EDA

November 26, 2021

0.1 Problem Overview

We have to generate a code repository that can be used across biomedical data science projects. The dataset used for the proof of concept can help physicians better understand chronic kidney disease (CKD) using numerous measurements and biomarkers that have been collected. The dataset can be found here.

0.2 Your Task

You have been tasked to write modularized reusable Python code that can be used by other similar datasets and other similar projects to help a physician understand 1. risk factors for CKD and 2. potential CKD subtypes.

Please spend 2-4 hours writing a code base to perform appropriate analysis. Once you have completed, please send us the code and instructions to run the code by November 26th using the OneDrive link shared separately. We expect to receive .py file for codes and .doc/txt/pdf file for instructions.

[1]:
```
!pip install liac-arff
```

```
/bin/sh: pip: command not found
```

[2]:
```
filename = '../clean_data//chronic_kidney_disease.arff'
```

[3]:
```
import arff
import pandas as pd
import numpy as np
from pandas.api.types import is_string_dtype, is_numeric_dtype
from collections import Counter
import seaborn as sns
from matplotlib import pyplot as plt
sns.set(rc={'figure.figsize':(20,20)})
```

[4]:
```
dataset = arff.load(open(filename, 'r'))
CODES_FOR_1 = ['yes', 'abnormal', 'present' ,'poor']
CODES_FOR_0 = ['no', 'normal', 'notpresent' ,'good']

headers = []
for feature in dataset['attributes']:
    headers.append(feature[0])
```

```
data = pd.DataFrame(dataset['data'], columns = headers)
data["class"] = (data["class"] == 'ckd').astype('int')

data.head()
```

[4]:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	…	pcv	wbcc	rbcc	htn	dm	cad	appet	pe	ane	class
0	48.0	80.0	1.020	1	0	None	normal	notpresent	notpresent	121.0	…	44.0	7800.0	5.2	yes	yes	no	good	no	no	1
1	7.0	50.0	1.020	4	0	None	normal	notpresent	notpresent	NaN	…	38.0	6000.0	NaN	no	no	no	good	no	no	1
2	62.0	80.0	1.010	2	3	normal	normal	notpresent	notpresent	423.0	…	31.0	7500.0	NaN	no	yes	no	poor	no	yes	1
3	48.0	70.0	1.005	4	0	normal	abnormal	present	notpresent	117.0	…	32.0	6700.0	3.9	yes	no	no	poor	yes	yes	1
4	51.0	80.0	1.010	2	0	normal	normal	notpresent	notpresent	106.0	…	35.0	7300.0	4.6	no	no	no	good	no	no	1

[5 rows x 25 columns]

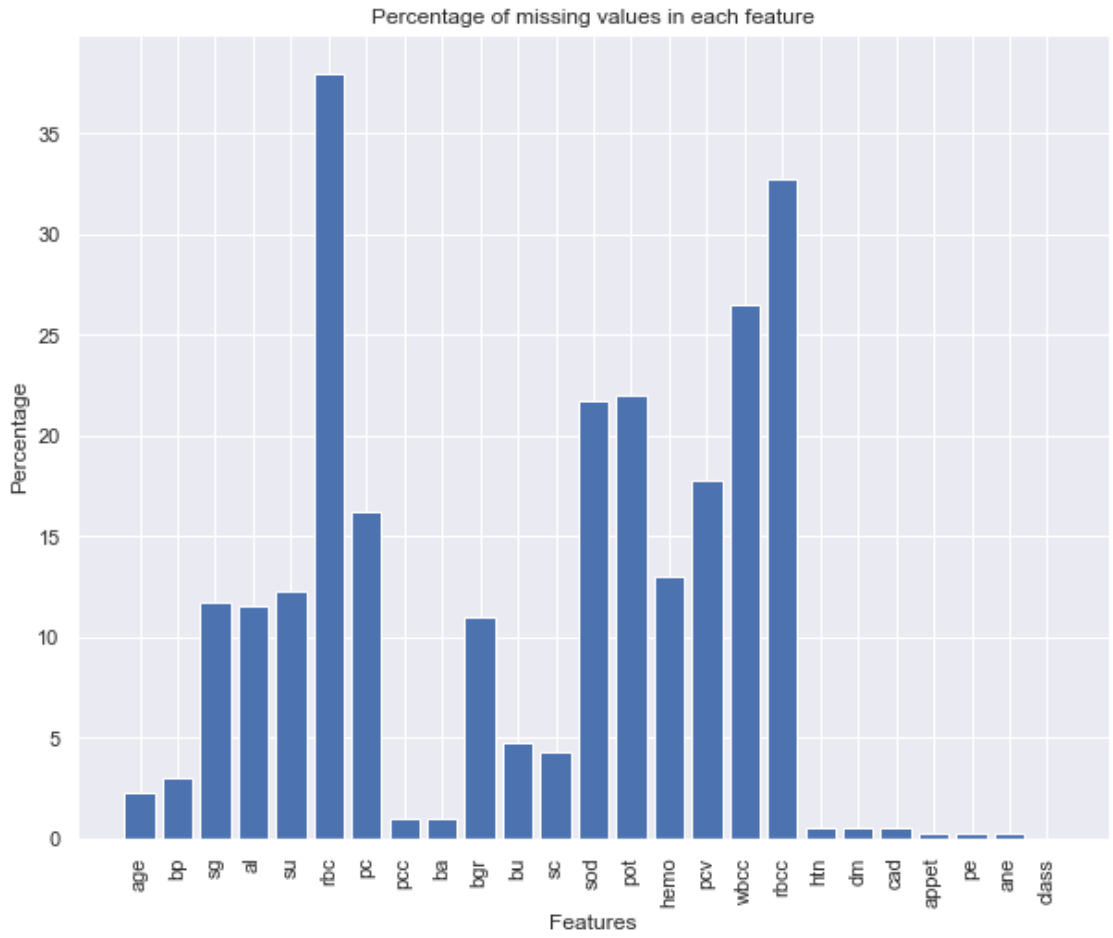
[5]:
```
raw_data = data.copy()
```

[6]:
```
distribution = Counter(data['class'])
print(f'Class distribution is {distribution} where 1:disease, 0:no disease')
```

```
Class distribution is Counter({1: 250, 0: 150}) where 1:disease, 0:no disease
```

[7]:
```
missing_precents = (data.isnull().sum()/len(data))*100
# missing_counts
```

[8]:
```
# Can sort it based on the values
plt.figure(figsize=(10,8))
plt.title("Percentage of missing values in each feature")
plt.xlabel('Features')
plt.ylabel('Percentage')
plt.bar(missing_precents.index,missing_precents)
plt.xticks(rotation=90);
```



[9]:
```
def numeric_category_features(data):
    numerics, categories = [], []
    for indx in data:
        if is_numeric_dtype(data[indx]):
            numerics.append(indx)
        else:
            categories.append(indx)

    return numerics, categories
```

[10]:
```
numerics, categories = numeric_category_features(data)
print(f' numerics and categories are {numerics}\n {categories}')
```

```
 numerics and categories are ['age', 'bp', 'bgr', 'bu', 'sc', 'sod', 'pot',
'hemo', 'pcv', 'wbcc', 'rbcc', 'class']
 ['sg', 'al', 'su', 'rbc', 'pc', 'pcc', 'ba', 'htn', 'dm', 'cad', 'appet', 'pe',
'ane']
```

1 Modify the below methods to use the calculated numerics and categories instead of using if checks

[11]:
```
# Impute missing values:

def impute_missing_value(feature, with_mean=True):
    if len(feature[feature.isnull()==True])==0:
        return

    if is_numeric_dtype(feature):
        if with_mean:
            mean_value = feature.mean()
            feature[feature.isnull()==True] = mean_value
        else:
            median_value = feature.median()
            feature[feature.isnull()==True] = median_value
        return

    mode_value = feature.mode()[0] # .mode() returns a Series this is why␣
↪indexing
    feature[feature.isnull()==True] = mode_value
    return
```

[12]:
```
for i,indx in enumerate(data):
    impute_missing_value(data[indx])
data.head()
```

```
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:10:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:17:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[12]:

	age	bp	sg	al	su	rbc	pc	pcc	ba	\
0	48.0	80.0	1.020	1	0	normal	normal	notpresent	notpresent	
1	7.0	50.0	1.020	4	0	normal	normal	notpresent	notpresent	
2	62.0	80.0	1.010	2	3	normal	normal	notpresent	notpresent	
3	48.0	70.0	1.005	4	0	normal	abnormal	present	notpresent	

```
4  51.0  80.0  1.010  2  0  normal    normal  notpresent  notpresent

          bgr  …   pcv    wbcc      rbcc  htn   dm  cad  appet   pe  ane  \
0  121.000000  …  44.0  7800.0  5.200000  yes  yes   no   good   no   no
1  148.036517  …  38.0  6000.0  4.707435   no   no   no   good   no   no
2  423.000000  …  31.0  7500.0  4.707435   no  yes   no   poor   no  yes
3  117.000000  …  32.0  6700.0  3.900000  yes   no   no   poor  yes  yes
4  106.000000  …  35.0  7300.0  4.600000   no   no   no   good   no   no

  class
0     1
1     1
2     1
3     1
4     1

[5 rows x 25 columns]
```

[13]:
```
def one_hot_encode(data):
    for indx in data:
        if not is_numeric_dtype(data[indx]):
            try:
                data[indx] = pd.to_numeric(data[indx])
            except:
#                 data[indx] = pd.get_dummies(data[indx])
                one_val = list(set(data[indx].unique()).
 ↪intersection(CODES_FOR_1))[0]
                data[indx] = (data[indx] == one_val).astype('int')
```

[14]:
```
set(np.unique(data['dm'])).intersection(CODES_FOR_1)
```

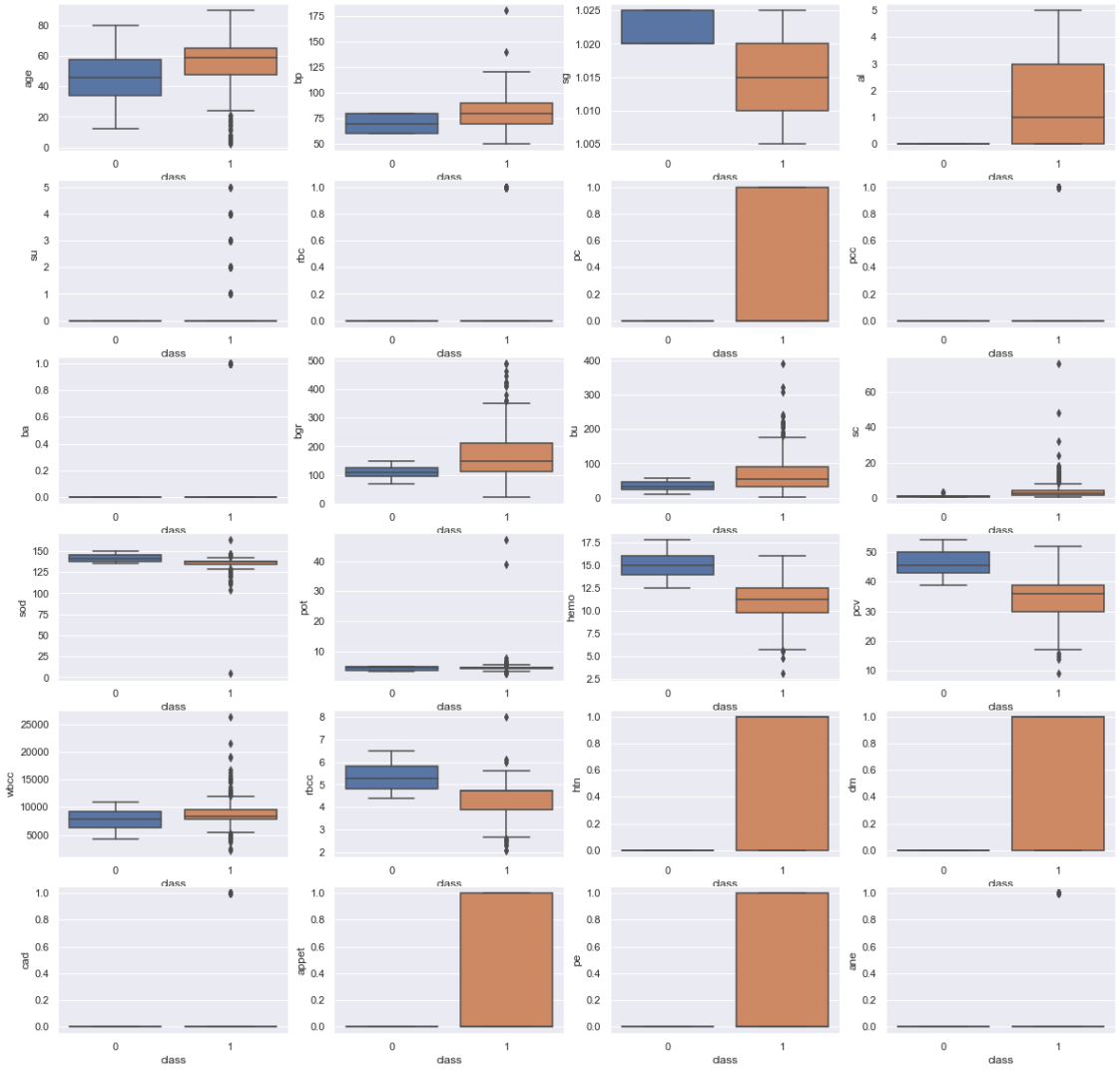
[14]:
```
{'yes'}
```

[15]:
```
one_hot_encode(data)
data.head()
```

[15]:
```
    age    bp     sg  al  su  rbc  pc  pcc  ba         bgr  …   pcv    wbcc  \
0  48.0  80.0  1.020   1   0    0   0    0   0  121.000000  …  44.0  7800.0
1   7.0  50.0  1.020   4   0    0   0    0   0  148.036517  …  38.0  6000.0
2  62.0  80.0  1.010   2   3    0   0    0   0  423.000000  …  31.0  7500.0
3  48.0  70.0  1.005   4   0    0   1    1   0  117.000000  …  32.0  6700.0
4  51.0  80.0  1.010   2   0    0   0    0   0  106.000000  …  35.0  7300.0

       rbcc  htn  dm  cad  appet  pe  ane  class
0  5.200000    1   1    0      0   0    0      1
1  4.707435    0   0    0      0   0    0      1
2  4.707435    0   1    0      1   0    1      1
```

```
3  3.900000    1   0    0       1   1    1       1
4  4.600000    0   0    0       0   0    0       1

[5 rows x 25 columns]
```

[16]:
```
all_names = list(data.columns)
all_names.remove('class')
import seaborn as sns

f, axes = plt.subplots(6, 4)
for i in range(6):
    for j in range(4):
        sns.boxplot(x="class", y=all_names[i*4+j], data=data, ax=axes[i,j])
```



[17]:
```
X = data.drop('class', axis=1)
Y = data['class']
```

[18]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.
 ↪30,random_state=245, stratify=Y)

print(f'Class distribution in train is: {Counter(Y_train)}')
print(f'Class distribution in test is: {Counter(Y_test)}')
```

```
Class distribution in train is: Counter({1: 175, 0: 105})
Class distribution in test is: Counter({1: 75, 0: 45})
```

[19]:
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler

def train_eval(model, X, Y):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.
 ↪30,random_state=245, stratify=Y)

    scaler = StandardScaler().fit(X_train)
    Xtrain_scaled = scaler.transform(X_train)
    Xtest_scaled = scaler.transform(X_test)

    model.fit(Xtrain_scaled, Y_train)

    acc, roc, prec, rec, f1, r2 = evaluate_metrics(model, Xtest_scaled, Y_test)
    display_df = pd.DataFrame([[acc, roc, prec, rec, f1, r2, Xtest_scaled.
 ↪shape[1]]], columns=["Accuracy", "ROC", "Precision", "Recall", "F1 Score",␣
 ↪"R2",'Feature Count'])

    return display_df

def fit_model(X,Y):
    model = RandomForestClassifier(criterion='entropy', random_state=47)

    model.fit(X, Y)
    return model
```

[20]:
```
from sklearn.metrics import accuracy_score, roc_auc_score, precision_score,␣
 ↪recall_score, f1_score, r2_score

def evaluate_metrics(model, X_test, Y_test):
    y_hat = model.predict(X_test)

    acc = accuracy_score(Y_test, y_hat)
    roc = roc_auc_score(Y_test, y_hat)
```

```
    prec = precision_score(Y_test, y_hat)
    rec = recall_score(Y_test, y_hat)
    f1 = f1_score(Y_test, y_hat)
    r2 = r2_score(Y_test, y_hat)

    return acc, roc, prec, rec, f1, r2
```

[21]:
```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(criterion='entropy', random_state=47)

results_df_all = train_eval(model, X, Y)
results_df_all
```

[21]:

	Accuracy	ROC	Precision	Recall	F1 Score	R2	Feature Count
0	0.983333	0.977778	0.974026	1.0	0.986842	0.928889	24

[22]:
```
TARGET_THRESHOLD = 0.4
data_cor = data.corr()
cor_target = abs(data_cor["class"])

high_corr_features = cor_target[cor_target > TARGET_THRESHOLD]

feature_names = [index for index, value in high_corr_features.items()]

feature_names.remove('class')

print(feature_names)
```

```
['sg', 'al', 'bgr', 'hemo', 'pcv', 'rbcc', 'htn', 'dm']
```

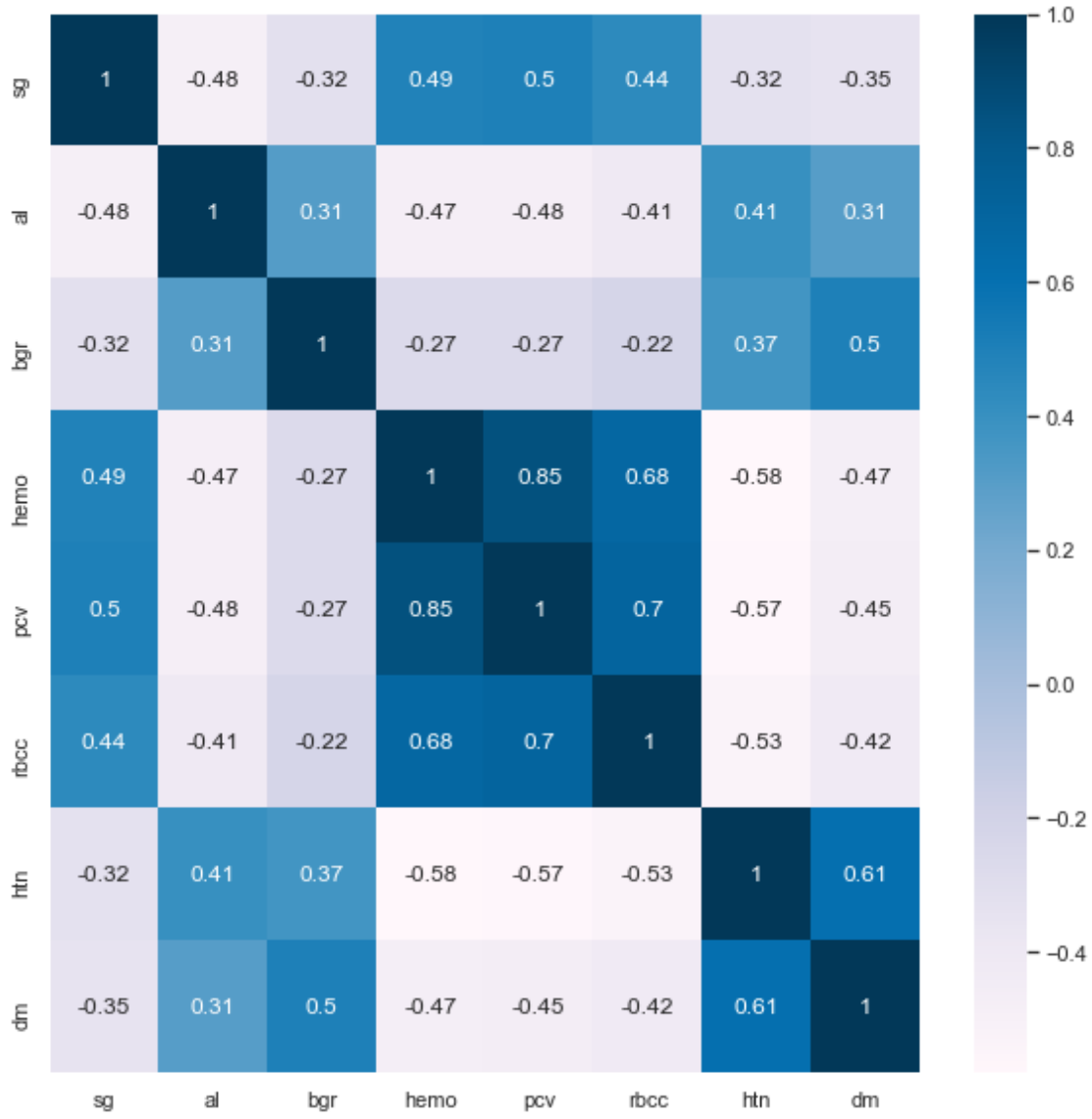
[23]:
```
# model = RandomForestClassifier(criterion='entropy', random_state=47)

results_df = train_eval(model, X[feature_names], Y)
results_df
```

[23]:

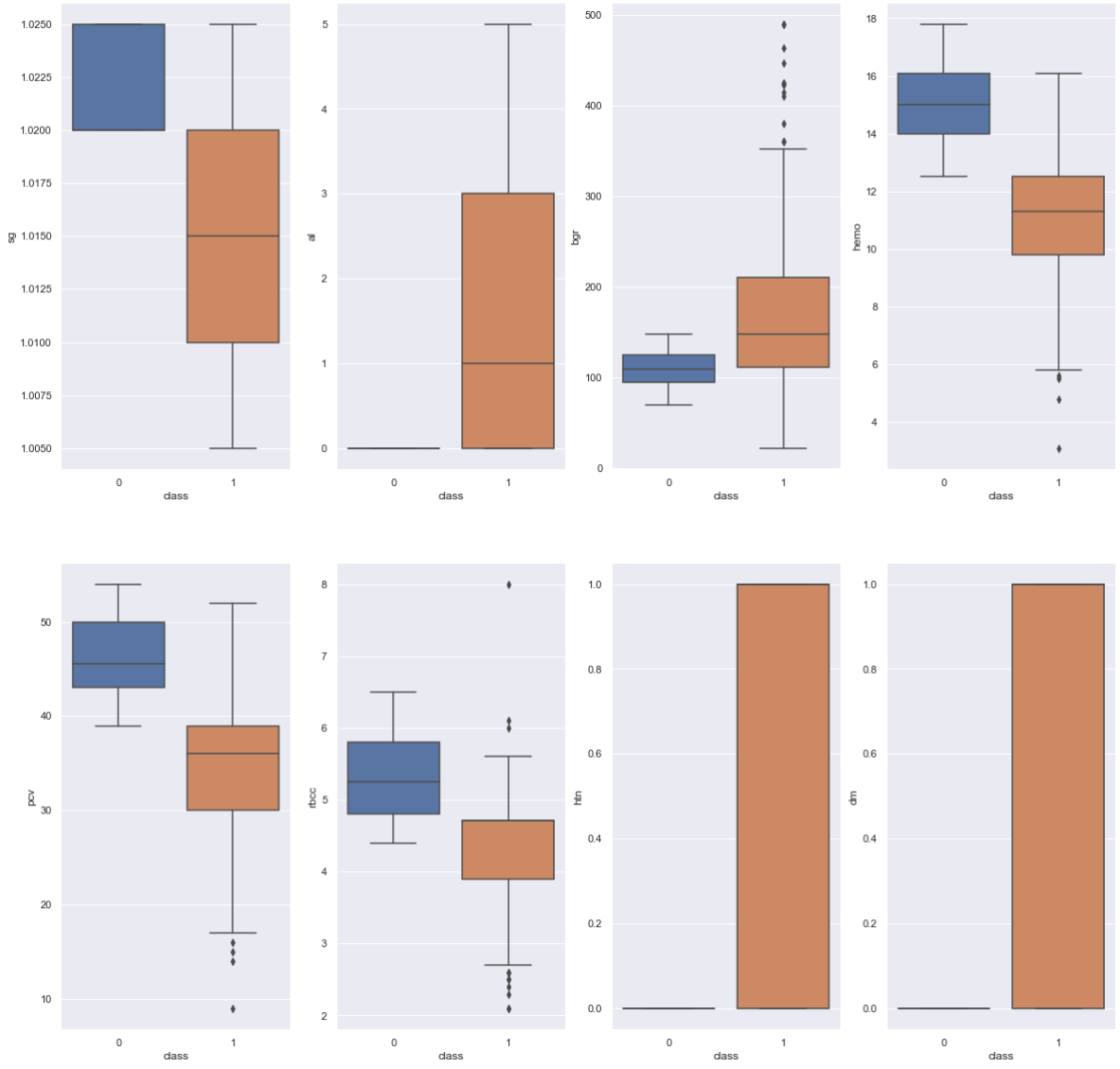
	Accuracy	ROC	Precision	Recall	F1 Score	R2	Feature Count
0	0.983333	0.977778	0.974026	1.0	0.986842	0.928889	8

[24]:
```
model_Res = pd.concat([results_df_all,results_df], axis=0)
```

[25]:
```
plt.figure(figsize=(10,10))
sns.heatmap(data_cor.loc[feature_names][feature_names],  annot=True, cmap=plt.
 ↪cm.PuBu);
```



[26]:
```
f, axes = plt.subplots(2, 4)

for i in range(2):
    for j in range(4):
        sns.boxplot(x="class", y=feature_names[i*4+j], data=data, ax=axes[i,j])
```



[27]:
```
def get_similar_features(data_cor, feature_names, corr_threshold = 0.75):
    subset = data_cor.loc[feature_names][feature_names]
    result = subset[subset > corr_threshold].isna().sum() < (len(subset)-1)

    similar_features = [index for index, value in result.items() if value==True]

    return similar_features
```

[28]:
```
FEATURE_THRESHOLD = 0.75
sim = get_similar_features(data_cor, feature_names)
```

[29]:
```
print(f'Keys factors are {feature_names}.\nOut of these feature, {sim} are␣
 ↪similar')
```

```
Keys factors are ['sg', 'al', 'bgr', 'hemo', 'pcv', 'rbcc', 'htn', 'dm'].
```

```
Out of these feature, ['hemo', 'pcv'] are similar
```

2 TASK 2

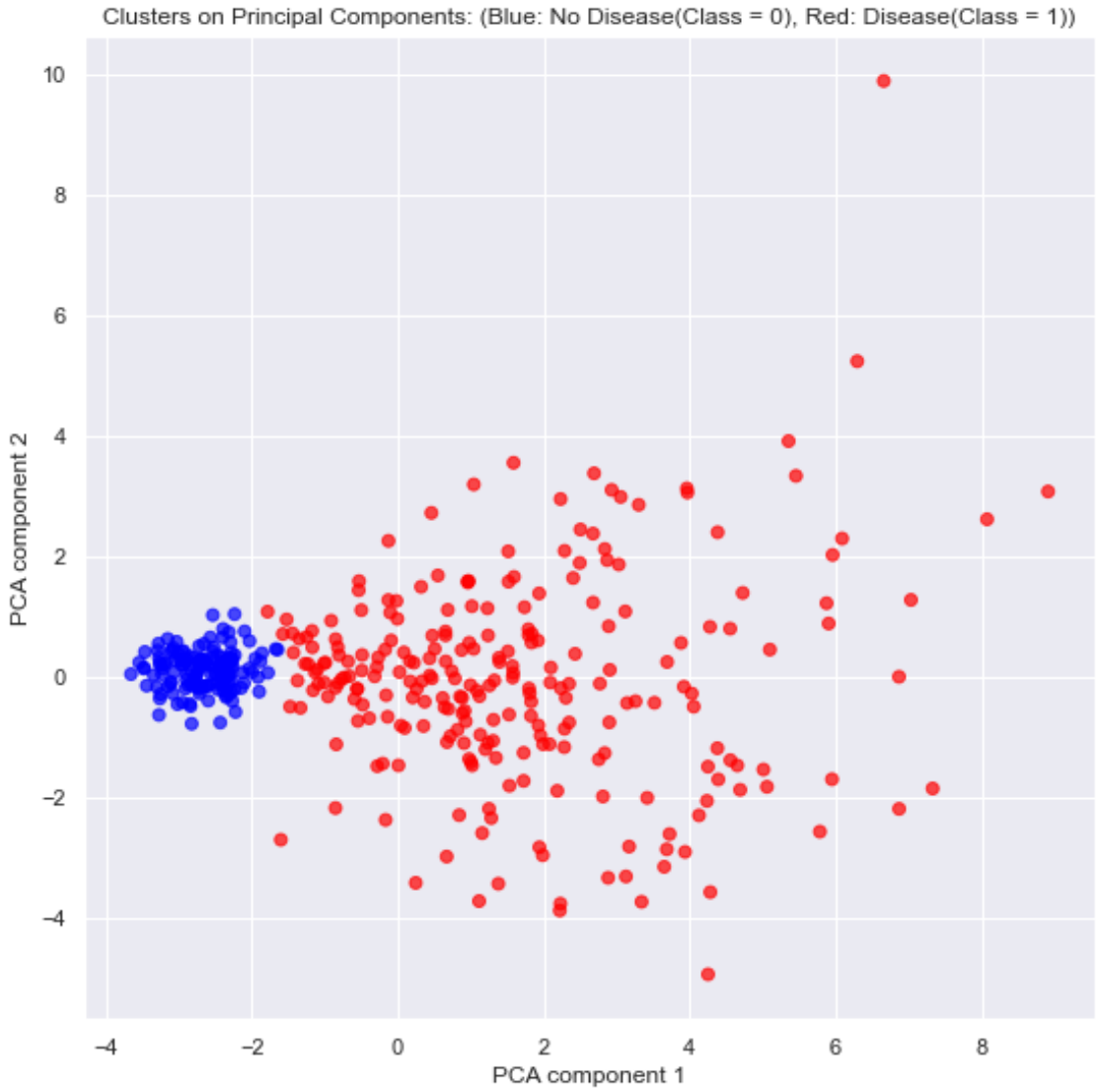
Second part asks to find subtypes of CKD (i.e. different types of patients that group together based on the variables that were observed) identified via data driven techniques. This could be by severity or other things as well (e.g. CKD with diabetes patients).

[30]:
```
from sklearn.decomposition import PCA
from sklearn import preprocessing
import matplotlib.pyplot as plt
sns.set(rc={'figure.figsize':(15,7)})
```

[31]:
```
label_color = ['red' if i==1 else 'green' for i in Y]
X_pca = data.drop(labels=['class'], axis=1)

# If I don't delete categories then I get better distinct clusters
# X_pca = pd.DataFrame(X_pca.drop(labels=categories, axis=1))
pca_columns = X_pca.columns
```

[32]:
```
X_pca = preprocessing.StandardScaler().fit_transform(X_pca)

pca = PCA(n_components=2)
pca.fit(X_pca)
pca_comp = pca.transform(X_pca)
pca_comp = pd.DataFrame(pca_comp)
```

[33]:
```
plt.figure(figsize=(9,9))
plt.title("Clusters on Principal Components: (Blue: No Disease(Class = 0), Red:␣
 ↪Disease(Class = 1))")
plt.xlabel('PCA component 1')
plt.ylabel('PCA component 2')
plt.scatter(x=pca_comp[0], y= pca_comp[1],c=Y, cmap=plt.cm.bwr, alpha=0.7)
plt.show()
```



[34]:
```
print(abs( pca.components_ ))
```

```
[[0.12739967 0.13117517 0.22985073 0.25516101 0.14836028 0.14500645
  0.22949601 0.16584007 0.12256624 0.17838606 0.24448047 0.17525979
  0.17123922 0.06536023 0.31295336 0.31510205 0.08565915 0.28427437
  0.27110724 0.24108898 0.15376761 0.19557275 0.19682429 0.191976  ]
 [0.12928264 0.00702726 0.15933282 0.17105591 0.41141244 0.03174676
  0.11822065 0.21142989 0.16310368 0.38303059 0.30379832 0.38177906
  0.25865686 0.10801031 0.15157744 0.13458455 0.21937104 0.12010987
  0.00452729 0.16640658 0.06523869 0.02890796 0.04418577 0.26704814]]
```

[35]:
```
print(f'Explained variance : {pca.explained_variance_ratio_}')
imp_indices = np.argsort(abs(pca.components_[0]))
```

```
print(f'imp indices {imp_indices} starting with least important')
imp_features = [pca_columns[indx] for indx in imp_indices[-8:]]
print(f'imp features are {imp_features}')
```

```
Explained variance : [0.29127372 0.07903137]
imp indices [13 16  8  0  1  5  4 20  7 12 11  9 23 21 22  6  2 19 10  3 18 17
14 15] starting with least important
imp features are ['sg', 'dm', 'bu', 'al', 'htn', 'rbcc', 'hemo', 'pcv']
```

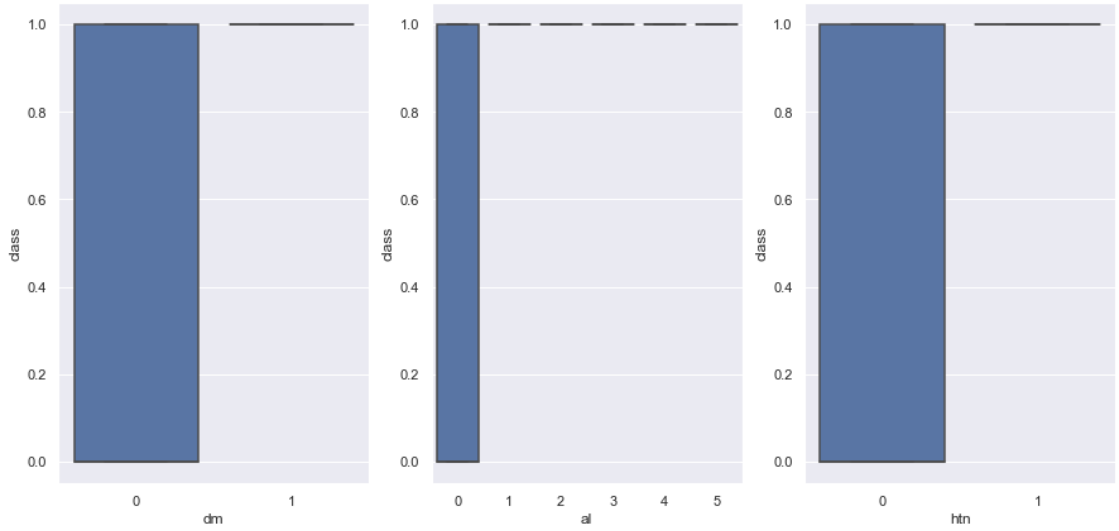
2.1 Pairplot for Features contributing most to the first principal component (orange color=ckd)

[36]:
```
imp_features = imp_features + ['class']
data_x = data[imp_features]
sns.pairplot(data_x,corner=True, hue='class');
```

```
/usr/local/lib/python3.7/site-packages/seaborn/distributions.py:288:
UserWarning: Data must have variance to compute a kernel density estimate.
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.7/site-packages/seaborn/distributions.py:288:
UserWarning: Data must have variance to compute a kernel density estimate.
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.7/site-packages/seaborn/distributions.py:288:
UserWarning: Data must have variance to compute a kernel density estimate.
  warnings.warn(msg, UserWarning)
```

2.2 Boxplots of categorical features identified by PCA

[37]:
```
# al,htn,dm

f, axes = plt.subplots(1, 3)

sns.boxplot(y="class", x='dm', data=data, ax=axes[0])
sns.boxplot(y="class", x='al', data=data, ax=axes[1])
sns.boxplot(y="class", x='htn', data=data, ax=axes[2])
plt.show()
```



2.2.1 There exists atleast one sub-category in each of the categories above where all the samples are ckd positive. For example:

if Diabetes Mellitus present (dm=1) then definitely ckd.

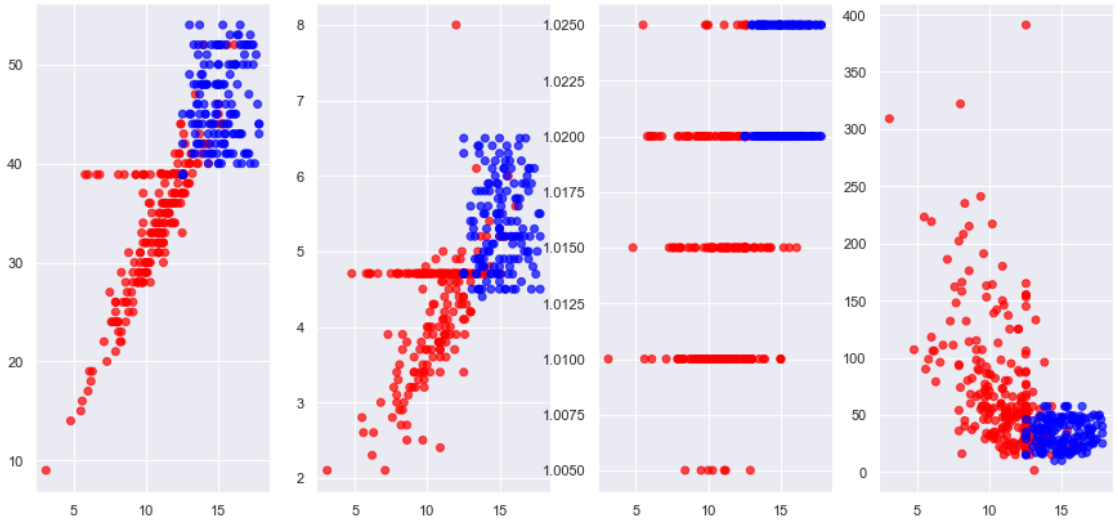
If Albumin level > 0 then definitely ckd.

If Hypertension present (htn=1) then definitely ckd.

2.3 Scatterplots of few numerical features identified by PCA

[38]:
```
# 'sg', , 'bu',  'rbcc', 'hemo', 'pcv'
plt.figure(figsize=(20,20))

f, axes = plt.subplots(1, 4)
axes[0].scatter(x=data['hemo'], y= data['pcv'],c=Y, cmap=plt.cm.bwr, alpha=0.7);
axes[1].scatter(x=data['hemo'], y= data['rbcc'],c=Y, cmap=plt.cm.bwr, alpha=0.
↪7);
axes[2].scatter(x=data['hemo'], y= data['sg'],c=Y, cmap=plt.cm.bwr, alpha=0.7);
axes[3].scatter(x=data['hemo'], y= data['bu'],c=Y, cmap=plt.cm.bwr, alpha=0.7);

plt.show()
```

```
<Figure size 1440x1440 with 0 Axes>
```



2.3.1 There exists range in each of the combinations above where most of the samples are ckd positive.

2.3.2 Low hemoglobin, low Packed Cell Volume, low Red Blood Cell Count, low Specific Gravity and high Blood Urea indicate ckd.

[39]:
```
Counter(data.loc[data['bu']>55]['class'])
```

[39]:
```
Counter({1: 122, 0: 6})
```

[ ]:
```

```

2.4 Model evaluation using PCA components

[40]:
```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(criterion='entropy', random_state=47)

results_df_pca = train_eval(model, pca_comp, Y)
model_Res = pd.concat([results_df_all, results_df, results_df_pca], axis=0)
```

[41]:
```
model_Res
```

[41]:

	Accuracy	ROC	Precision	Recall	F1 Score	R2	Feature Count
0	0.983333	0.977778	0.974026	1.000000	0.986842	0.928889	24
0	0.983333	0.977778	0.974026	1.000000	0.986842	0.928889	8
0	0.983333	0.986667	1.000000	0.973333	0.986486	0.928889	2

2.5 Model evaluation using important features as per PCA

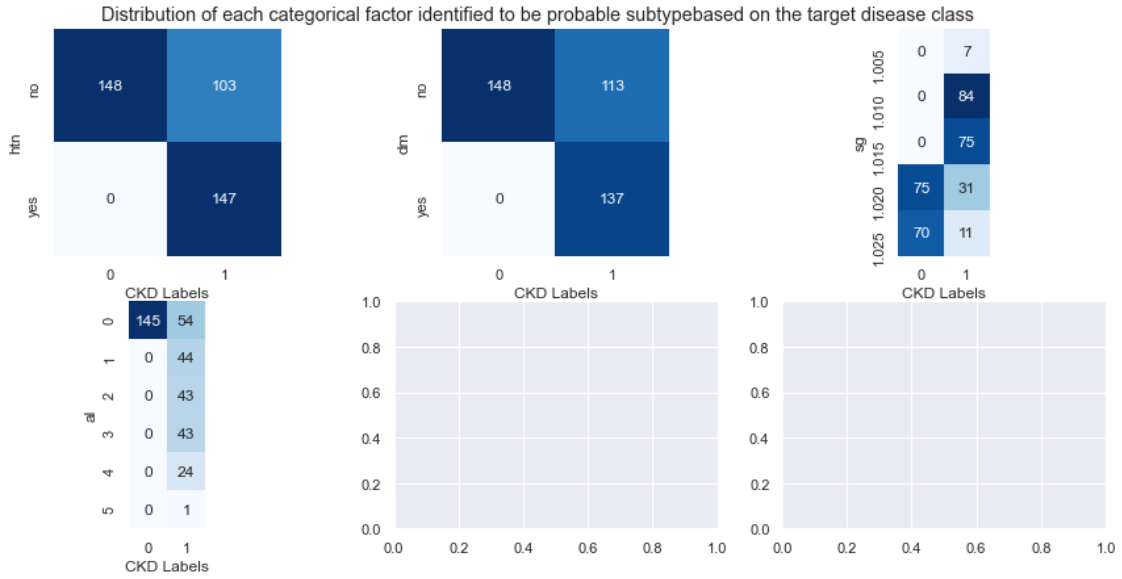
[42]:
```
model = RandomForestClassifier(criterion='entropy', random_state=47)
imp_features.remove('class')
print(f'Running model on features: {imp_features}')
X_pca_imp = data[imp_features]
results_df_pca_imp = train_eval(model, X_pca_imp, Y)
model_Res = pd.concat([results_df_all,results_df, results_df_pca,␣
 ↪results_df_pca_imp], axis=0)
model_Res
```

```
Running model on features: ['sg', 'dm', 'bu', 'al', 'htn', 'rbcc', 'hemo',
'pcv']
```

[42]:

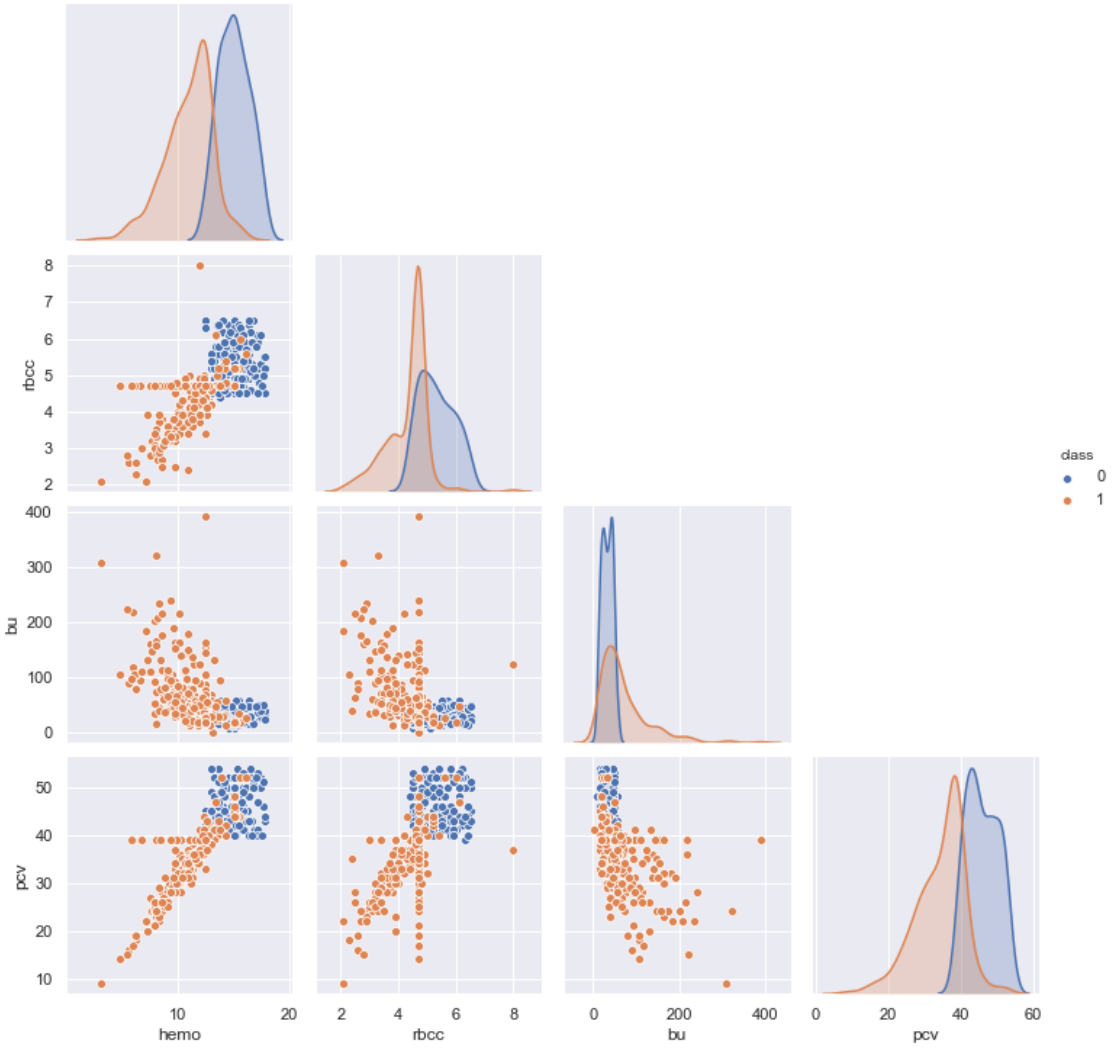
	Accuracy	ROC	Precision	Recall	F1 Score	R2	Feature Count
0	0.983333	0.977778	0.974026	1.000000	0.986842	0.928889	24
0	0.983333	0.977778	0.974026	1.000000	0.986842	0.928889	8
0	0.983333	0.986667	1.000000	0.973333	0.986486	0.928889	2
0	0.991667	0.988889	0.986842	1.000000	0.993377	0.964444	8

[43]:
```
cat_imp = list(set(imp_features).intersection(categories))

m = int(np.ceil(len(cat_imp)/3))
n = 3
fig, axes = plt.subplots(m,n, squeeze=False)
fig.suptitle('\n\nDistribution of each categorical factor identified to be␣
 ↪probable subtype\
based on the target disease class')
for i in range(m):
    for j in range(n):
        if i*n+j == len(cat_imp):
            break
        categ = cat_imp[i*n+j]
        sns.heatmap(pd.crosstab(raw_data[categ],raw_data['class']),
                    ax=axes[i,j],
                    cmap='Blues',
                    square='True',
                    cbar=False,
                    annot=True,
                   fmt='d')

        axes[i,j].set_ylabel(categ)
        axes[i,j].set_xlabel("CKD Labels")
plt.savefig('test.png')
```



[45]:
```
num_imp = list(set(imp_features).intersection(numerics))

imp_features = num_imp + ['class']
data_x = data[imp_features]
pairplot = sns.pairplot(data_x,corner=True, hue='class');
pairplot.fig.suptitle("Distribution of each numerical factor identified to be␣
 ↪probable subtype\
colored on the target disease class", y=1.08)
pairplot.savefig('test2.png')
```

Distribution of each numerical factor identified to be probable subtypecolored on the target disease class



[ ]: