```

[220]:
```
# Cross-validate Adaboost model

cvScore = cross_val_score(ab_model, X_test2, y_test2.ravel(),␣
 ↪scoring="accuracy")
print("CV Scores:", cvScore)
print("CV Scores Mean:", cvScore.mean())
```

```
CV Scores: [0.875  0.875  1.     0.9375 1.    ]
CV Scores Mean: 0.9375
```

[224]:
```
# Tune hyperparameters with Adaboost
from sklearn.model_selection import GridSearchCV

# Define hyperparameter candidates
param_grid = {'n_estimators': [100, 200, 400, 600, 800, 1000],
              'learning_rate': [0.001, 0.01, 0.1, 0.2, 0.5]}

# Instatiate GridSearchCV model
gs_ab = GridSearchCV(AdaBoostClassifier(), param_grid = param_grid)

# Train model
gs_ab_model = gs_ab.fit(X_train2, y_train2.ravel())

# Cross-validate model
cvScore2 = cross_val_score(gs_ab_model, X_test2, y_test2.ravel(),␣
 ↪scoring="accuracy")
print("CV Scores:", cvScore2)
print("CV Scores Mean:", cvScore2.mean())
```

```
CV Scores: [0.9375 0.875  0.9375 0.9375 0.9375]
CV Scores Mean: 0.925
```

[222]:
```
# View best hyperparameters
gs_ab_model.best_params_
```

[222]:
```
{'learning_rate': 0.5, 'n_estimators': 100}
```

[225]:
```
# Compute predictions

# compute predict probabilties on test set
# only need the probabilities for the positive class only
y_pred_probs_ab = gs_ab_model.predict_proba(X_test2)[:, 1]

# compute predictions on test set
y_preds_ab = gs_ab_model.predict(X_test2)
```

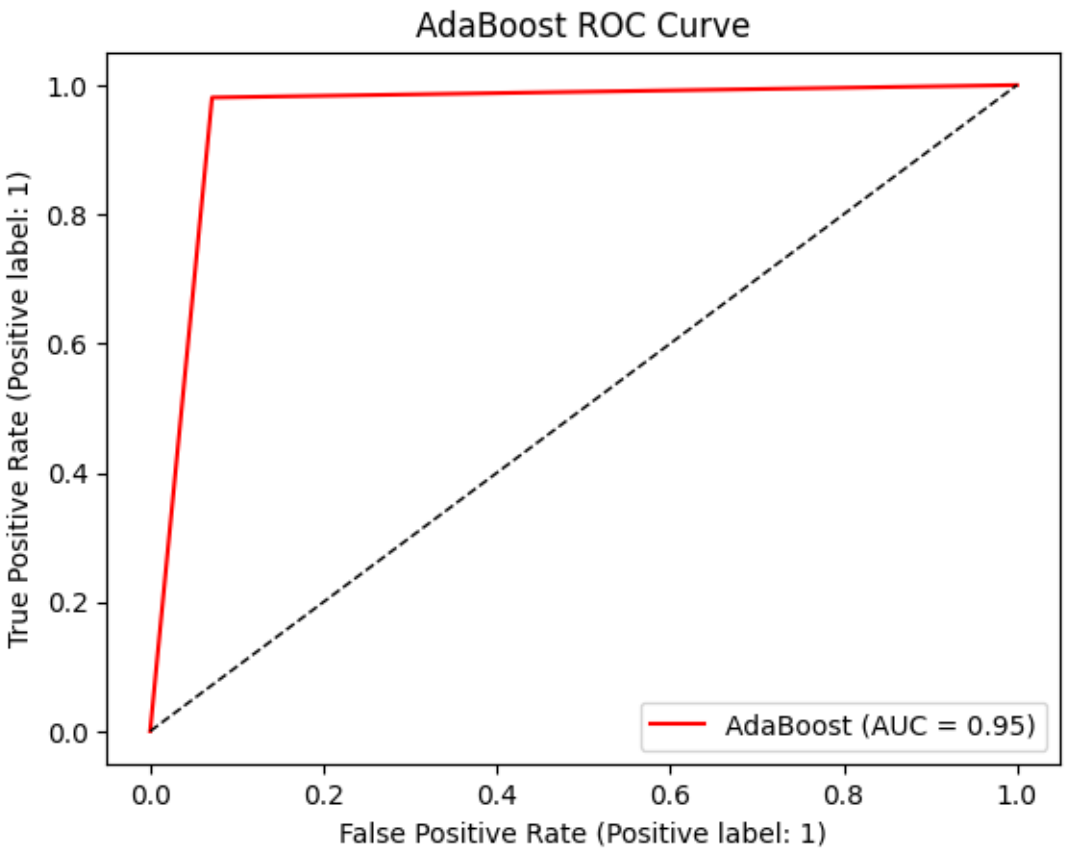
[226]:
```
# Compute AUC
auc_score = roc_auc_score(y_test2, y_pred_probs_ab)
round(auc_score,4)
```

[226]:
```
0.9979
```

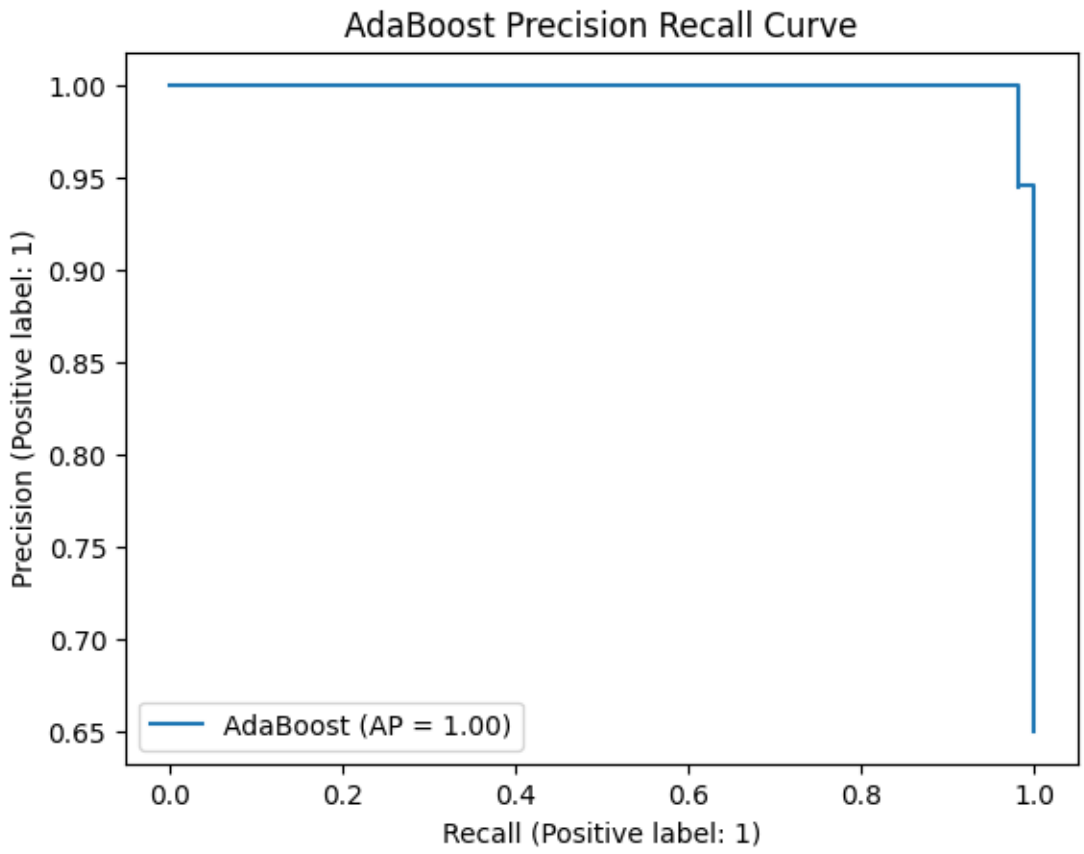
[227]:
```
#Check the accuracy
print(f'Accuracy:, {accuracy_score(y_test2, y_preds_ab)}')
```

```
Accuracy:, 0.9625
```

[228]:
```
#Define parameters for ROC Curve plot
test = y_test2
preds = y_preds_ab
Name = 'AdaBoost'
Title = 'AdaBoost ROC Curve'
roc_curve_plt(test, preds, Name,Title)
```



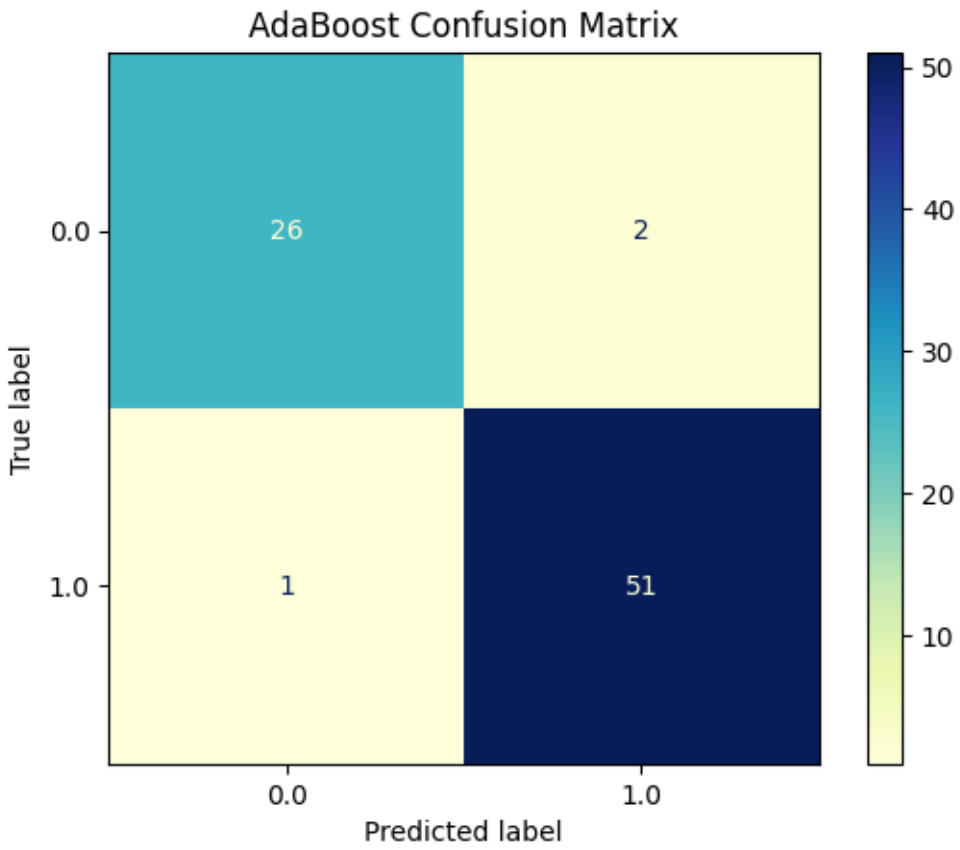
[229]:
```
#Define parameters for prc
test = y_test2
pred_probs = y_pred_probs_ab
Name = "AdaBoost"
Title = "AdaBoost Precision Recall Curve"
prc_plt(test, pred_probs, Name, Title)
```



[230]:
```
# Build Classification report
print(classification_report(y_test2, y_preds_ab))
```

```
              precision    recall  f1-score   support

         0.0       0.96      0.93      0.95        28
         1.0       0.96      0.98      0.97        52

    accuracy                           0.96        80
   macro avg       0.96      0.95      0.96        80
weighted avg       0.96      0.96      0.96        80
```

[231]:
```
# Create Confusion Matrix from predictions

ConfusionMatrixDisplay.from_predictions(y_test2, y_preds_ab, cmap= 'YlGnBu')

plt.title("AdaBoost Confusion Matrix")
plt.show()
```



5.3 XgBoost Model

[250]:
```
# xgboost model

import xgboost as xgb
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
```

[ ]:
```

```

[251]:
```
#Define model parameters for input to the GridSearch

params={'eta':[0.05,0.10,0.15,0.25],
        'max_depth':[3,4,5,6],
        'min_child_weight': [1,3,5],
        'gamma':[0.0,0.1,0.2,0.4],
        'colsample_bytree':[0.3,0.4,0.7]}
```

[252]:
```
#Build and fit the model with cross-validation.

xg = xgb.XGBClassifier()

grid = GridSearchCV(xg, params, n_jobs=4,
                    scoring="accuracy",
                    cv=3)

grid.fit(X_train2, y_train2)
```

[252]:
```
GridSearchCV(cv=3,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     callbacks=None, colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None,
                                     early_stopping_rounds=None,
                                     enable_categorical=False, eval_metric=None,
                                     feature_types=None, gamma=None,
                                     gpu_id=None, grow_policy=None,
                                     importance_type=None,
                                     interaction_constraints=None,
                                     learning_rate=None,…
                                     max_delta_step=None, max_depth=None,
                                     max_leaves=None, min_child_weight=None,
                                     missing=nan, monotone_constraints=None,
                                     n_estimators=100, n_jobs=None,
                                     num_parallel_tree=None, predictor=None,
                                     random_state=None, …),
             n_jobs=4,
             param_grid={'colsample_bytree': [0.3, 0.4, 0.7],
                         'eta': [0.05, 0.1, 0.15, 0.25],
                         'gamma': [0.0, 0.1, 0.2, 0.4],
                         'max_depth': [3, 4, 5, 6],
                         'min_child_weight': [1, 3, 5]},
             scoring='accuracy')
```

[253]:
```
grid.best_estimator_
```

[253]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.3, early_stopping_rounds=None,
              enable_categorical=False, eta=0.25, eval_metric=None,
              feature_types=None, gamma=0.4, gpu_id=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=6,
              max_leaves=None, min_child_weight=1, missing=nan,
```

```
                monotone_constraints=None, n_estimators=100, n_jobs=None,
                num_parallel_tree=None, predictor=None, …)
```

[254]:
```
grid.best_params_
```

[254]:
```
{'colsample_bytree': 0.3,
 'eta': 0.25,
 'gamma': 0.4,
 'max_depth': 6,
 'min_child_weight': 1}
```

[255]:
```
best_params = {'eta':[0.25],
        'max_depth':[6],
        'min_child_weight': [1,3,5],
        'gamma':[0.4],
        'colsample_bytree':[0.3,]}
```

[256]:
```
grid.n_features_in_
```

[256]:
```
13
```

[257]:
```
#rerun the model with the best parameters
xg = xgb.XGBClassifier()

grid = GridSearchCV(xg, best_params, n_jobs=4,
                    scoring="accuracy",
                    cv=3)

grid.fit(X_train2, y_train2)
```

[257]:
```
GridSearchCV(cv=3,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     callbacks=None, colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None,
                                     early_stopping_rounds=None,
                                     enable_categorical=False, eval_metric=None,
                                     feature_types=None, gamma=None,
                                     gpu_id=None, grow_policy=None,
                                     importance_type=None,
                                     interaction_constraints=None,
                                     learning_rate=None,…
                                     max_cat_to_onehot=None,
                                     max_delta_step=None, max_depth=None,
                                     max_leaves=None, min_child_weight=None,
                                     missing=nan, monotone_constraints=None,
                                     n_estimators=100, n_jobs=None,
```

```
                                num_parallel_tree=None, predictor=None,
                                random_state=None, …),
             n_jobs=4,
             param_grid={'colsample_bytree': [0.3], 'eta': [0.25],
                         'gamma': [0.4], 'max_depth': [6],
                         'min_child_weight': [1, 3, 5]},
             scoring='accuracy')
```

[258]:
```
# Score the training set
grid.best_score_
```

[258]:
```
0.9749309351672251
```

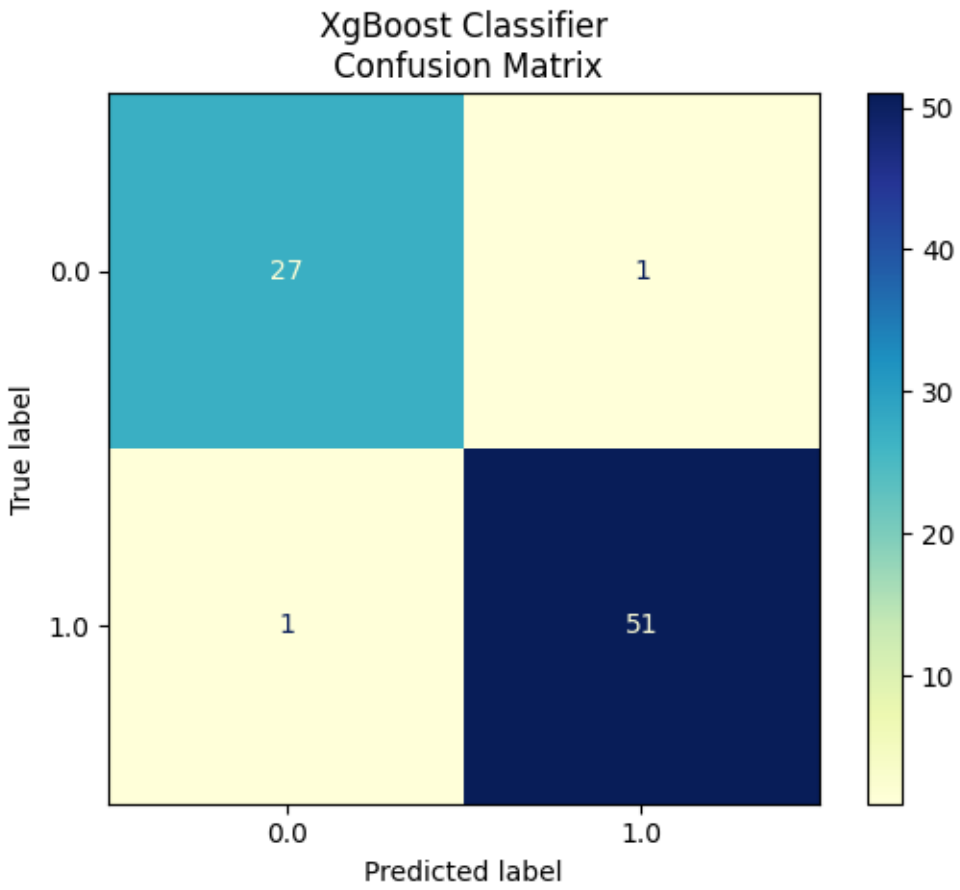
[259]:
```
#Score the test set
grid.score(X_test2, y_test2)
```

[259]:
```
0.975
```

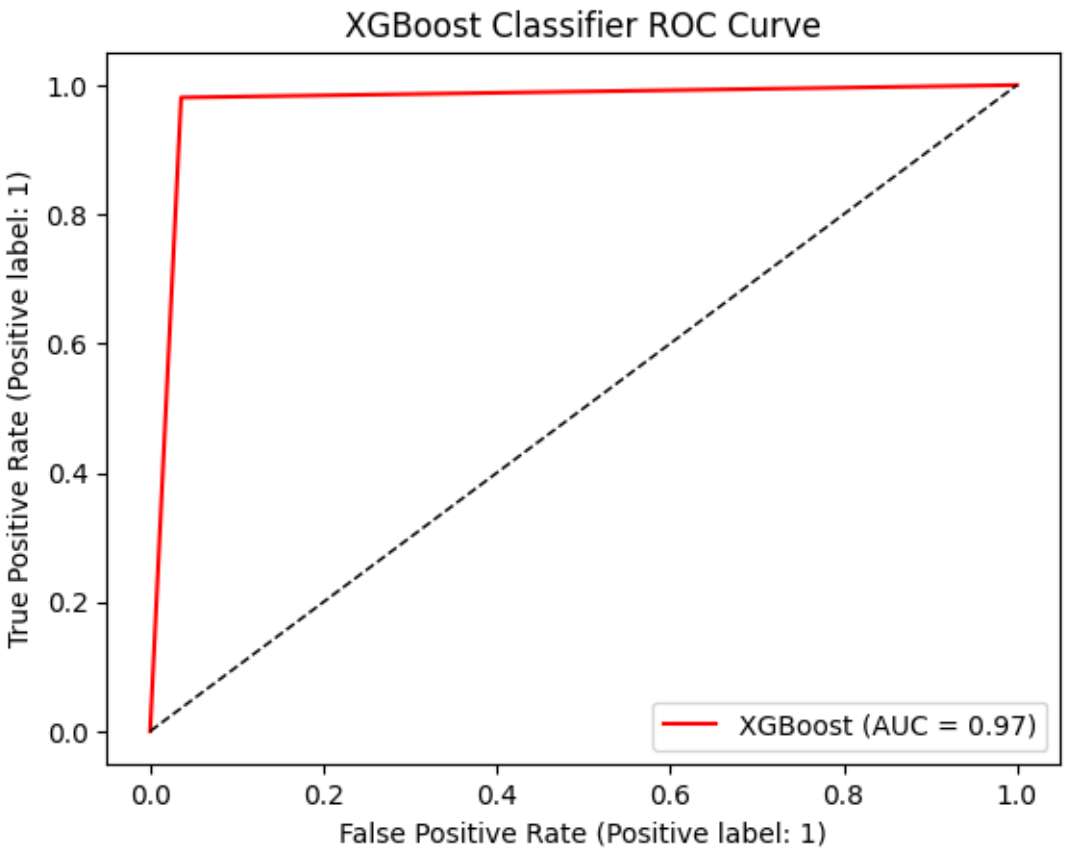
[260]:
```
#predictions
xg1_preds = grid.predict(X_test2)
```

[261]:
```
# Compute predictions

# compute predict probabilties on test set
# only need the probabilities for the positive class only
pred_probs_xg1 = grid.predict_proba(X_test2)[:, 1]
```

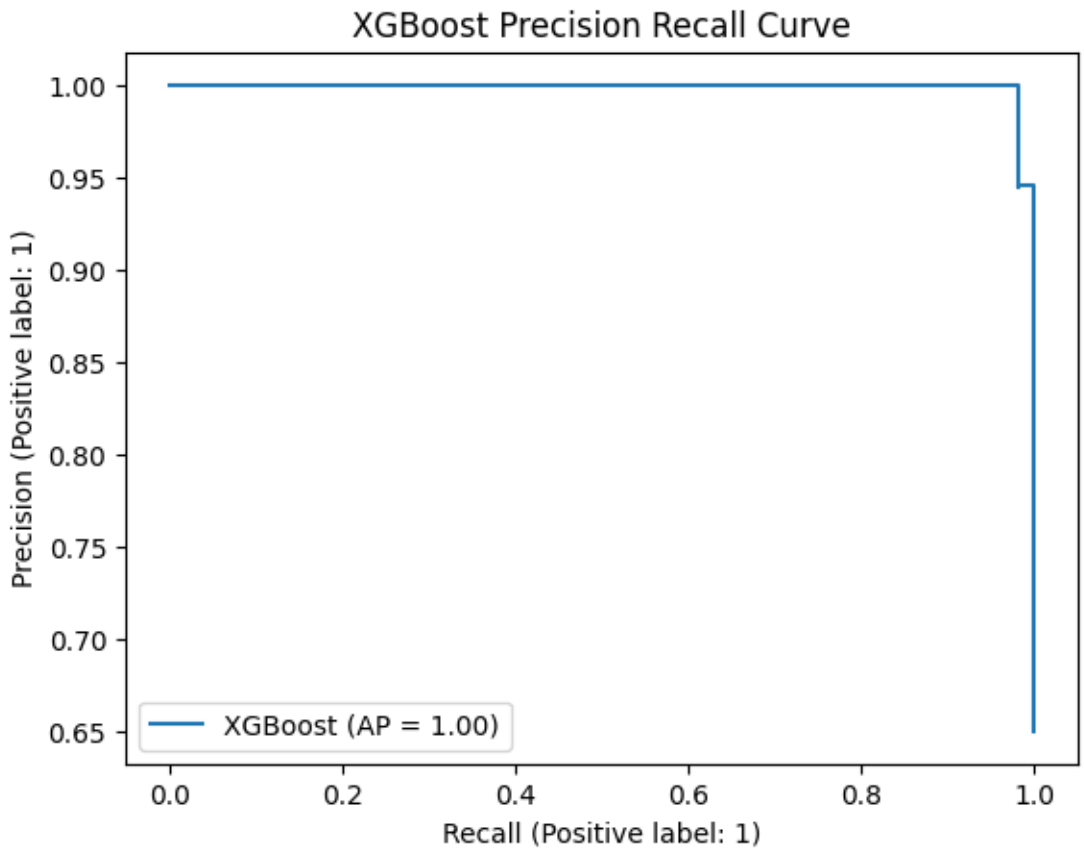
[262]:
```
# Create Confusion Matrix from predictions

ConfusionMatrixDisplay.from_predictions(y_test2, xg1_preds, cmap= 'YlGnBu')

plt.title("XgBoost Classifier\n Confusion Matrix")
plt.show()
```



[263]:
```
#Define parameters for ROC Curve plot
test = y_test2
preds = xg1_preds
Name = 'XGBoost'
Title = 'XGBoost Classifier ROC Curve'
roc_curve_plt(test, preds, Name,Title)
```



[264]:
```
#Define parameters for prc
test = y_test2
pred_probs = pred_probs_xg1
Name = "XGBoost"
Title = "XGBoost Precision Recall Curve"
prc_plt(test, pred_probs, Name, Title)
```



[265]:
```
# Build Classification report
print(classification_report(y_test2, y_preds_ab))
```

```
              precision    recall  f1-score   support

         0.0       0.96      0.93      0.95        28
         1.0       0.96      0.98      0.97        52

    accuracy                           0.96        80
   macro avg       0.96      0.95      0.96        80
weighted avg       0.96      0.96      0.96        80
```

6 Check the models against Validation Set

[236]:
```
#Check the models against the validation set
#Compute predictions

def preds_counts(model):
```

```
    val_preds = model.predict(val_scale_test) #val set obs

    #create prediction dataframe
    val_preds_df = pd.DataFrame(val_preds, columns=['class'])

    #actual class
    val_scale_class_df = pd.DataFrame(val_scale_class, columns=['actual_class'])

    #add the preds column to the actuals df
    val_scale_class_df['pred_class'] = val_preds_df['class']

    # Compare the columns and display count
    val_scale_class_df['diff']= (val_scale_class_df.apply(lambda x:␣
↪x['actual_class'] if x['actual_class'] ==\
                                                    x['pred_class'] else␣
↪'False', axis =1))
    # Count the differences
    counts = val_scale_class_df.loc[val_scale_class_df['diff'] == 'False'].
↪value_counts()
    return(counts, val_preds)
```

[233]:
```
def roc_curve_plt(test, preds, Name,Title):
    lw = 1
    RocCurveDisplay.from_predictions(test, preds,
                                name = Name,
                                color="red", pos_label=None)
    plt.plot([0, 1], [0, 1], 'k--', lw=lw)
    plt.title(Title)
    plt.show()
    return
```

[234]:
```
# Compute average precision score
# Plot the precision recall curve

def prc_plt(test, pred_probs, Name, Title):
    PrecisionRecallDisplay.from_predictions(
         test, pred_probs, name = Name)
    plt.title(Title)
    plt.show()
    return
```

6.1 RF with Validation Set

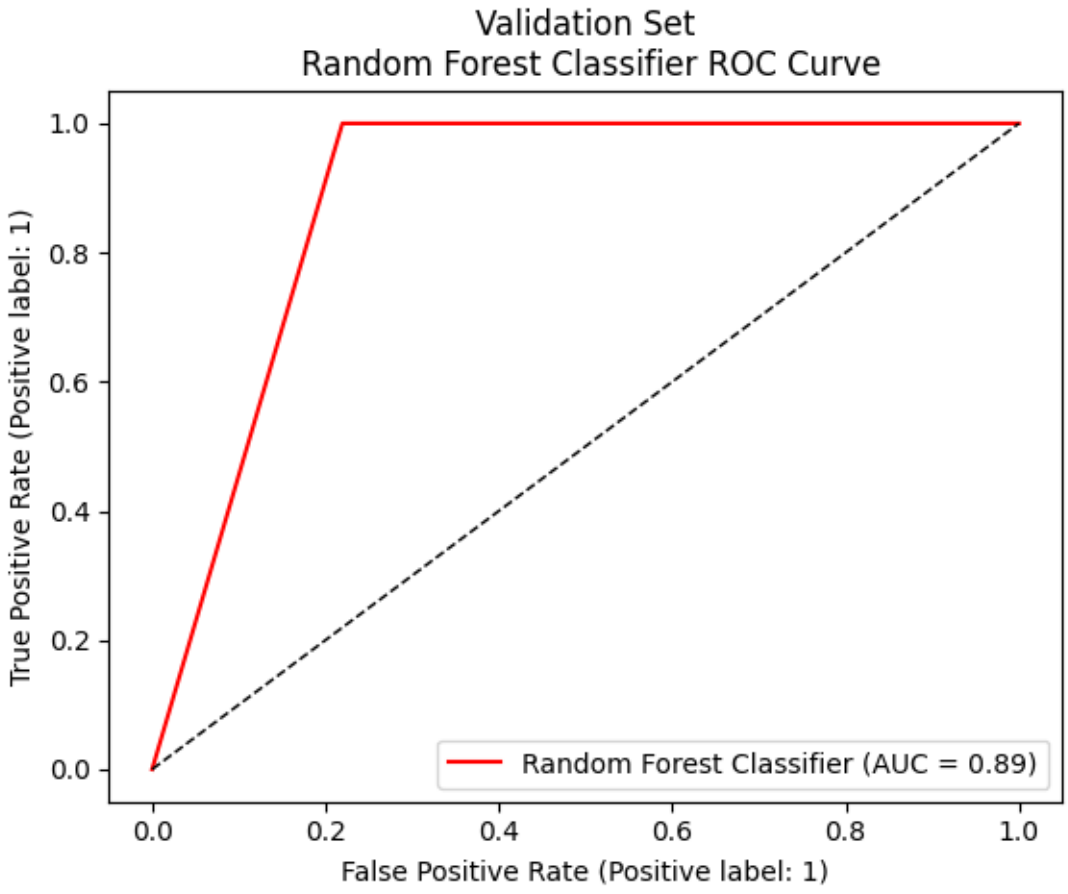
[266]:
```
model = rf_model
counts, rf_preds2 = preds_counts(model)
counts
```

[266]:
```
actual_class  pred_class  diff
0.0           1.0         False    9
dtype: int64
```

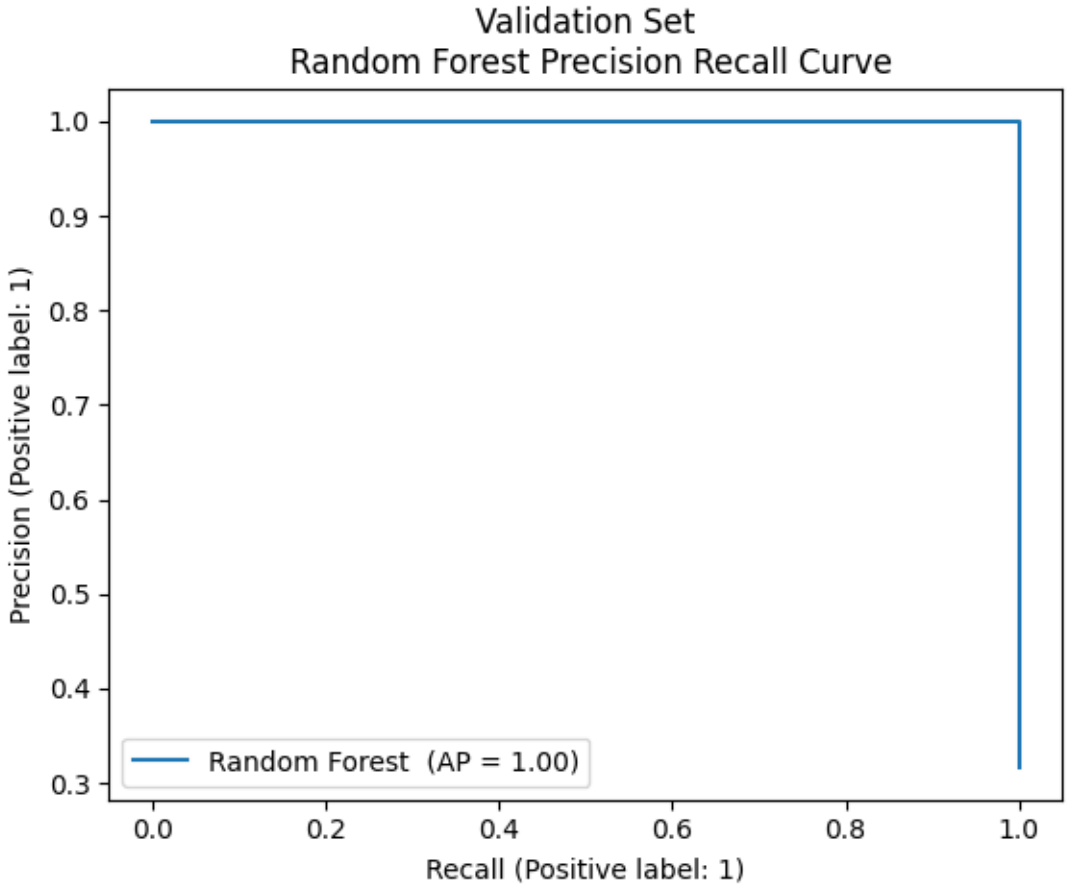
[268]:
```
# Compute predictions

# compute predict probabilties on test set
# only need the probabilities for the positive class only
pred_probs_rf2 = rf_model.predict_proba(val_scale_test)[:, 1]
```

[ ]:
```

```

[269]:
```
#Define parameters for ROC Curve plot
test = val_scale_class
preds = rf_preds2
Name = 'Random Forest Classifier'
Title = 'Validation Set\n Random Forest Classifier ROC Curve'
roc_curve_plt(test, preds, Name,Title)
```



[270]:
```
#Define parameters for prc
test = val_scale_class
pred_probs = pred_probs_rf2
Name = "Random Forest "
Title = "Validation Set\n Random Forest Precision Recall Curve"
prc_plt(test, pred_probs, Name, Title)
```



6.2 AdaBoost with Validation Set

[ ]:
```
```

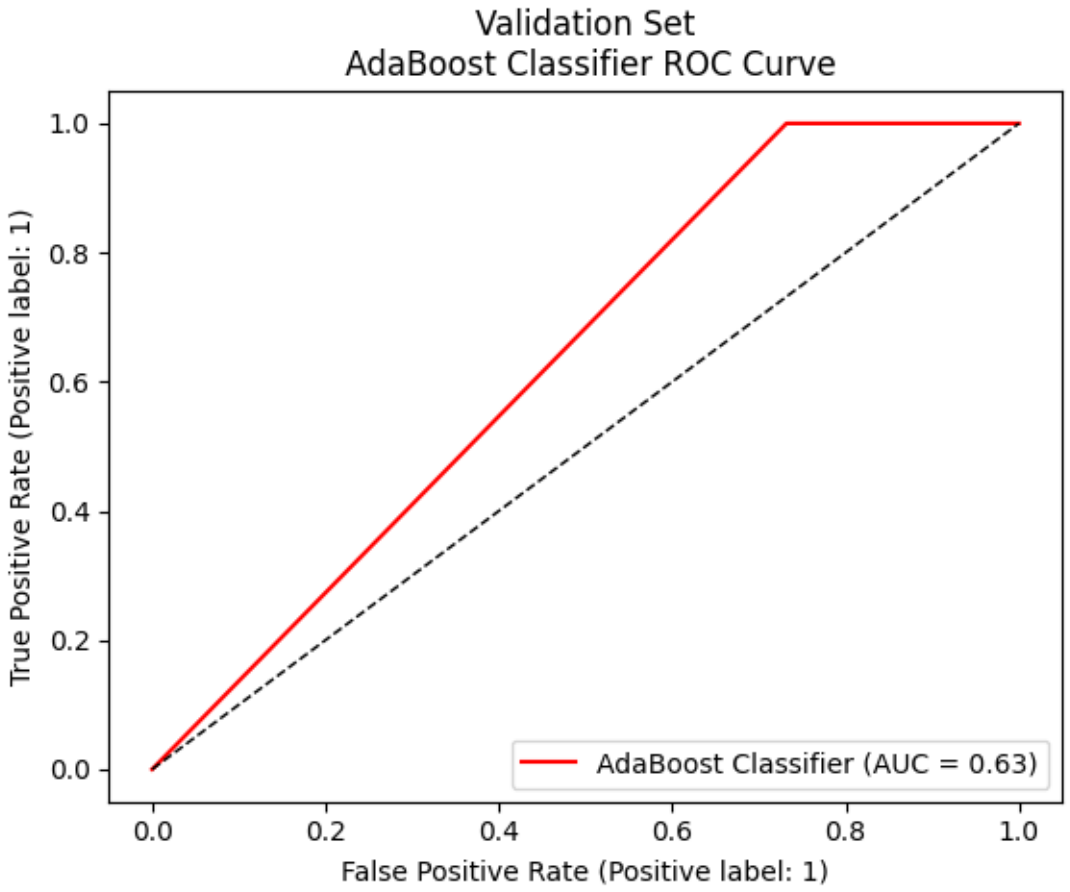
[237]:
```
model = gs_ab_model
counts, ab_preds2 = preds_counts(model)
counts
```

[237]:
```
actual_class  pred_class  diff
0.0           1.0         False    30
dtype: int64
```

[239]:
```
# Compute predictions

# compute predict probabilties on test set
# only need the probabilities for the positive class only
pred_probs_ab2 = gs_ab_model.predict_proba(val_scale_test)[:, 1]
```

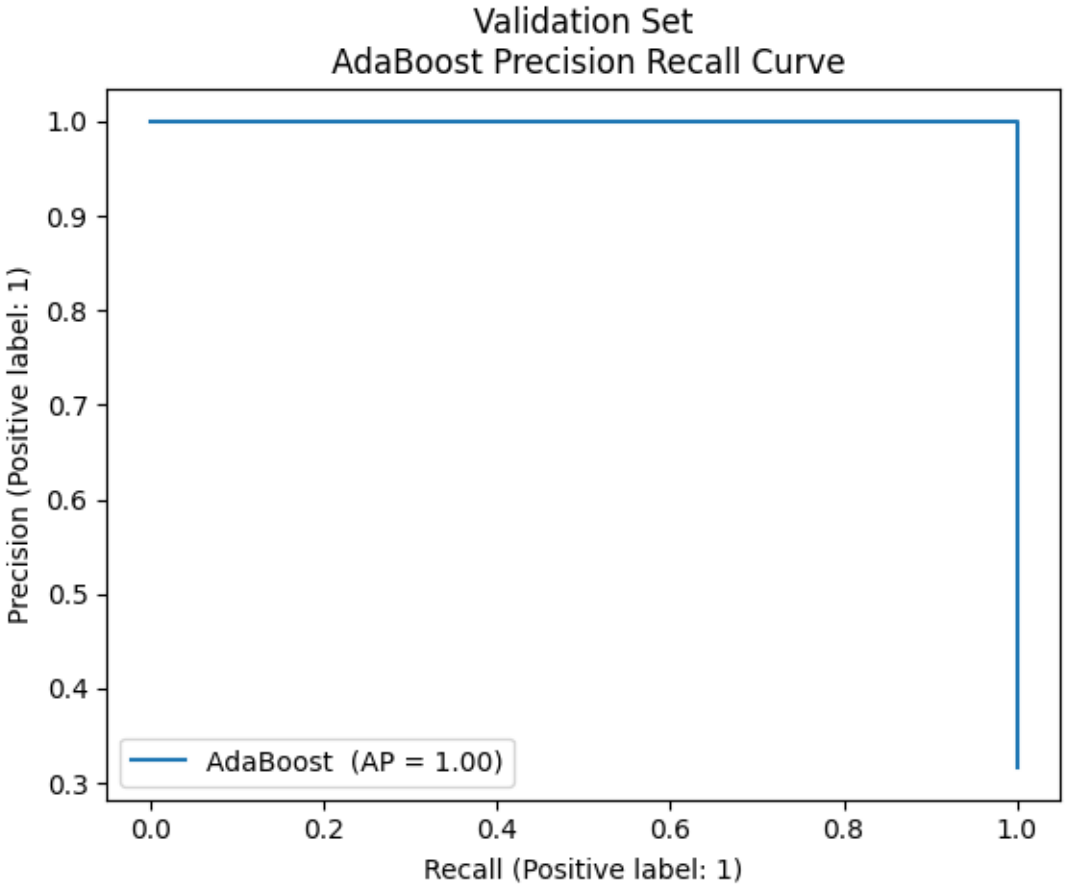
[ ]:
```
```

[242]:
```
#Define parameters for ROC Curve plot
test = val_scale_class
preds = ab_preds2
Name = 'AdaBoost Classifier'
Title = 'Validation Set\n AdaBoost Classifier ROC Curve'
roc_curve_plt(test, preds, Name,Title)
```



[243]:
```
#Define parameters for prc
test = val_scale_class
pred_probs = pred_probs_ab2
Name = "AdaBoost "
```

```
Title = "Validation Set\n AdaBoost Precision Recall Curve"
prc_plt(test, pred_probs, Name, Title)
```



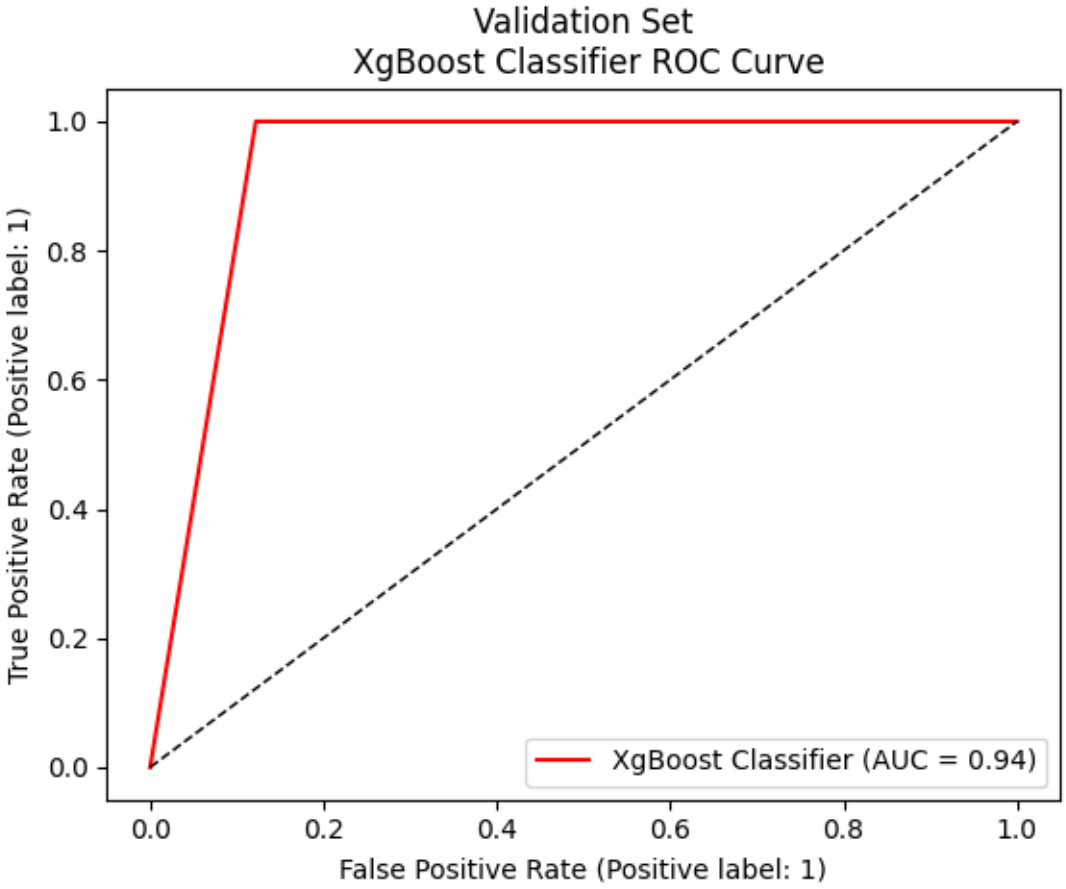
6.3 XgBoost with Validation Set

[271]:
```
model = grid
counts, xg_preds2 = preds_counts(model)
counts
```

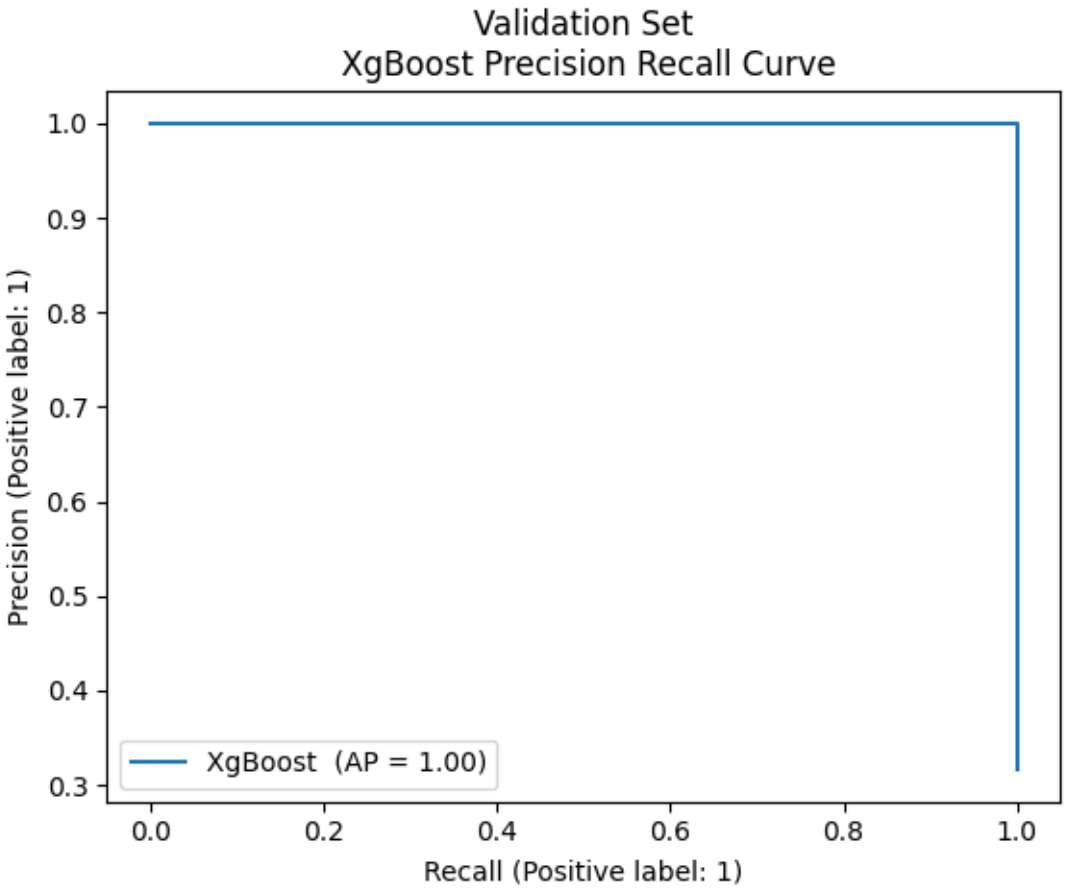
[271]:
```
actual_class  pred_class  diff
0.0           1           False    5
dtype: int64
```

[272]:
```
# Compute predictions

# compute predict probabilties on test set
# only need the probabilities for the positive class only
pred_probs_xg2 = grid.predict_proba(val_scale_test)[:, 1]
```

[274]:
```
#Define parameters for ROC Curve plot
test = val_scale_class
preds = xg_preds2
Name = 'XgBoost Classifier'
Title = 'Validation Set\n XgBoost Classifier ROC Curve'
roc_curve_plt(test, preds, Name,Title)
```



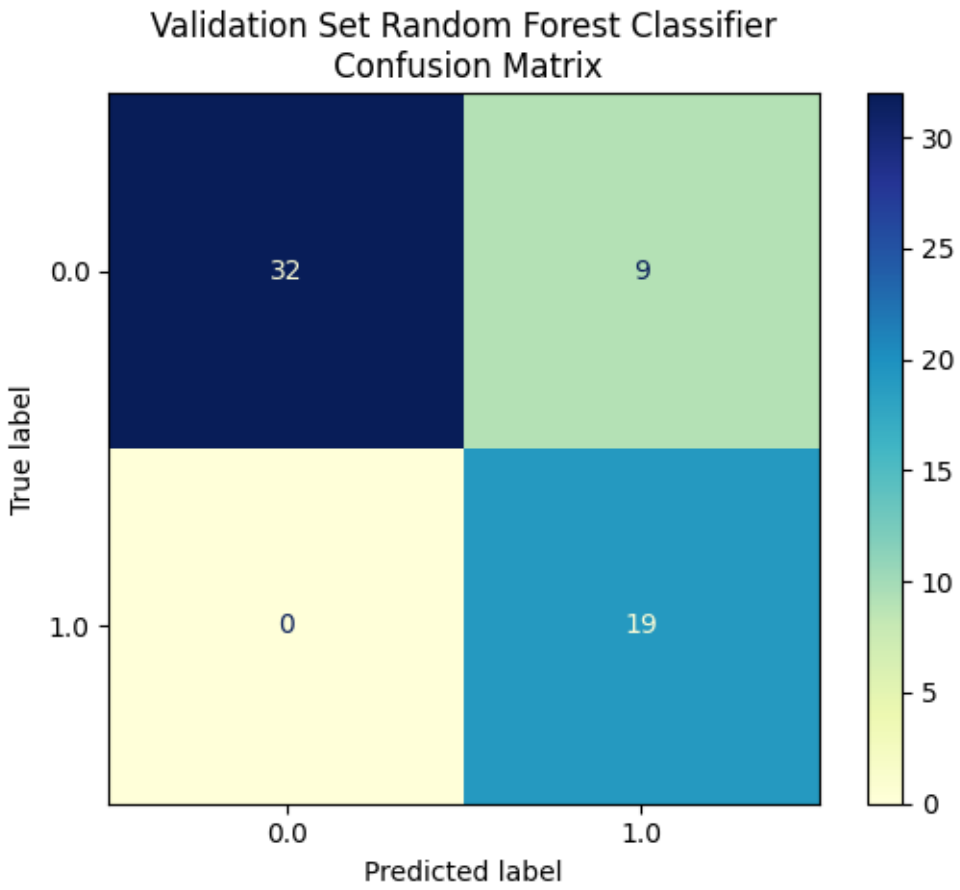
[275]:
```
#Define parameters for prc
test = val_scale_class
pred_probs = pred_probs_xg2
Name = "XgBoost "
Title = "Validation Set\n XgBoost Precision Recall Curve"
prc_plt(test, pred_probs, Name, Title)
```



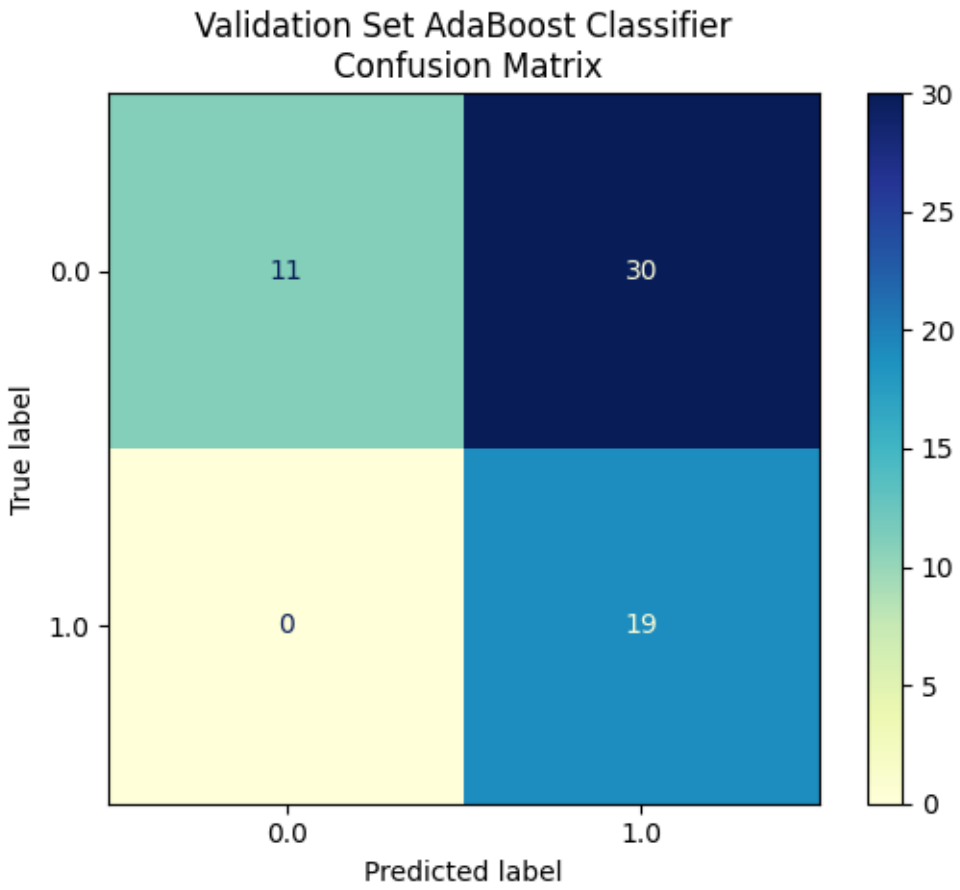
6.4 Confusion Matrices and Accuracy Scores

[276]:
```
# Create Confusion Matrix from predictions using validation set
#val_scale = scaler.fit_transform(val_set_test)
#val_class_scale = scaler.fit_transform(val_set_class)

def cm_model(Title, preds):
    ConfusionMatrixDisplay.from_predictions(val_scale_class, preds, cmap=␣
 ↪'YlGnBu')

    plt.title(Title)
    plt.show()
    return
```

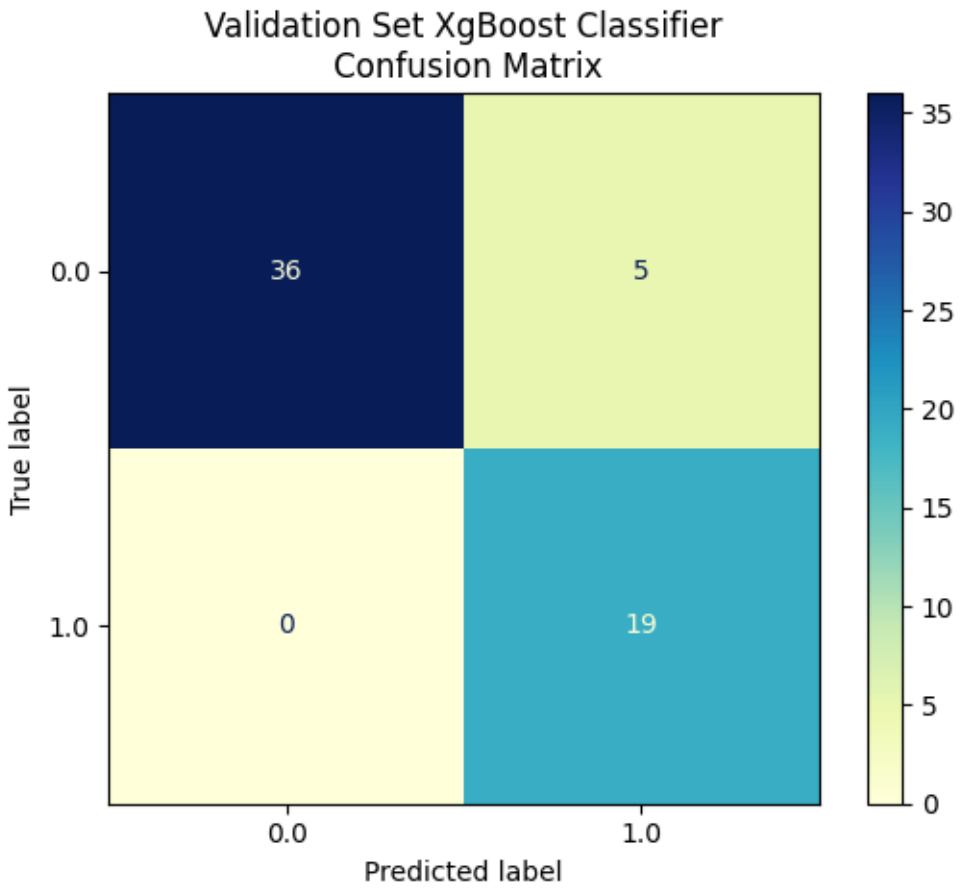
[277]:
```
Title = 'Validation Set Random Forest Classifier\n Confusion Matrix'
preds = rf_preds2
cm_model(Title, preds)
```



[246]:
```
# deinately appears that AdaBoost suffers from overfitting

Title = 'Validation Set AdaBoost Classifier\n Confusion Matrix'
preds = ab_preds2
cm_model(Title, preds)
```



[278]:
```
Title = 'Validation Set XgBoost Classifier\n Confusion Matrix'
preds = xg_preds2
cm_model(Title, preds)
```



[248]:
```
# Accuracy scores
# View accuracy score
def acc_score(preds):
    score = accuracy_score(val_scale_class, preds)
    return(score)
```

[279]:
```
preds = rf_preds2
accuracy = acc_score(preds)
accuracy
```

[279]: 0.85

[249]:
```
preds = ab_preds2
accuracy = acc_score(preds)
accuracy
```

[249]: 0.5

[280]:
```
preds = xg_preds2
accuracy = acc_score(preds)
accuracy
```

[280]:
```
0.9166666666666666
```

[ ]:
```

```