CHRONIC KIDNEY DISEASE PROJECT

IMPORT LIBRARIES

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
df = pd.read_csv("Chronic_Kidney_Disease.csv",na_values=["\t?","\t43","\t6200","\t8400","\t84","?"])
```

In [3]:

```
df
```

Out[3]:

	Age	Blood_Pressure	Specific_Gravity	Albumin	Sugar	Red_Blood_Cells	Pus_Cell	Pus_Cell_clumps	Bacteria	Blood_Glucose_Random	...	Packe
0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	121.0	...	
1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	NaN	...	
2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	423.0	...	
3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	117.0	...	
4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	106.0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
395	55.0	80.0	1.020	0.0	0.0	normal	normal	notpresent	notpresent	140.0	...	
396	42.0	70.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	75.0	...	
397	12.0	80.0	1.020	0.0	0.0	normal	normal	notpresent	notpresent	100.0	...	
398	17.0	60.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	114.0	...	
399	58.0	80.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	131.0	...	

400 rows × 25 columns

INFORMATION OF DATA

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 25 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Age                      385 non-null    float64
 1   Blood_Pressure           388 non-null    float64
 2   Specific_Gravity         353 non-null    float64
 3   Albumin                  354 non-null    float64
 4   Sugar                    351 non-null    float64
 5   Red_Blood_Cells          248 non-null    object 
 6   Pus_Cell                 335 non-null    object 
 7   Pus_Cell_clumps          396 non-null    object 
 8   Bacteria                 396 non-null    object 
 9   Blood_Glucose_Random     355 non-null    float64
 10  Blood_Urea               381 non-null    float64
 11  Serum_Creatinine         383 non-null    float64
 12  Sodium                   313 non-null    float64
 13  Potassium                312 non-null    float64
 14  Hemoglobin               348 non-null    float64
 15  Packed_Cell_Volume       314 non-null    float64
 16  White_Blood_Cell_Count   285 non-null    float64
 17  Red_Blood_Cell_Count     269 non-null    float64
 18  Hypertension             398 non-null    object 
 19  Diabetes                 398 non-null    object 
 20  Coronary_Artery_Disease  398 non-null    object 
 21  Appetite                 399 non-null    object 
 22  Pedal_Edema              399 non-null    object 
 23  Anemia                   399 non-null    object 
 24  Class                    400 non-null    object 
dtypes: float64(14), object(11)
memory usage: 78.2+ KB
```

Checked null values

In [5]:

```
df.Age.unique()
```

Out[5]:

```
array([48.,  7., 62., 51., 60., 68., 24., 52., 53., 50., 63., 40., 47.,
       61., 21., 42., 75., 69., nan, 73., 70., 65., 76., 72., 82., 46.,
       45., 35., 54., 11., 59., 67., 15., 55., 44., 26., 64., 56.,  5.,
       74., 38., 58., 71., 34., 17., 12., 41., 57.,  8., 39., 66., 81.,
       14., 27., 83., 30.,  4.,  3.,  6., 32., 80., 49., 90., 78., 19.,
        2., 33., 36., 37., 23., 25., 20., 29., 28., 22., 79.])
```

In [6]:

```
df.Blood_Pressure.unique()
```

Out[6]:

```
array([ 80.,  50.,  70.,  90.,  nan, 100.,  60., 110., 140., 180., 120.])
```

In [7]:

```
df.Specific_Gravity.unique()
```

Out[7]:

```
array([1.02 , 1.01 , 1.005, 1.015,   nan, 1.025])
```

In [8]:

```
df.Albumin.unique()
```

Out[8]:

```
array([ 1.,  4.,  2.,  3.,  0., nan,  5.])
```

In [9]:

```
df.Sugar.unique()
```

Out[9]:

```
array([ 0.,  3.,  4.,  1., nan,  2.,  5.])
```

In [10]:

```
df.Red_Blood_Cells.unique()
```

Out[10]:

```
array([nan, 'normal', 'abnormal'], dtype=object)
```

In [11]:

```
df.Pus_Cell.unique()
```

Out[11]:

```
array(['normal', 'abnormal', nan], dtype=object)
```

In [12]:

```
df.Pus_Cell_clumps.unique()
```

Out[12]:

```
array(['notpresent', 'present', nan], dtype=object)
```

In [13]:

```
df.Bacteria.unique()
```

Out[13]:

```
array(['notpresent', 'present', nan], dtype=object)
```

In [14]:

```
df.Blood_Glucose_Random.unique()
```

Out[14]:

```
array([121.,  nan, 423., 117., 106.,  74., 100., 410., 138.,  70., 490.,
       380., 208.,  98., 157.,  76.,  99., 114., 263., 173.,  95., 108.,
       156., 264., 123.,  93., 107., 159., 140., 171., 270.,  92., 137.,
       204.,  79., 207., 124., 144.,  91., 162., 246., 253., 141., 182.,
        86., 150., 146., 425., 112., 250., 360., 163., 129., 133., 102.,
       158., 165., 132., 104., 127., 415., 169., 251., 109., 280., 210.,
       219., 295.,  94., 172., 101., 298., 153.,  88., 226., 143., 115.,
        89., 297., 233., 294., 323., 125.,  90., 308., 118., 224., 128.,
       122., 214., 213., 268., 256., 105., 288., 139.,  78., 273., 242.,
       424., 303., 148., 160., 192., 307., 220., 447., 309.,  22., 111.,
       261., 215., 234., 131., 352.,  80., 239., 110., 130., 184., 252.,
       113., 230., 341., 255., 103., 238., 248., 120., 241., 269., 201.,
       203., 463., 176.,  82., 119.,  97.,  96.,  81., 116., 134.,  85.,
        83.,  87.,  75.])
```

In [15]:

```
df.Packed_Cell_Volume.unique()
```

Out[15]:

```
array([44., 38., 31., 32., 35., 39., 36., 33., 29., 28., nan, 16., 24.,
       37., 30., 34., 40., 45., 27., 48., 52., 14., 22., 18., 42., 17.,
       46., 23., 19., 25., 41., 26., 15., 21., 20., 47.,  9., 49., 50.,
       53., 51., 54.])
```

In [16]:

```
df.White_Blood_Cell_Count.unique()
```

Out[16]:

```
array([ 7800.,  6000.,  7500.,  6700.,  7300.,    nan,  6900.,  9600.,
       12100.,  4500., 12200., 11000.,  3800., 11400.,  5300.,  9200.,
        8300., 10300.,  9800.,  9100.,  7900.,  6400.,  8600., 18900.,
       21600.,  4300.,  8500., 11300.,  7200.,  7700., 14600.,  6300.,
        7100., 11800.,  9400.,  5500.,  5800., 13200., 12500.,  5600.,
        7000., 11900., 10400., 10700., 12700.,  6800.,  6500., 13600.,
       10200.,  9000., 14900.,  8200., 15200.,  5000., 16300., 12400.,
       10500.,  4200.,  4700., 10900.,  8100.,  9500.,  2200., 12800.,
       11200., 19100., 12300., 16700.,  2600., 26400.,  8800.,  7400.,
        4900.,  8000., 12000., 15700.,  4100.,  5700., 11500.,  5400.,
       10800.,  9900.,  5200.,  5900.,  9300.,  9700.,  5100.,  6600.])
```

In [17]:

```
df.Red_Blood_Cell_Count.unique()
```

Out[17]:

```
array([5.2, nan, 3.9, 4.6, 4.4, 5. , 4. , 3.7, 3.8, 3.4, 2.6, 2.8, 4.3,
       3.2, 3.6, 4.1, 4.9, 2.5, 4.2, 4.5, 3.1, 4.7, 3.5, 6. , 2.1, 5.6,
       2.3, 2.9, 2.7, 8. , 3.3, 3. , 2.4, 4.8, 5.4, 6.1, 6.2, 6.3, 5.1,
       5.8, 5.5, 5.3, 6.4, 5.7, 5.9, 6.5])
```

In [18]:

```
df.Hypertension.unique()
```

Out[18]:

```
array(['yes', 'no', nan], dtype=object)
```

In [19]:

```
df.Diabetes.unique()
```

Out[19]:

```
array(['yes', 'no', '\tno', '\tyes', nan], dtype=object)
```

In [20]:

```
df.Diabetes.value_counts()
```

Out[20]:

```
no       260
yes      133
\tno       3
\tyes      2
Name: Diabetes, dtype: int64
```

In [21]:

```
df.replace("\tno","no",inplace=True)
df.replace("\tyes","yes",inplace=True)
```

In [22]:

```
df.Diabetes.value_counts()
```

Out[22]:

```
no     263
yes    135
Name: Diabetes, dtype: int64
```

In [23]:

```
df.Coronary_Artery_Disease.unique()
```

Out[23]:

```
array(['no', 'yes', nan], dtype=object)
```

In [24]:

```
df.Appetite.unique()
```

Out[24]:

```
array(['good', 'poor', nan], dtype=object)
```

In [25]:

```
df.Pedal_Edema.unique()
```

Out[25]:

```
array(['no', 'yes', nan], dtype=object)
```

In [26]:

```
df. Anemia.unique()
```

Out[26]:

```
array(['no', 'yes', nan], dtype=object)
```

In [27]:

```
df.Class.unique()
```

Out[27]:

```
array(['ckd', 'ckd\t', 'notckd'], dtype=object)
```

In [28]:

```
df.replace("ckd\t","ckd",inplace=True)
```

In [29]:

```
df.Class.unique()
```

Out[29]:

```
array(['ckd', 'notckd'], dtype=object)
```

FILLED MISSING VALUES OF NUMERIC COLUMN

In [30]:

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

In [31]:

```
si = SimpleImputer(missing_values=np.nan,strategy="mean")
```

In [32]:

```
df[["Age","Blood_Pressure","Specific_Gravity","Albumin","Sugar","Blood_Glucose_Random","Blood_Urea","Serum_Creatinine","Sodium","Potassiu
```

In [33]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 25 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Age                      400 non-null    float64
 1   Blood_Pressure           400 non-null    float64
 2   Specific_Gravity         400 non-null    float64
 3   Albumin                  400 non-null    float64
 4   Sugar                    400 non-null    float64
 5   Red_Blood_Cells          248 non-null    object 
 6   Pus_Cell                 335 non-null    object 
 7   Pus_Cell_clumps          396 non-null    object 
 8   Bacteria                 396 non-null    object 
 9   Blood_Glucose_Random     400 non-null    float64
 10  Blood_Urea               400 non-null    float64
 11  Serum_Creatinine         400 non-null    float64
 12  Sodium                   400 non-null    float64
 13  Potassium                400 non-null    float64
 14  Hemoglobin               400 non-null    float64
 15  Packed_Cell_Volume       400 non-null    float64
 16  White_Blood_Cell_Count   400 non-null    float64
 17  Red_Blood_Cell_Count     400 non-null    float64
 18  Hypertension             398 non-null    object 
 19  Diabetes                 398 non-null    object 
 20  Coronary_Artery_Disease  398 non-null    object 
 21  Appetite                 399 non-null    object 
 22  Pedal_Edema              399 non-null    object 
 23  Anemia                   399 non-null    object 
 24  Class                    400 non-null    object 
dtypes: float64(14), object(11)
memory usage: 78.2+ KB
```

FILLED MISSING VALUES OF CATEGORICAL COLUMN

In [34]:

```
si2 = SimpleImputer(missing_values=np.nan,strategy="most_frequent")
```

In [35]:

```
df[["Red_Blood_Cells","Pus_Cell","Pus_Cell_clumps","Bacteria","Hypertension","Diabetes","Coronary_Artery_Disease","Appetite","Pedal_Edema
```

In [36]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 25 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Age                     400 non-null    float64
 1   Blood_Pressure          400 non-null    float64
 2   Specific_Gravity        400 non-null    float64
 3   Albumin                 400 non-null    float64
 4   Sugar                   400 non-null    float64
 5   Red_Blood_Cells         400 non-null    object 
 6   Pus_Cell                400 non-null    object 
 7   Pus_Cell_clumps         400 non-null    object 
 8   Bacteria                400 non-null    object 
 9   Blood_Glucose_Random    400 non-null    float64
 10  Blood_Urea              400 non-null    float64
 11  Serum_Creatinine        400 non-null    float64
 12  Sodium                  400 non-null    float64
 13  Potassium               400 non-null    float64
 14  Hemoglobin              400 non-null    float64
 15  Packed_Cell_Volume      400 non-null    float64
 16  White_Blood_Cell_Count  400 non-null    float64
 17  Red_Blood_Cell_Count    400 non-null    float64
 18  Hypertension            400 non-null    object 
 19  Diabetes                400 non-null    object 
 20  Coronary_Artery_Disease 400 non-null    object 
 21  Appetite                400 non-null    object 
 22  Pedal_Edema             400 non-null    object 
 23  Anemia                  400 non-null    object 
 24  Class                   400 non-null    object 
dtypes: float64(14), object(11)
memory usage: 78.2+ KB
```

In [37]:

```
df.describe()
```

Out[37]:

	Age	Blood_Pressure	Specific_Gravity	Albumin	Sugar	Blood_Glucose_Random	Blood_Urea	Serum_Creatinine	Sodium	Potassi
count	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000
mean	51.615584	76.469072	1.017408	1.016949	0.450142	148.216901	57.425722	3.072454	137.528754	4.627
std	16.942562	13.476298	0.005369	1.272318	1.029487	74.713694	49.285887	5.617490	9.204273	2.819
min	2.000000	50.000000	1.005000	0.000000	0.000000	22.000000	1.500000	0.400000	4.500000	2.500
25%	42.000000	70.000000	1.015000	0.000000	0.000000	101.750000	27.000000	0.900000	135.000000	4.000
50%	54.000000	78.234536	1.017408	1.000000	0.000000	127.000000	44.000000	1.400000	137.528754	4.627
75%	64.000000	80.000000	1.020000	2.000000	0.450142	150.000000	61.750000	3.072454	141.000000	4.800
max	90.000000	180.000000	1.025000	5.000000	5.000000	490.000000	391.000000	76.000000	163.000000	47.000

In [38]:

```
df.columns
```

Out[38]:

```
Index(['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', 'Hemoglobin', 'Packed_Cell_Volume',
       'White_Blood_Cell_Count', 'Red_Blood_Cell_Count', 'Hypertension',
       'Diabetes', 'Coronary_Artery_Disease', 'Appetite', 'Pedal_Edema',
       'Anemia', 'Class'],
      dtype='object')
```

DATA SPLITING INTO FEATURES AND TARGET

In [39]:

```
x = df.iloc[:,[0,1,4,8,14,16,17,18,19,20,21,23]]
```

In [40]:

```
x
```

Out[40]:

	Age	Blood_Pressure	Sugar	Bacteria	Hemoglobin	White_Blood_Cell_Count	Red_Blood_Cell_Count	Hypertension	Diabetes	Coronary_Artery_Disea
0	48.0	80.0	0.0	notpresent	15.4	7800.0	5.200000	yes	yes	
1	7.0	50.0	0.0	notpresent	11.3	6000.0	4.707435	no	no	
2	62.0	80.0	3.0	notpresent	9.6	7500.0	4.707435	no	yes	
3	48.0	70.0	0.0	notpresent	11.2	6700.0	3.900000	yes	no	
4	51.0	80.0	0.0	notpresent	11.6	7300.0	4.600000	no	no	
...	...	...	...	...	...	...	...	...	...	
395	55.0	80.0	0.0	notpresent	15.7	6700.0	4.900000	no	no	
396	42.0	70.0	0.0	notpresent	16.5	7800.0	6.200000	no	no	
397	12.0	80.0	0.0	notpresent	15.8	6600.0	5.400000	no	no	
398	17.0	60.0	0.0	notpresent	14.2	7200.0	5.900000	no	no	
399	58.0	80.0	0.0	notpresent	15.8	6800.0	6.100000	no	no	

400 rows × 12 columns

In [41]:

```
y = df.iloc[:,-1]
```

In [42]:

```
y
```

Out[42]:

```
0         ckd
1         ckd
2         ckd
3         ckd
4         ckd
        ...
395    notckd
396    notckd
397    notckd
398    notckd
399    notckd
Name: Class, Length: 400, dtype: object
```

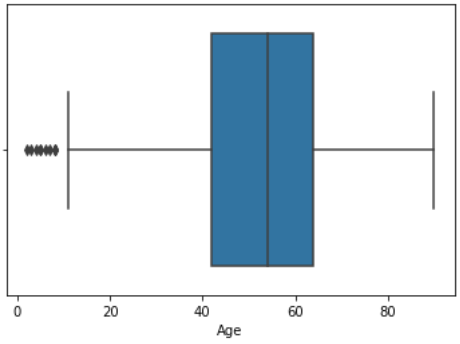
OUTLIERS OF FEATURES

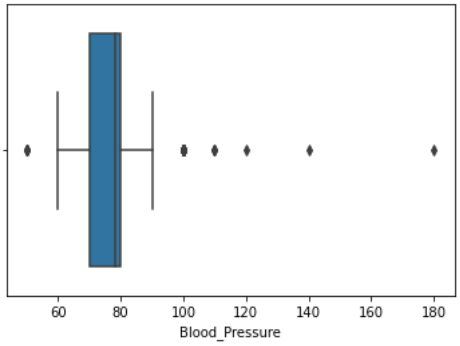
In [43]:

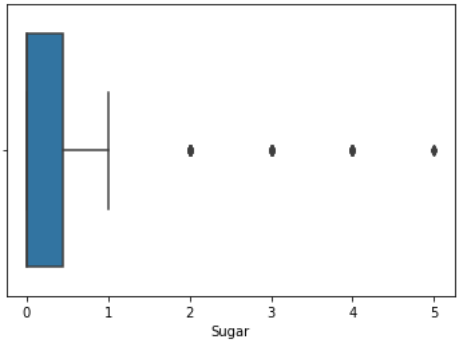
```
colname = x.select_dtypes("float64").columns
```

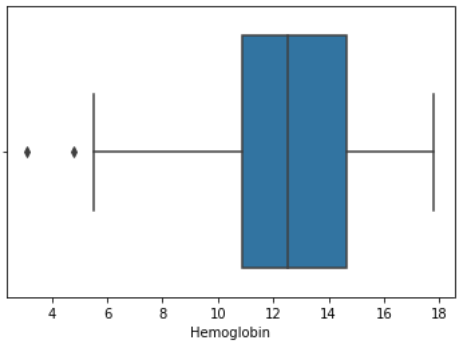
In [44]:

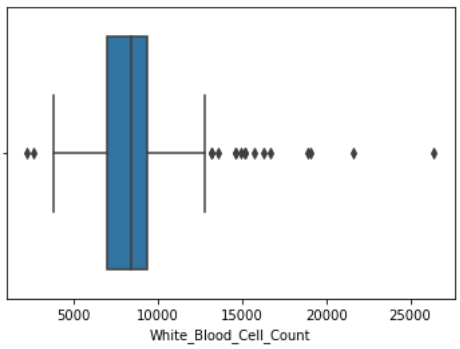
```
for col in colname:
    plt.figure()
    sns.boxplot(data=df,x=col)
    plt.show()
```

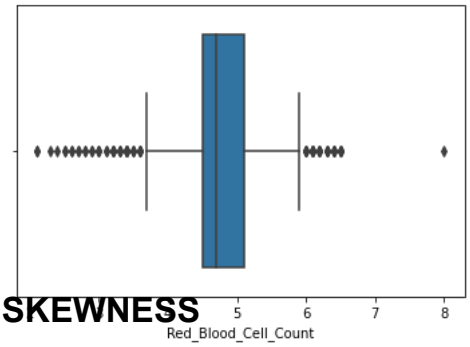












SKEWNESS

In [45]:

```
colname = x.select_dtypes("float64").columns
```

In [46]:

```
colname
```

Out[46]:

```
Index(['Age', 'Blood_Pressure', 'Sugar', 'Hemoglobin',
       'White_Blood_Cell_Count', 'Red_Blood_Cell_Count'],
      dtype='object')
```

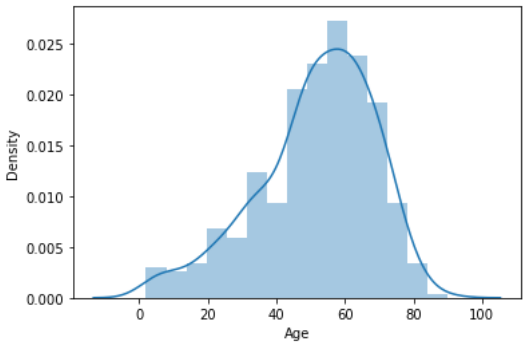
In [47]:

```
from scipy.stats import skew
```

In [48]:

```
for col in x[colname]:
    print(col)
    print(skew(x[col]))
    plt.figure()
    sns.distplot(x[col])
    plt.show()
```

```
Age
-0.6987313258428666
```




```
Blood_Pressure
```

In [50]:

```
df["Class"].value_counts()
```

Out[50]:

```
ckd       250
notckd    150
Name: Class, dtype: int64
```

In [51]:

```
df["Class"]
```

Out[51]:

```
0         ckd
1         ckd
2         ckd
3         ckd
4         ckd
        ...
395    notckd
396    notckd
397    notckd
398    notckd
399    notckd
Name: Class, Length: 400, dtype: object
```

GRAPHICAL REPRESENTATION

In [52]:

```
df['Hypertension'].value_counts()
```

Out[52]:

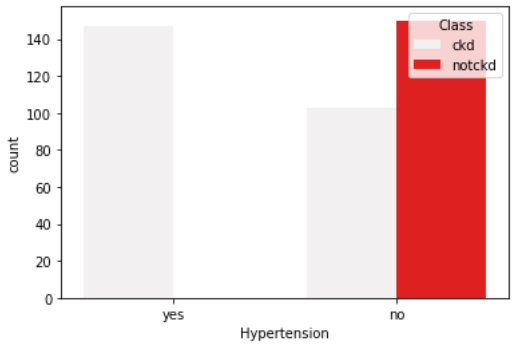
```
no     253
yes    147
Name: Hypertension, dtype: int64
```

In [53]:

```
sns.countplot(df['Hypertension'],hue=df['Class'],color="red")
```

Out[53]:

```
<AxesSubplot:xlabel='Hypertension', ylabel='count'>
```

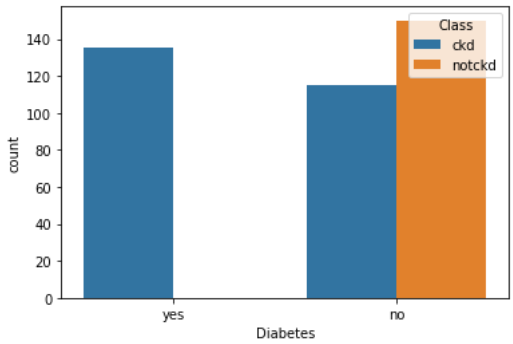


In [54]:

```
sns.countplot(df['Diabetes'],hue=df['Class'])
```

Out[54]:

```
<AxesSubplot:xlabel='Diabetes', ylabel='count'>
```

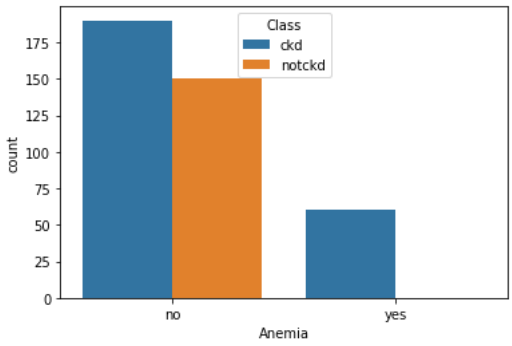


In [55]:

```
sns.countplot(df['Anemia'],hue=df['Class'])
```

Out[55]:

```
<AxesSubplot:xlabel='Anemia', ylabel='count'>
```

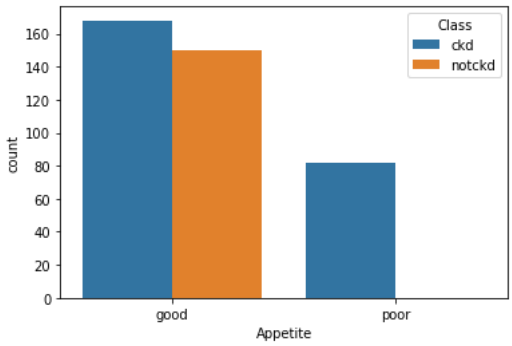


In [56]:

```
sns.countplot(df['Appetite'],hue=df['Class'])
```

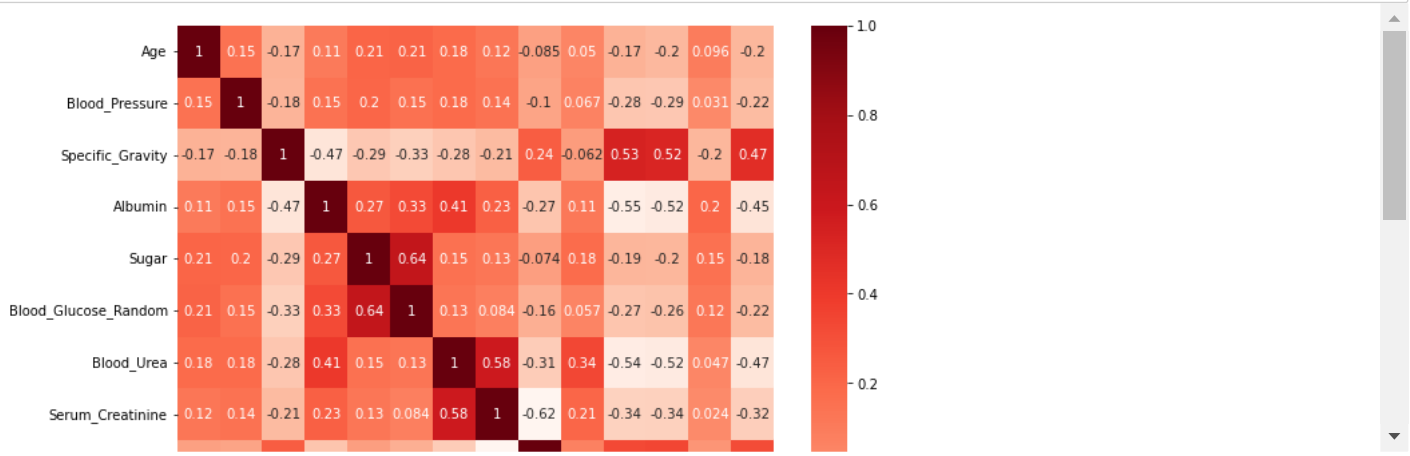
Out[56]:

```
<AxesSubplot:xlabel='Appetite', ylabel='count'>
```



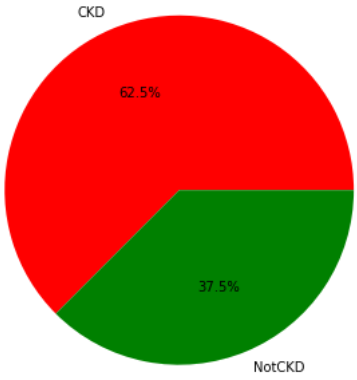
In [57]:

```
plt.figure(figsize=(10,10))
sns.heatmap(df.corr(),annot=True,cmap="Reds")
plt.show()
```



In [58]:

```
plt.pie(df["Class"].value_counts(), labels=["CKD","NotCKD"], colors=["red","green"], autopct="%1.1f%%",radius=1.5)
plt.show()
```

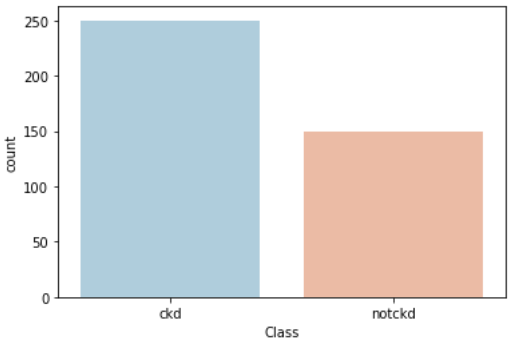


In [59]:

```
sns.countplot(data=df, x="Class", palette="RdBu_r")
```

Out[59]:

```
<AxesSubplot:xlabel='Class', ylabel='count'>
```



In [60]:

```
x.select_dtypes(object)
```

Out[60]:

	Bacteria	Hypertension	Diabetes	Coronary_Artery_Disease	Appetite	Anemia
0	notpresent	yes	yes	no	good	no
1	notpresent	no	no	no	good	no
2	notpresent	no	yes	no	poor	yes
3	notpresent	yes	no	no	poor	yes
4	notpresent	no	no	no	good	no
...	...	...	...	...	...	...
395	notpresent	no	no	no	good	no
396	notpresent	no	no	no	good	no
397	notpresent	no	no	no	good	no
398	notpresent	no	no	no	good	no
399	notpresent	no	no	no	good	no

400 rows × 6 columns

In [61]:

```
from sklearn.preprocessing import OrdinalEncoder
oe = OrdinalEncoder()
x[["Bacteria","Hypertension","Diabetes","Coronary_Artery_Disease","Appetite","Anemia"]]=oe.fit_transform(x[["Bacteria","Hypertension","Dia
```

SPLITING DATA INTO TRAINING AND TESTING

In [62]:

```
from sklearn.model_selection import train_test_split

xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size=0.3,random_state=1,stratify=y)
```

IMPORTED CLASSIFICATION ALGORITHM

In [63]:

```
x
```

Out[63]:

	Age	Blood_Pressure	Sugar	Bacteria	Hemoglobin	White_Blood_Cell_Count	Red_Blood_Cell_Count	Hypertension	Diabetes	Coronary_Artery_Diseas
0	48.0	80.0	0.0	0.0	15.4	7800.0	5.200000	1.0	1.0	0.
1	7.0	50.0	0.0	0.0	11.3	6000.0	4.707435	0.0	0.0	0.
2	62.0	80.0	3.0	0.0	9.6	7500.0	4.707435	0.0	1.0	0.
3	48.0	70.0	0.0	0.0	11.2	6700.0	3.900000	1.0	0.0	0.
4	51.0	80.0	0.0	0.0	11.6	7300.0	4.600000	0.0	0.0	0.
...	...	...	...	...	...	...	...	...	...	.
395	55.0	80.0	0.0	0.0	15.7	6700.0	4.900000	0.0	0.0	0.
396	42.0	70.0	0.0	0.0	16.5	7800.0	6.200000	0.0	0.0	0.
397	12.0	80.0	0.0	0.0	15.8	6600.0	5.400000	0.0	0.0	0.
398	17.0	60.0	0.0	0.0	14.2	7200.0	5.900000	0.0	0.0	0.
399	58.0	80.0	0.0	0.0	15.8	6800.0	6.100000	0.0	0.0	0.

400 rows × 12 columns

In [64]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

In [65]:

```
from sklearn.metrics import confusion_matrix,classification_report
```

In [66]:

```
def mymodel(model):
    model.fit(xtrain,ytrain)
    ypred = model.predict(xtest)

    train = model.score(xtrain,ytrain)
    test = model.score(xtest,ytest)

    cr = classification_report(ytest,ypred)
    cm = confusion_matrix(ytest,ypred)

    print(f"Training Score:{train}\nTesting Score:{test}\nClassification Report:\n{cr}\n{cm}")
```

In [67]:

```
lr = mymodel(LogisticRegression())
```

```
Training Score:0.9321428571428572
Testing Score:0.925
Classification Report:
              precision    recall  f1-score   support

         ckd       0.93      0.95      0.94        75
      notckd       0.91      0.89      0.90        45

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

[[71  4]
 [ 5 40]]
```

SCALING ON FEATURES

In [68]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x = sc.fit_transform(x)
```

In [69]:

```
x
```

Out[69]:

```
array([[-0.21366972,  0.26233836, -0.4377969 , ..., -0.30478874,
        -0.50780078, -0.42008403],
       [-2.63664137, -1.96658024, -0.4377969 , ..., -0.30478874,
        -0.50780078, -0.42008403],
       [ 0.61368645,  0.26233836,  2.47992547, ..., -0.30478874,
         1.96927621,  2.38047614],
       ...,
       [-2.34115702,  0.26233836, -0.4377969 , ..., -0.30478874,
        -0.50780078, -0.42008403],
       [-2.04567267, -1.22360737, -0.4377969 , ..., -0.30478874,
        -0.50780078, -0.42008403],
       [ 0.37729897,  0.26233836, -0.4377969 , ..., -0.30478874,
        -0.50780078, -0.42008403]])
```

In [70]:

```
sv = mymodel(SVC())
```

```
Training Score:0.625
Testing Score:0.625
Classification Report:
              precision    recall  f1-score   support

         ckd       0.62      1.00      0.77        75
      notckd       0.00      0.00      0.00        45

    accuracy                           0.62       120
   macro avg       0.31      0.50      0.38       120
weighted avg       0.39      0.62      0.48       120

[[75  0]
 [45  0]]
```

In [71]:

```
knn = mymodel(KNeighborsClassifier())
```

```
Training Score:0.7928571428571428
Testing Score:0.6833333333333333
Classification Report:
              precision    recall  f1-score   support

         ckd       0.76      0.72      0.74        75
      notckd       0.57      0.62      0.60        45

    accuracy                           0.68       120
   macro avg       0.67      0.67      0.67       120
weighted avg       0.69      0.68      0.69       120

[[54 21]
 [17 28]]
```

In [72]:

```
dt = mymodel(DecisionTreeClassifier())
```

```
Training Score:1.0
Testing Score:0.925
Classification Report:
              precision    recall  f1-score   support

         ckd       0.93      0.95      0.94        75
      notckd       0.91      0.89      0.90        45

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

[[71  4]
 [ 5 40]]
```

In [73]:

```
rf = mymodel(RandomForestClassifier())
```

```
Training Score:1.0
Testing Score:0.9833333333333333
Classification Report:
              precision    recall  f1-score   support

         ckd       0.97      1.00      0.99        75
      notckd       1.00      0.96      0.98        45

    accuracy                           0.98       120
   macro avg       0.99      0.98      0.98       120
weighted avg       0.98      0.98      0.98       120

[[75  0]
 [ 2 43]]
```

HYPERPARAMATER TUNNING

In [74]:

```
parameters = {
    "criterion" : ["gini","entropy"],
    "max_depth" : list(range(1,10)),
    "min_samples_leaf" : list(range(1,10))
}
```

In [75]:

```
from sklearn.model_selection import GridSearchCV
```

In [76]:

```
grid = GridSearchCV(RandomForestClassifier(),parameters,verbose=2,cv=5,scoring="accuracy")
```

In [77]:

```
grid.fit(xtrain,ytrain)
```

```
Fitting 5 folds for each of 162 candidates, totalling 810 fits
[CV] END ....criterion=gini, max_depth=1, min_samples_leaf=1; total time=   0.0s
[CV] END ....criterion=gini, max_depth=1, min_samples_leaf=1; total time=   0.1s
[CV] END ....criterion=gini, max_depth=1, min_samples_leaf=1; total time=   0.0s
[CV] END ....criterion=gini, max_depth=1, min_samples_leaf=1; total time=   0.0s
[CV] END ....criterion=gini, max_depth=1, min_samples_leaf=1; total time=   0.1s
[CV] END ....criterion=gini, max_depth=1, min_samples_leaf=2; total time=   0.1s
[CV] END ....criterion=gini, max_depth=1, min_samples_leaf=2; total time=   0.0s
[CV] END ....criterion=gini, max_depth=1, min_samples_leaf=2; total time=   0.0s
[CV] END ....criterion=gini, max_depth=1, min_samples_leaf=2; total time=   0.0s
[CV] END ....criterion=gini, max_depth=1, min_samples_leaf=2; total time=   0.0s
[CV] END ....criterion=gini, max_depth=1, min_samples_leaf=3; total time=   0.0s
[CV] END ....criterion=gini, max_depth=1, min_samples_leaf=3; total time=   0.0s
[CV] END ....criterion=gini, max_depth=1, min_samples_leaf=3; total time=   0.1s
[CV] END ....criterion=gini, max_depth=1, min_samples_leaf=3; total time=   0.0s
[CV] END ....criterion=gini, max_depth=1, min_samples_leaf=3; total time=   0.0s
[CV] END ....criterion=gini, max_depth=1, min_samples_leaf=4; total time=   0.0s
[CV] END ....criterion=gini, max_depth=1, min_samples_leaf=4; total time=   0.1s
[CV] END ....criterion=gini, max_depth=1, min_samples_leaf=4; total time=   0.0s
```

In [78]:

```
grid.best_params_
```

Out[78]:

```
{'criterion': 'gini', 'max_depth': 4, 'min_samples_leaf': 1}
```

In [79]:

```
grid.best_score_
```

Out[79]:

```
0.975
```

In [80]:

```
grid.best_estimator_
```

Out[80]:

```
RandomForestClassifier(max_depth=4)
```

In [81]:

```
rf1 = mymodel(grid.best_estimator_)
```

```
Training Score:0.9821428571428571
Testing Score:0.975
Classification Report:
              precision    recall  f1-score   support

         ckd       0.96      1.00      0.98        75
      notckd       1.00      0.93      0.97        45

    accuracy                           0.97       120
   macro avg       0.98      0.97      0.97       120
weighted avg       0.98      0.97      0.97       120

[[75  0]
 [ 3 42]]
```

NEW OBSERVATION

In [82]:

```
def predictckd():
    age = float(input('Enter your age:-'))
    bp = float(input('Enter your blood pressure:-'))
    sugar = float(input('Enter sugar range(0-5):-'))
    bac = input('Enter input in present  or notpresent if you have bacteria:-')
    hemo = float(input('Enter hemoglobin:-'))
    wbcc = float(input('Enter wbc count:-'))
    rbcc = float(input('Enter rbc count:-'))
    hypert = input('Do you have hypertension??')
    diab = input('Do tou have diabetes??')
    cad = input('Do you have coronary artery disease??')
    apet = input('How is your appetite good or poor??')
    anemia = input('Do you have anemia problem')

    rf1 = RandomForestClassifier()
    rf1.fit(xtrain,ytrain)
    ypred = rf1.predict(xtest)

    train = rf1.score(xtrain,ytrain)
    test = rf1.score(xtest,ytest)


    newob=[age,bp,sugar,bac,hemo,wbcc,rbcc,hypert,diab,cad,apet,anemia]
    newob[3],newob[7],newob[8],newob[9],newob[10],newob[11]=oe.transform([[newob[3],newob[7],newob[8],newob[9],newob[10],newob[11]]])[0]
    p=rf1.predict([newob])[0]

    if p==1:
        print(f'You do not have chronic kidney disease')
        return p
    else:
        print(f'You have chronic kidney disease')
        return p
```

In [83]:

```
xtrain
```

	Blood_Pressure	Sugar	Bacteria	Hemoglobin	White_Blood_Cell_Count	Red_Blood_Cell_Count	Hypertension	Diabetes	Coronary_Artery_Disease	Appetite	Anemia
)	50.0	0.000000	0.0	9.600000	15700.000000	3.800000	0.0	1.0	0.0	0.0	0.0
)	70.0	0.450142	0.0	12.526437	8444.912281	4.707435	1.0	0.0	0.0	0.0	0.0
)	90.0	0.000000	1.0	8.300000	12400.000000	3.900000	0.0	0.0	0.0	0.0	1.0
)	70.0	0.000000	0.0	13.100000	11200.000000	4.707435	0.0	0.0	0.0	0.0	0.0
)	60.0	0.000000	0.0	15.000000	7000.000000	5.200000	0.0	0.0	0.0	0.0	0.0
.	...	...	...	...	...	...	...	...	...	...	...
)	70.0	0.450142	0.0	7.900000	8444.912281	4.707435	1.0	1.0	1.0	0.0	1.0
)	90.0	0.450142	0.0	11.500000	8444.912281	4.707435	1.0	1.0	1.0	0.0	0.0
)	90.0	0.000000	0.0	12.200000	7800.000000	4.400000	1.0	1.0	0.0	0.0	0.0

In [85]:

```
predictckd()
```

```
Enter your age:-60
Enter your blood pressure:-90
Enter sugar range(0-5):-0
Enter input in present  or notpresent if you have bacteria:-present
Enter hemoglobin:-12.200000
Enter wbc count:-7800.000000
Enter rbc count:-4.400000
Do you have hypertension??no
Do tou have diabetes??no
Do you have coronary artery disease??yes
How is your appetite good or poor??good
Do you have anemia problemyes
You have chronic kidney disease
```

Out[85]:

```
'ckd'
```