MANOJ KUMAR - 2048015

P9: KNN Classifier

Implement KNN classier and experiment with the following concepts

Check the performance of KNN with different distance measures
Check the performance of KNN with binary, categorical & numerical data

Importing basic libraries

In [1]:
```
import pandas as pd
import numpy as np
import time
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib as mpl
```

Reading the dataset

In [2]:
```
ckd_df = pd.read_csv('kidney_disease.csv')

#Check the shape
print(ckd_df.shape)
```

```
(400, 26)
```

In [3]:
```
#check the columns
ckd_df.columns
```

Out[3]:
```
Index(['id', 'age', 'bp', 'sg', 'al', 'su', 'rbc', 'pc', 'pcc', 'ba', 'bgr',
       'bu', 'sc', 'sod', 'pot', 'hemo', 'pcv', 'wc', 'rc', 'htn', 'dm', 'cad',
       'appet', 'pe', 'ane', 'classification'],
      dtype='object')
```

Rename the columns to have meaningful names

In [4]:
```
col_dict={"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",
          "wc":"white_blood_cell_count",
          "rc":"red_blood_cell_count",
          "htn":"hypertension",
          "dm":"diabetes_mellitus",
          "cad":"coronary_artery_disease",
          "appet":"appetite",
          "pe":"pedal_edema",
          "ane":"anemia"}

ckd_df.rename(columns=col_dict, inplace=True)

#Check the column names again
ckd_df.columns
```

Out[4]:
```
Index(['id', '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_mellitus', 'coronary_artery_disease', 'appetite',
       'pedal_edema', 'anemia', 'classification'],
      dtype='object')
```

Observing the data

```
In [5]: ckd_df.head(11).T
```

Out[5]:

	0	1	2	3	4	5	6	7	8	9	1
id	0	1	2	3	4	5	6	7	8	9	1
age	48	7	62	48	51	60	68	24	52	53	5
blood_pressure	80	50	80	70	80	90	70	NaN	100	90	6
specific_gravity	1.02	1.02	1.01	1.005	1.01	1.015	1.01	1.015	1.015	1.02	1.01
albumin	1	4	2	4	2	3	0	2	3	2	
sugar	0	0	3	0	0	0	0	4	0	0	
red_blood_cells	NaN	NaN	normal	normal	normal	NaN	NaN	normal	normal	abnormal	Na
pus_cell	normal	normal	normal	abnormal	normal	NaN	normal	abnormal	abnormal	abnormal	abnorma
pus_cell_clumps	notpresent	notpresent	notpresent	present	notpresent	notpresent	notpresent	notpresent	present	present	presen
bacteria	notpresent	notpresent	notpresent	notpresent	notpresent	notpresent	notpresent	notpresent	notpresent	notpresent	notpresen
blood_glucose_random	121	NaN	423	117	106	74	100	410	138	70	49
blood_urea	36	18	53	56	26	25	54	31	60	107	5
serum_creatinine	1.2	0.8	1.8	3.8	1.4	1.1	24	1.1	1.9	7.2	
sodium	NaN	NaN	NaN	111	NaN	142	104	NaN	NaN	114	Na
potassium	NaN	NaN	NaN	2.5	NaN	3.2	4	NaN	NaN	3.7	Na
hemoglobin	15.4	11.3	9.6	11.2	11.6	12.2	12.4	12.4	10.8	9.5	9.
packed_cell_volume	44	38	31	32	35	39	36	44	33	29	2
white_blood_cell_count	7800	6000	7500	6700	7300	7800	NaN	6900	9600	12100	Na
red_blood_cell_count	5.2	NaN	NaN	3.9	4.6	4.4	NaN	5	4.0	3.7	Na
hypertension	yes	no	no	yes	no	yes	no	no	yes	yes	ye
diabetes_mellitus	yes	no	yes	no	no	yes	no	yes	yes	yes	ye
coronary_artery_disease	no	no	no	no	no	no	no	no	no	no	n
appetite	good	good	poor	poor	good	good	good	good	good	poor	goo
pedal_edema	no	no	no	yes	no	yes	no	yes	no	no	n
anemia	no	no	yes	yes	no	no	no	no	yes	yes	ye
classification	ckd	ckd	ckd	ckd	ckd	ckd	ckd	ckd	ckd	ckd	ck

Data DeepDive

```
In [6]: for i in ckd_df.drop("id",axis=1).columns:
            print('unique values in "{}":\n'.format(i),ckd_df[i].unique())
```

```
unique values in "age":
 [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. 43. 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.]
unique values in "blood_pressure":
 [ 80.  50.  70.  90.  nan 100.  60. 110. 140. 180. 120.]
unique values in "specific_gravity":
 [1.02  1.01  1.005 1.015   nan 1.025]
unique values in "albumin":
 [ 1.  4.  2.  3.  0. nan  5.]
unique values in "sugar":
 [ 0.  3.  4.  1. nan  2.  5.]
unique values in "red_blood_cells":
 [nan 'normal' 'abnormal']
unique values in "pus_cell":
 ['normal' 'abnormal' nan]
unique values in "pus_cell_clumps":
 ['notpresent' 'present' nan]
```

In [7]:
```python
#Replace incorrect values
ckd_df['diabetes_mellitus'] =ckd_df['diabetes_mellitus'].replace(to_replace={'\tno':'no','\tyes':'yes',' yes':'y
ckd_df['coronary_artery_disease'] = ckd_df['coronary_artery_disease'].replace(to_replace='\tno',value='no')
ckd_df['white_blood_cell_count'] = ckd_df['white_blood_cell_count'].replace(to_replace='\t8400',value='8400')
ckd_df["classification"]=ckd_df["classification"].replace("ckd\t", "ckd")

for i in range(ckd_df.shape[0]):
    if ckd_df.iloc[i,16]=='\t?':
        ckd_df.iloc[i,16]=np.nan
    if ckd_df.iloc[i,16]=='\t43':
        ckd_df.iloc[i,16]='43'
    if ckd_df.iloc[i,17]=='\t?':
        ckd_df.iloc[i,17]=np.nan
    if ckd_df.iloc[i,17]=='\t6200':
        ckd_df.iloc[i,17]= '6200'
    if ckd_df.iloc[i,18]=='\t?':
        ckd_df.iloc[i,18]=np.nan
    if ckd_df.iloc[i,25]=='ckd':
        ckd_df.iloc[i,25]='1'
    if ckd_df.iloc[i,25]=='notckd':
        ckd_df.iloc[i,25]='0'

for i in ckd_df.drop("id",axis=1).columns:
    print('unique values in "{}":\n'.format(i),ckd_df[i].unique())
```

```
unique values in "age":
 [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. 43. 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.]
unique values in "blood_pressure":
 [ 80.  50.  70.  90.  nan 100.  60. 110. 140. 180. 120.]
unique values in "specific_gravity":
 [1.02  1.01  1.005 1.015   nan 1.025]
unique values in "albumin":
 [ 1.  4.  2.  3.  0. nan  5.]
unique values in "sugar":
 [ 0.  3.  4.  1. nan  2.  5.]
unique values in "red_blood_cells":
 [nan 'normal' 'abnormal']
unique values in "pus_cell":
 ['normal' 'abnormal' nan]
unique values in "pus_cell_clumps":
 ['notpresent' 'present' nan]
unique values in "bacteria":
 ['notpresent' 'present' nan]
unique values in "blood_glucose_random":
 [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.  84. 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.]
unique values in "blood_urea":
 [ 36.   18.   53.   56.   26.   25.   54.   31.   60.  107.   55.   72.
  86.   90.  162.   46.   87.   27.  148.  180.  163.    nan  50.   75.
  45.   28.  155.   33.   39.  153.   29.   65.  103.   70.   80.   20.
 202.   77.   89.   24.   17.   32.  114.   66.   38.  164.  142.   96.
 391.   15.  111.   73.   19.   92.   35.   16.  139.   48.   85.   98.
 186.   37.   47.   52.   82.   51.  106.   22.  217.   88.  118.   50.1
  71.   34.   40.   21.  219.   30.  125.  166.   49.  208.  176.   68.
 145.  165.  322.   23.  235.  132.   76.   42.   44.   41.  113.    1.5
 146.   58.  133.  137.   67.  115.  223.   98.6 158.   94.   74.  150.
  61.   57.   95.  191.   93.  241.   64.   79.  215.  309.   10. ]
unique values in "serum_creatinine":
 [ 1.2   0.8   1.8   3.8   1.4   1.1  24.    1.9   7.2   4.    2.7   2.1
   4.6   4.1   9.6   2.2   5.2   1.3   1.6   3.9  76.    7.7    nan  2.4
   7.3   1.5   2.5   2.    3.4   0.7   1.   10.8   6.3   5.9   0.9   3.
   3.25  9.7   6.4   3.2  32.    0.6   6.1   3.3   6.7   8.5   2.8  15.
   2.9   1.7   3.6   5.6   6.5   4.4  10.2  11.5   0.5  12.2   5.3   9.2
  13.8  16.9   6.    7.1  18.    2.3  13.   48.1  14.2  16.4   2.6   7.5
   4.3  18.1  11.8   9.3   6.8  13.5  12.8  11.9  12.   13.4  15.2  13.3
   0.4 ]
unique values in "sodium":
 [  nan 111.  142.  104.  114.  131.  138.  135.  130.  141.  139.    4.5
 136.  129.  140.  132.  133.  134.  125.  163.  137.  128.  143.  127.
 146.  126.  122.  147.  124.  115.  145.  113.  120.  150.  144. ]
unique values in "potassium":
 [ nan  2.5  3.2  4.   3.7  4.2  5.8  3.4  6.4  4.9  4.1  4.3  5.2  3.8
  4.6  3.9  4.7  5.9  4.8  4.4  6.6 39.   5.5  5.   3.5  3.6  7.6  2.9
  4.5  5.7  5.4  5.3 47.   6.3  5.1  5.6  3.   2.8  2.7  6.5  3.3]
unique values in "hemoglobin":
 [15.4 11.3  9.6 11.2 11.6 12.2 12.4 10.8  9.5  9.4  9.7  9.8  5.6  7.6
 12.6 12.1 12.7 10.3  7.7 10.9  nan 11.1  9.9 12.5 12.9 10.1 12.  13.
  7.9  9.3 15.  10.   8.6 13.6 10.2 10.5  6.6 11.   7.5 15.6 15.2  4.8
  9.1  8.1 11.9 13.5  8.3  7.1 16.1 10.4  9.2  6.2 13.9 14.1  6.  11.8
 11.7 11.4 14.   8.2 13.2  6.1  8.  12.3  8.4 14.3  9.   8.7 10.6 13.1
 10.7  5.5  5.8  6.8  8.8  8.5 13.8 11.5  7.3 13.7 12.8 13.4  6.3  3.1
 17.  15.9 14.5 15.5 16.2 14.4 14.2 16.3 14.8 16.5 15.7 13.3 14.6 16.4
 16.9 16.  14.7 16.6 14.9 16.7 16.8 15.8 15.1 17.1 17.2 15.3 17.3 17.4
 17.7 17.8 17.5 17.6]
unique values in "packed_cell_volume":
 ['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' '43' '20' '47' '9' '49' '50' '53' '51' '54']
unique values in "white_blood_cell_count":
 ['7800' '6000' '7500' '6700' '7300' nan '6900' '9600' '12100' '4500'
 '12200' '11000' '3800' '11400' '5300' '9200' '6200' '8300' '8400' '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']
unique values in "red_blood_cell_count":
 ['5.2' nan '3.9' '4.6' '4.4' '5' '4.0' '3.7' '3.8' '3.4' '2.6' '2.8' '4.3'
 '3.2' '3.6' '4' '4.1' '4.9' '2.5' '4.2' '4.5' '3.1' '4.7' '3.5' '6.0'
 '5.0' '2.1' '5.6' '2.3' '2.9' '2.7' '8.0' '3.3' '3.0' '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']
unique values in "hypertension":
 ['yes' 'no' nan]
unique values in "diabetes_mellitus":
 ['yes' 'no' nan]
unique values in "coronary_artery_disease":
 ['no' 'yes' nan]
unique values in "appetite":
 ['good' 'poor' nan]
unique values in "pedal_edema":
 ['no' 'yes' nan]
unique values in "anemia":
 ['no' 'yes' nan]
unique values in "classification":
 ['1' '0']
```

In [8]:
```
# Observing the summarized information of data
ckd_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 26 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       400 non-null    int64  
 1   age                      391 non-null    float64
 2   blood_pressure           388 non-null    float64
 3   specific_gravity         353 non-null    float64
 4   albumin                  354 non-null    float64
 5   sugar                    351 non-null    float64
 6   red_blood_cells          248 non-null    object 
 7   pus_cell                 335 non-null    object 
 8   pus_cell_clumps          396 non-null    object 
 9   bacteria                 396 non-null    object 
 10  blood_glucose_random     356 non-null    float64
 11  blood_urea               381 non-null    float64
 12  serum_creatinine         383 non-null    float64
 13  sodium                   313 non-null    float64
 14  potassium                312 non-null    float64
 15  hemoglobin               348 non-null    float64
 16  packed_cell_volume       329 non-null    object 
 17  white_blood_cell_count   294 non-null    object 
 18  red_blood_cell_count     269 non-null    object 
 19  hypertension             398 non-null    object 
 20  diabetes_mellitus        398 non-null    object 
 21  coronary_artery_disease  398 non-null    object 
 22  appetite                 399 non-null    object 
 23  pedal_edema              399 non-null    object 
 24  anemia                   399 non-null    object 
 25  classification           400 non-null    object 
dtypes: float64(11), int64(1), object(14)
memory usage: 81.4+ KB
```

In [9]:
```python
ckd_df.iloc[:,-1]=ckd_df.iloc[:,-1].astype('int64')
ckd_df.head(11).T
```

Out[9]:

	0	1	2	3	4	5	6	7	8	9	10
id	0	1	2	3	4	5	6	7	8	9	10
age	48	7	62	48	51	60	68	24	52	53	50
blood_pressure	80	50	80	70	80	90	70	NaN	100	90	60
specific_gravity	1.02	1.02	1.01	1.005	1.01	1.015	1.01	1.015	1.015	1.02	1.01
albumin	1	4	2	4	2	3	0	2	3	2	2
sugar	0	0	3	0	0	0	0	4	0	0	4
red_blood_cells	NaN	NaN	normal	normal	normal	NaN	NaN	normal	normal	abnormal	NaN
pus_cell	normal	normal	normal	abnormal	normal	NaN	normal	abnormal	abnormal	abnormal	abnormal
pus_cell_clumps	notpresent	notpresent	notpresent	present	notpresent	notpresent	notpresent	notpresent	present	present	present
bacteria	notpresent	notpresent	notpresent	notpresent	notpresent	notpresent	notpresent	notpresent	notpresent	notpresent	notpresent
blood_glucose_random	121	NaN	423	117	106	74	100	410	138	70	490
blood_urea	36	18	53	56	26	25	54	31	60	107	55
serum_creatinine	1.2	0.8	1.8	3.8	1.4	1.1	24	1.1	1.9	7.2	4
sodium	NaN	NaN	NaN	111	NaN	142	104	NaN	NaN	114	NaN
potassium	NaN	NaN	NaN	2.5	NaN	3.2	4	NaN	NaN	3.7	NaN
hemoglobin	15.4	11.3	9.6	11.2	11.6	12.2	12.4	12.4	10.8	9.5	9.4
packed_cell_volume	44	38	31	32	35	39	36	44	33	29	28
white_blood_cell_count	7800	6000	7500	6700	7300	7800	NaN	6900	9600	12100	NaN
red_blood_cell_count	5.2	NaN	NaN	3.9	4.6	4.4	NaN	5	4.0	3.7	NaN
hypertension	yes	no	no	yes	no	yes	no	no	yes	yes	yes
diabetes_mellitus	yes	no	yes	no	no	yes	no	yes	yes	yes	yes
coronary_artery_disease	no	no	no	no	no	no	no	no	no	no	no
appetite	good	good	poor	poor	good	good	good	good	good	poor	good
pedal_edema	no	no	no	yes	no	yes	no	yes	no	no	no
anemia	no	no	yes	yes	no	no	no	no	yes	yes	yes
classification	1	1	1	1	1	1	1	1	1	1	1

In [10]:
```python
print(ckd_df['packed_cell_volume'].unique())
print(ckd_df['white_blood_cell_count'].unique())
print(ckd_df['red_blood_cell_count'].unique())
```

```
['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' '43' '20' '47' '9' '49' '50' '53' '51' '54']
['7800' '6000' '7500' '6700' '7300' nan '6900' '9600' '12100' '4500'
 '12200' '11000' '3800' '11400' '5300' '9200' '6200' '8300' '8400' '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']
['5.2' nan '3.9' '4.6' '4.4' '5' '4.0' '3.7' '3.8' '3.4' '2.6' '2.8' '4.3'
 '3.2' '3.6' '4' '4.1' '4.9' '2.5' '4.2' '4.5' '3.1' '4.7' '3.5' '6.0'
 '5.0' '2.1' '5.6' '2.3' '2.9' '2.7' '8.0' '3.3' '3.0' '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 [11]: mistyped=['packed_cell_volume','white_blood_cell_count','red_blood_cell_count']
         for col in mistyped:
                 ckd_df[col]=ckd_df[col].astype('float')

         numeric=[]
         for i in ckd_df.columns:
             if ckd_df[i].dtype=='float64':
                 numeric.append(i)

         numeric
```

```
Out[11]: ['age',
          'blood_pressure',
          'specific_gravity',
          'albumin',
          'sugar',
          'blood_glucose_random',
          'blood_urea',
          'serum_creatinine',
          'sodium',
          'potassium',
          'hemoglobin',
          'packed_cell_volume',
          'white_blood_cell_count',
          'red_blood_cell_count']
```

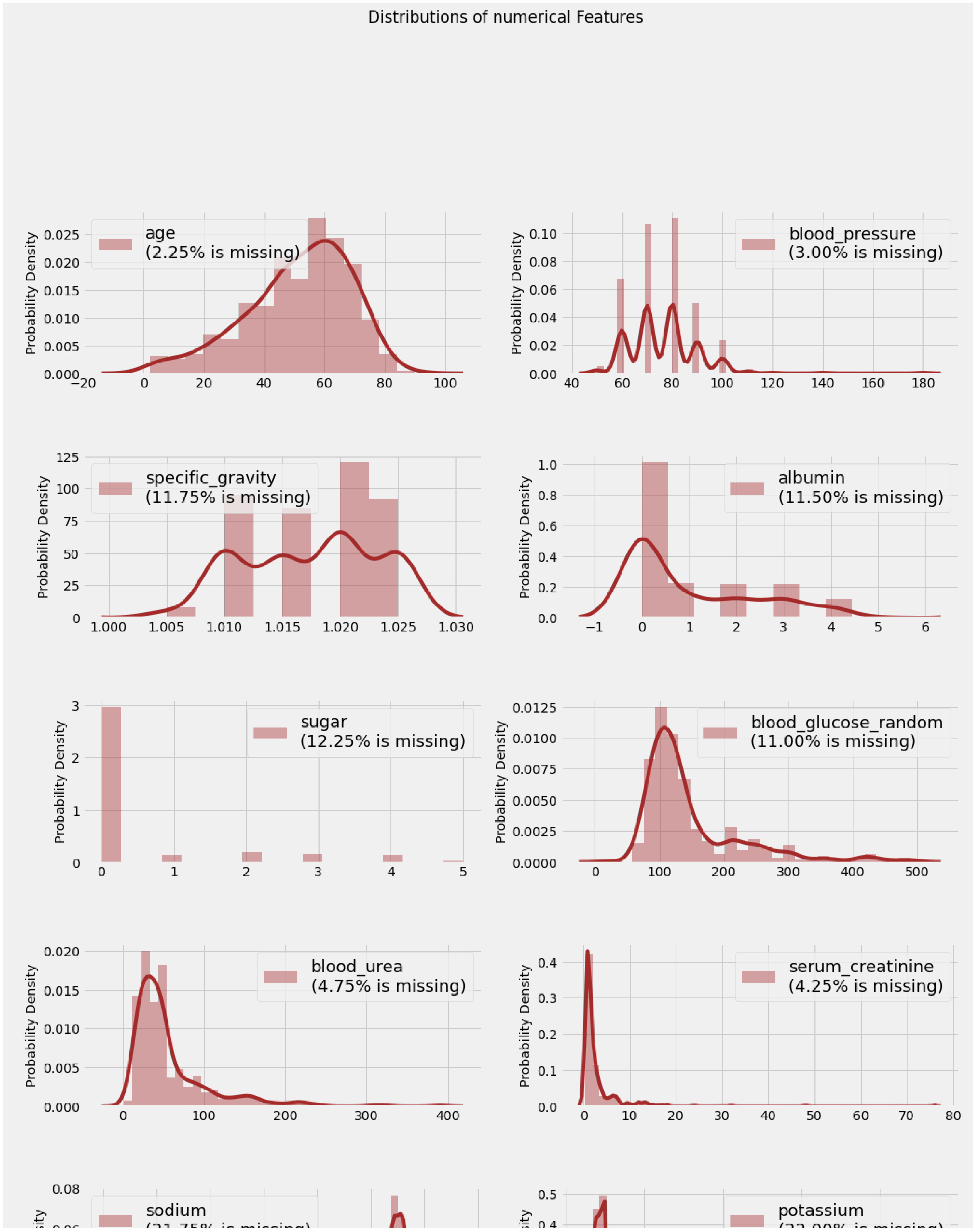
```
In [12]: ckd_df.drop('id',axis=1,inplace=True)

         categoricals=[]

         for col in ckd_df.columns:
             if not col in numeric:
                 categoricals.append(col)
         categoricals.remove('classification')

         categoricals
```

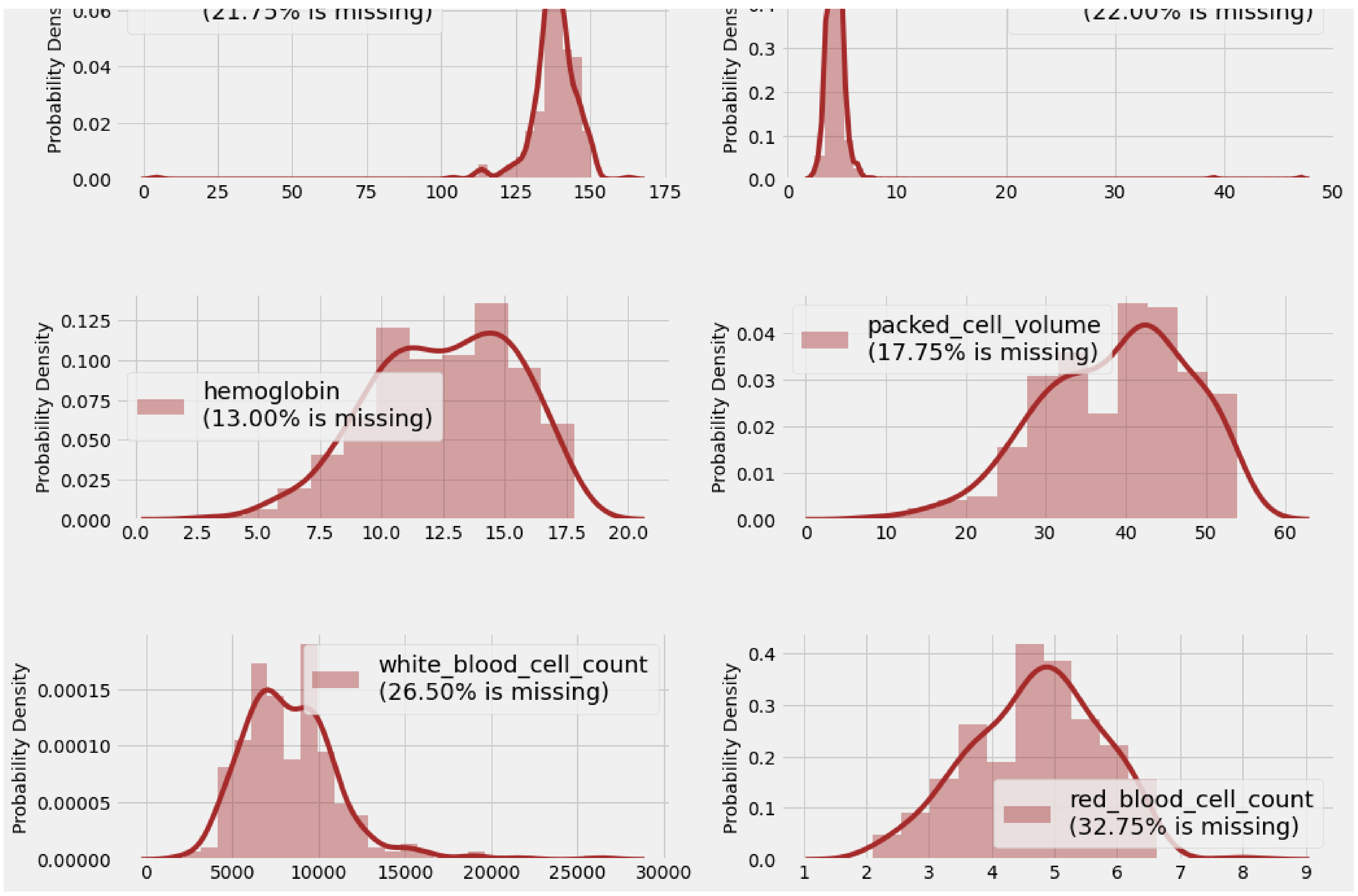
```
Out[12]: ['red_blood_cells',
          'pus_cell',
          'pus_cell_clumps',
          'bacteria',
          'hypertension',
          'diabetes_mellitus',
          'coronary_artery_disease',
          'appetite',
          'pedal_edema',
          'anemia']
```

```
In [13]: import warnings
         warnings.simplefilter('ignore')

         import matplotlib.style as style
         style.use('fivethirtyeight')
```

Checking distribution of the numerical features

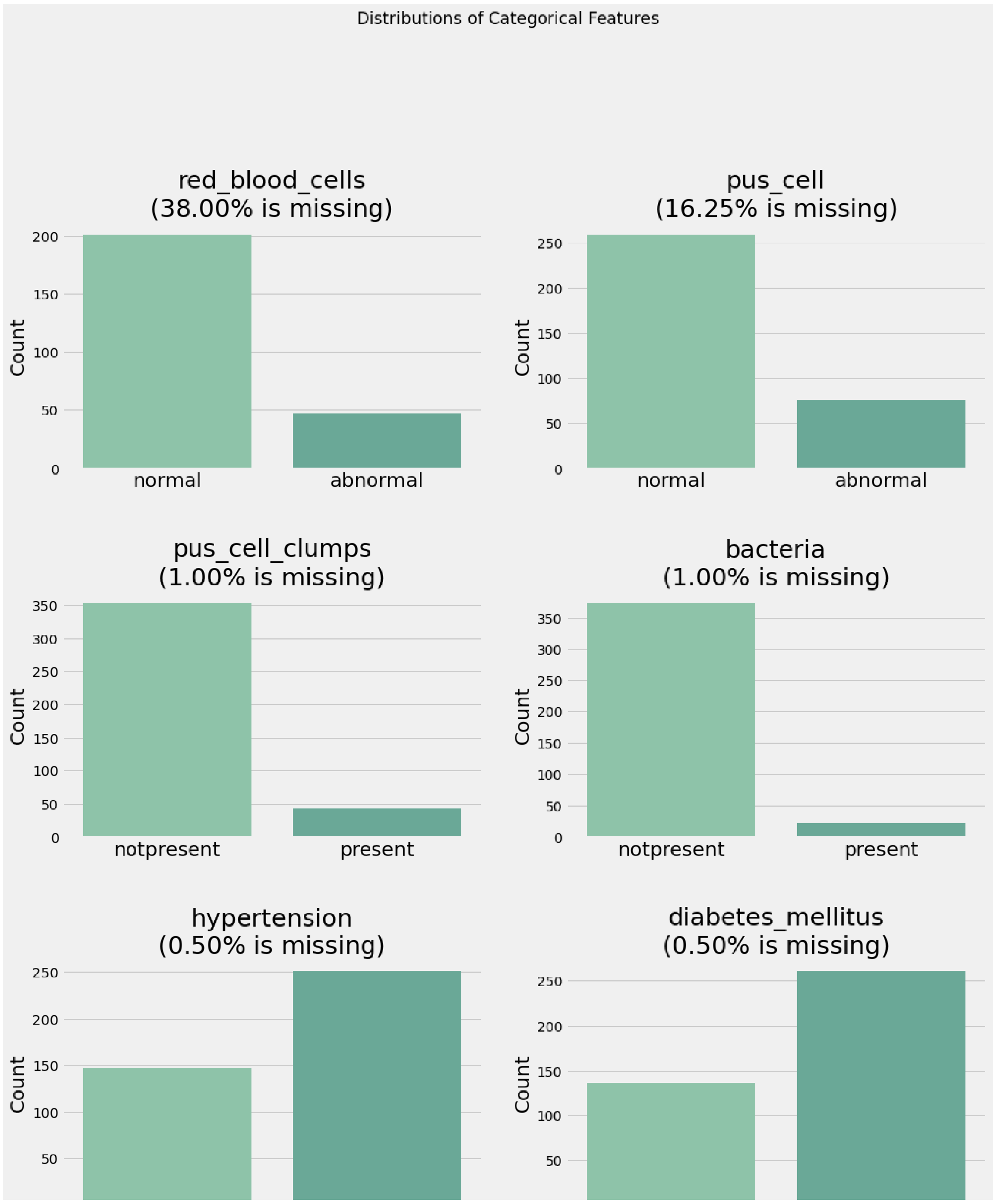
In [14]:
```python
fig, axes = plt.subplots(nrows=7, ncols=2, figsize=(15,30))
fig.subplots_adjust(hspace=0.5)
fig.suptitle('Distributions of numerical Features')

n_rows, n_cols = (7,2)

for index, column in enumerate(numeric):

    i,j = (index // n_cols), (index % n_cols)
    miss_perc="%.2f"%(100*(1-(ckd_df[column].dropna().shape[0])/ckd_df.shape[0]))
    collabel=column+"\n({}% is missing)".format(miss_perc)
    fig=sns.distplot(ckd_df[column], color="brown", label=collabel,
                     norm_hist=True, ax=axes[i,j], kde_kws={"lw":4})
    fig=fig.legend(loc='best', fontsize=18)

    axes[i,j].set_ylabel("Probability Density",fontsize='medium')
    axes[i,j].set_xlabel(None)

plt.show()
```

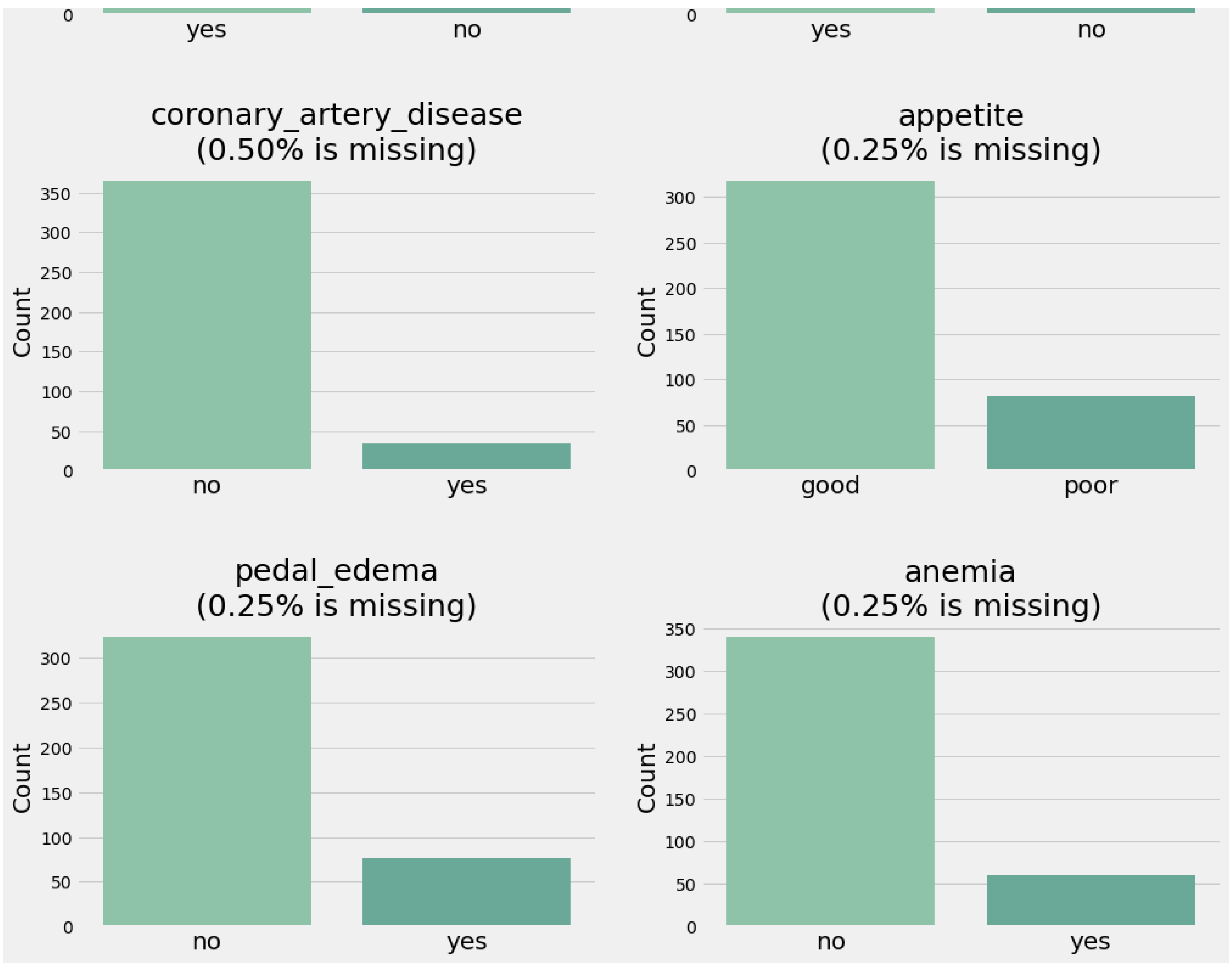




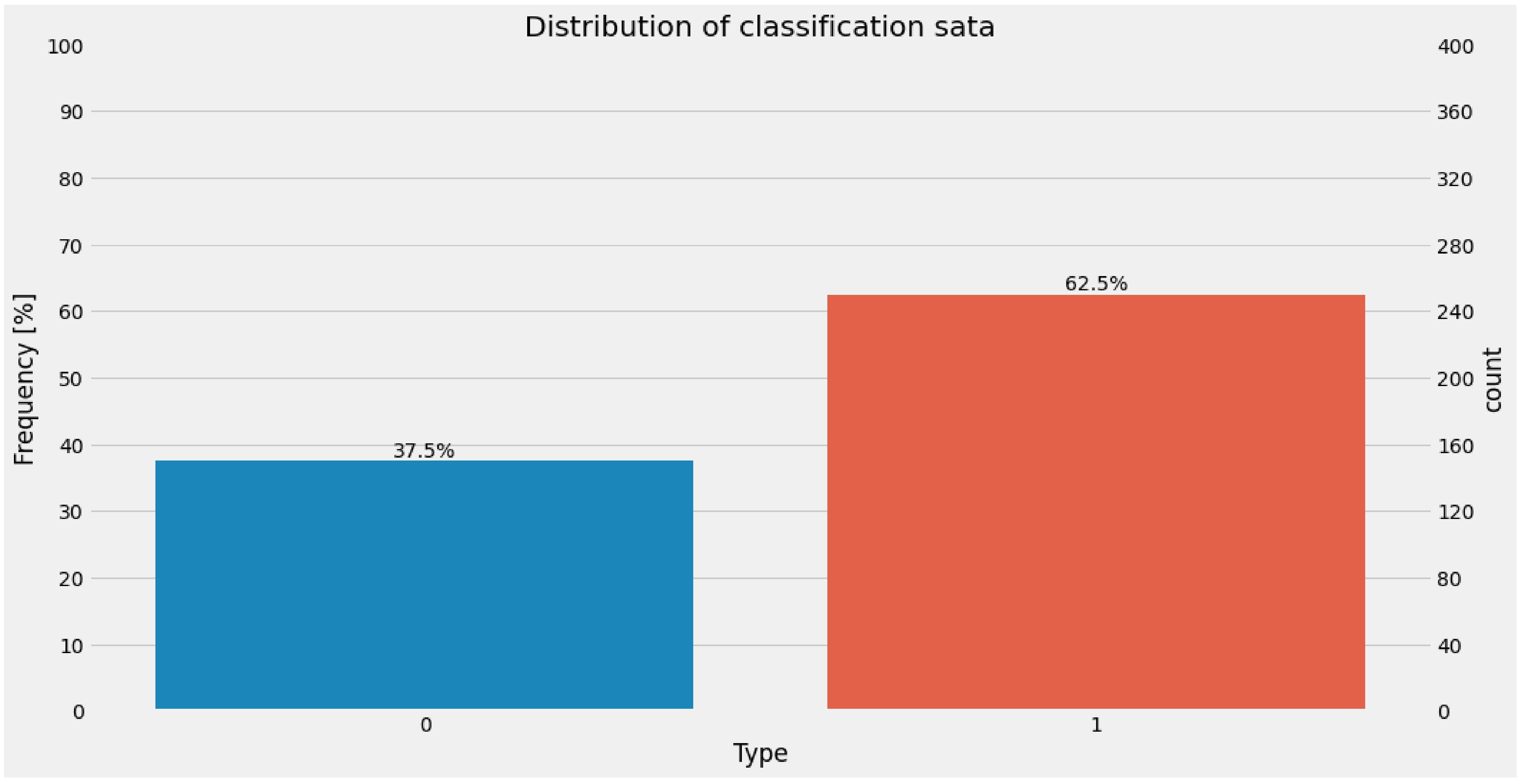
Checking distribution of the Categorical features

In [15]:
```
style.use('fivethirtyeight')

fig, axes = plt.subplots(nrows=5, ncols=2, figsize=(15,30))
fig.subplots_adjust(hspace=0.5)
fig.suptitle('Distributions of Categorical Features')

n_rows, n_cols = (5,2)

for index, column in enumerate(categoricals):

    i,j = index // n_cols, index % n_cols
    miss_perc="%.2f"%(100*(1-(ckd_df[column].dropna().shape[0])/ckd_df.shape[0]))
    collabel=column+"\n({}% is missing)".format(miss_perc)
    fig = sns.countplot(x=column, data=ckd_df,label=collabel,
                        palette=sns.cubehelix_palette(rot=-.4,light=0.7,hue=1), ax=axes[i,j])

    axes[i,j].set_title(collabel,fontsize=25)
    axes[i,j].set_xlabel(None)
    axes[i,j].set_ylabel("Count",fontsize=20)
    axes[i,j].set_xticklabels(axes[i,j].get_xticklabels(), Fontsize=20)

plt.show()
```




0
yes
no
0
yes
no
coronary_artery_disease
(0.50% is missing)
Count
350
300
250
200
150
100
50
0
no
yes
appetite
(0.25% is missing)
Count
300
250
200
150
100
50
0
good
poor
pedal_edema
(0.25% is missing)
Count
300
250
200
150
100
50
0
no
yes
anemia
(0.25% is missing)
Count
350
300
250
200
150
100
50
0
no
yes

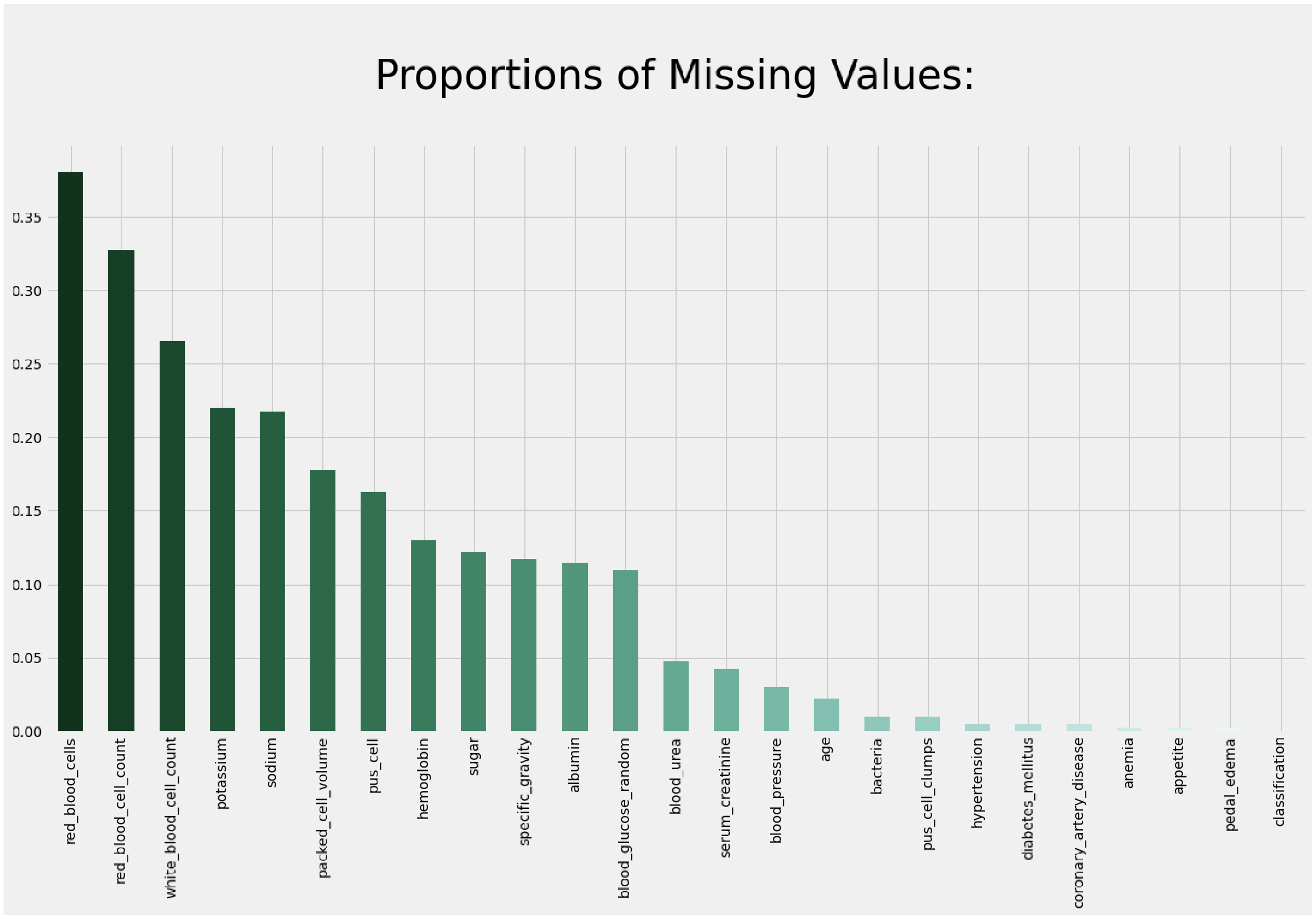
In [16]:
```python
import matplotlib.ticker as ticker
style.use('fivethirtyeight')
# Some random data
ncount = 400

plt.figure(figsize=(15,8))
ax = sns.countplot(x="classification", data=ckd_df)
plt.title('Distribution of classification sata')
plt.xlabel('Type')

# Make twin axis
ax2=ax.twinx()

# Switch so count axis is on right, frequency on left
ax2.yaxis.tick_left()
ax.yaxis.tick_right()

# Also switch the labels over
ax.yaxis.set_label_position('right')
ax2.yaxis.set_label_position('left')

ax2.set_ylabel('Frequency [%]')

for p in ax.patches:
    x=p.get_bbox().get_points()[:,0]
    y=p.get_bbox().get_points()[1,1]
    ax.annotate('{:.1f}%'.format(100.*y/ncount), (x.mean(), y),
            ha='center', va='bottom') # set the alignment of the text

# Use a LinearLocator to ensure the correct number of ticks
ax.yaxis.set_major_locator(ticker.LinearLocator(11))

# Fix the frequency range to 0-100
ax2.set_ylim(0,100)
ax.set_ylim(0,ncount)

# And use a MultipleLocator to ensure a tick spacing of 10
ax2.yaxis.set_major_locator(ticker.MultipleLocator(10))

# Need to turn the grid on ax2 off, otherwise the gridlines end up on top of the bars
ax2.grid(None)
```



In [17]:
```python
for i in range(ckd_df.shape[0]):
    if ckd_df.iloc[i,24]=='ckd':
        ckd_df.iloc[i,24]='1'
    if ckd_df.iloc[i,24]=='notckd':
        ckd_df.iloc[i,24]='0'
```

Missing Values

In [18]:
```python
style.use('fivethirtyeight')

d=((ckd_df.isnull().sum()/ckd_df.shape[0])).sort_values(ascending=False)
d.plot(kind='bar',
       color=sns.cubehelix_palette(start=2,
                                    rot=0.15,
                                    dark=0.15,
                                    light=0.95,
                                    reverse=True,
                                    n_colors=24),
        figsize=(20,10))
plt.title("\nProportions of Missing Values:\n",fontsize=40)
plt.show()
```



```
In [19]: ckd_df.head(10).T
```

Out[19]:

	0	1	2	3	4	5	6	7	8	9
age	48	7	62	48	51	60	68	24	52	53
blood_pressure	80	50	80	70	80	90	70	NaN	100	90
specific_gravity	1.02	1.02	1.01	1.005	1.01	1.015	1.01	1.015	1.015	1.02
albumin	1	4	2	4	2	3	0	2	3	2
sugar	0	0	3	0	0	0	0	4	0	0
red_blood_cells	NaN	NaN	normal	normal	normal	NaN	NaN	normal	normal	abnormal
pus_cell	normal	normal	normal	abnormal	normal	NaN	normal	abnormal	abnormal	abnormal
pus_cell_clumps	notpresent	notpresent	notpresent	present	notpresent	notpresent	notpresent	notpresent	present	present
bacteria	notpresent	notpresent	notpresent	notpresent	notpresent	notpresent	notpresent	notpresent	notpresent	notpresent
blood_glucose_random	121	NaN	423	117	106	74	100	410	138	70
blood_urea	36	18	53	56	26	25	54	31	60	107
serum_creatinine	1.2	0.8	1.8	3.8	1.4	1.1	24	1.1	1.9	7.2
sodium	NaN	NaN	NaN	111	NaN	142	104	NaN	NaN	114
potassium	NaN	NaN	NaN	2.5	NaN	3.2	4	NaN	NaN	3.7
hemoglobin	15.4	11.3	9.6	11.2	11.6	12.2	12.4	12.4	10.8	9.5
packed_cell_volume	44	38	31	32	35	39	36	44	33	29
white_blood_cell_count	7800	6000	7500	6700	7300	7800	NaN	6900	9600	12100
red_blood_cell_count	5.2	NaN	NaN	3.9	4.6	4.4	NaN	5	4	3.7
hypertension	yes	no	no	yes	no	yes	no	no	yes	yes
diabetes_mellitus	yes	no	yes	no	no	yes	no	yes	yes	yes
coronary_artery_disease	no	no	no	no	no	no	no	no	no	no
appetite	good	good	poor	poor	good	good	good	good	good	poor
pedal_edema	no	no	no	yes	no	yes	no	yes	no	no
anemia	no	no	yes	yes	no	no	no	no	yes	yes
classification	1	1	1	1	1	1	1	1	1	1

One-Hot Encoding

In [20]:
```
onehotdata=pd.get_dummies(ckd_df,drop_first=True,prefix_sep=': ')
onehotdata.head(13).T
```

Out[20]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
age	48.00	7.00	62.00	48.000	51.00	60.000	68.00	24.000	52.000	53.00	50.00	63.00	68.000
blood_pressure	80.00	50.00	80.00	70.000	80.00	90.000	70.00	NaN	100.000	90.00	60.00	70.00	70.000
specific_gravity	1.02	1.02	1.01	1.005	1.01	1.015	1.01	1.015	1.015	1.02	1.01	1.01	1.015
albumin	1.00	4.00	2.00	4.000	2.00	3.000	0.00	2.000	3.000	2.00	2.00	3.00	3.000
sugar	0.00	0.00	3.00	0.000	0.00	0.000	0.00	4.000	0.000	0.00	4.00	0.00	1.000
blood_glucose_random	121.00	NaN	423.00	117.000	106.00	74.000	100.00	410.000	138.000	70.00	490.00	380.00	208.000
blood_urea	36.00	18.00	53.00	56.000	26.00	25.000	54.00	31.000	60.000	107.00	55.00	60.00	72.000
serum_creatinine	1.20	0.80	1.80	3.800	1.40	1.100	24.00	1.100	1.900	7.20	4.00	2.70	2.100
sodium	NaN	NaN	NaN	111.000	NaN	142.000	104.00	NaN	NaN	114.00	NaN	131.00	138.000
potassium	NaN	NaN	NaN	2.500	NaN	3.200	4.00	NaN	NaN	3.70	NaN	4.20	5.800
hemoglobin	15.40	11.30	9.60	11.200	11.60	12.200	12.40	12.400	10.800	9.50	9.40	10.80	9.700
packed_cell_volume	44.00	38.00	31.00	32.000	35.00	39.000	36.00	44.000	33.000	29.00	28.00	32.00	28.000
white_blood_cell_count	7800.00	6000.00	7500.00	6700.000	7300.00	7800.000	NaN	6900.000	9600.000	12100.00	NaN	4500.00	12200.000
red_blood_cell_count	5.20	NaN	NaN	3.900	4.60	4.400	NaN	5.000	4.000	3.70	NaN	3.80	3.400
classification	1.00	1.00	1.00	1.000	1.00	1.000	1.00	1.000	1.000	1.00	1.00	1.00	1.000
red_blood_cells: normal	0.00	0.00	1.00	1.000	1.00	0.000	0.00	1.000	1.000	0.00	0.00	0.00	0.000
pus_cell: normal	1.00	1.00	1.00	0.000	1.00	0.000	1.00	0.000	0.000	0.00	0.00	0.00	1.000
pus_cell_clumps: present	0.00	0.00	0.00	1.000	0.00	0.000	0.00	0.000	1.000	1.00	1.00	1.00	1.000
bacteria: present	0.00	0.00	0.00	0.000	0.00	0.000	0.00	0.000	0.000	0.00	0.00	0.00	0.000
hypertension: yes	1.00	0.00	0.00	1.000	0.00	1.000	0.00	0.000	1.000	1.00	1.00	1.00	1.000
diabetes_mellitus: yes	1.00	0.00	1.00	0.000	0.00	1.000	0.00	1.000	1.000	1.00	1.00	1.00	1.000
coronary_artery_disease: yes	0.00	0.00	0.00	0.000	0.00	0.000	0.00	0.000	0.000	0.00	0.00	0.00	1.000
appetite: poor	0.00	0.00	1.00	1.000	0.00	0.000	0.00	0.000	0.000	1.00	0.00	1.00	1.000
pedal_edema: yes	0.00	0.00	0.00	1.000	0.00	1.000	0.00	1.000	0.000	0.00	0.00	1.00	1.000
anemia: yes	0.00	0.00	1.00	1.000	0.00	0.000	0.00	0.000	1.000	1.00	1.00	0.00	0.000

In [21]:
```
# define imputer
from sklearn.impute import KNNImputer

imputer = KNNImputer(n_neighbors=5, weights='uniform', metric='nan_euclidean')

impute_columns=list(set(onehotdata.columns)-set(["classification"]))
print(impute_columns)
```

```
['blood_urea', 'serum_creatinine', 'appetite: poor', 'anemia: yes', 'potassium', 'sugar', 'bacteria: presen
t', 'albumin', 'hemoglobin', 'age', 'hypertension: yes', 'blood_pressure', 'red_blood_cell_count', 'specific_
gravity', 'pedal_edema: yes', 'pus_cell: normal', 'pus_cell_clumps: present', 'packed_cell_volume', 'coronary
_artery_disease: yes', 'sodium', 'white_blood_cell_count', 'red_blood_cells: normal', 'diabetes_mellitus: ye
s', 'blood_glucose_random']
```

In [22]:
```
imputer.fit(onehotdata[impute_columns])
```

Out[22]:
```
KNNImputer()
```

In [23]:
```
X_trans=pd.DataFrame(imputer.transform(onehotdata[impute_columns]), columns=impute_columns)
```

In [24]:
```
X_trans.head(13).T
```
Out[24]:

	0	1	2	3	4	5	6	7	8	9	10	11	1
blood_urea	36.00	18.00	53.00	56.000	26.00	25.000	54.00	31.000	60.000	107.00	55.00	60.00	72.00
serum_creatinine	1.20	0.80	1.80	3.800	1.40	1.100	24.00	1.100	1.900	7.20	4.00	2.70	2.10
appetite: poor	0.00	0.00	1.00	1.000	0.00	0.000	0.00	0.000	0.000	1.00	0.00	1.00	1.00
anemia: yes	0.00	0.00	1.00	1.000	0.00	0.000	0.00	0.000	1.000	1.00	1.00	0.00	0.00
potassium	4.20	3.92	4.20	2.500	3.98	3.200	4.00	4.200	4.960	3.70	4.56	4.20	5.80
sugar	0.00	0.00	3.00	0.000	0.00	0.000	0.00	4.000	0.000	0.00	4.00	0.00	1.00
bacteria: present	0.00	0.00	0.00	0.000	0.00	0.000	0.00	0.000	0.000	0.00	0.00	0.00	0.00
albumin	1.00	4.00	2.00	4.000	2.00	3.000	0.00	2.000	3.000	2.00	2.00	3.00	3.00
hemoglobin	15.40	11.30	9.60	11.200	11.60	12.200	12.40	12.400	10.800	9.50	9.40	10.80	9.70
age	48.00	7.00	62.00	48.000	51.00	60.000	68.00	24.000	52.000	53.00	50.00	63.00	68.00
hypertension: yes	1.00	0.00	0.00	1.000	0.00	1.000	0.00	0.000	1.000	1.00	1.00	1.00	1.00
blood_pressure	80.00	50.00	80.00	70.000	80.00	90.000	70.00	74.000	100.000	90.00	60.00	70.00	70.00
red_blood_cell_count	5.20	4.96	3.80	3.900	4.60	4.400	4.64	5.000	4.000	3.70	4.92	3.80	3.40
specific_gravity	1.02	1.02	1.01	1.005	1.01	1.015	1.01	1.015	1.015	1.02	1.01	1.01	1.01
pedal_edema: yes	0.00	0.00	0.00	1.000	0.00	1.000	0.00	1.000	0.000	0.00	0.00	1.00	1.00
pus_cell: normal	1.00	1.00	1.00	0.000	1.00	0.000	1.00	0.000	0.000	0.00	0.00	0.00	1.00
pus_cell_clumps: present	0.00	0.00	0.00	1.000	0.00	0.000	0.00	0.000	1.000	1.00	1.00	1.00	1.00
packed_cell_volume	44.00	38.00	31.00	32.000	35.00	39.000	36.00	44.000	33.000	29.00	28.00	32.00	28.00
coronary_artery_disease: yes	0.00	0.00	0.00	0.000	0.00	0.000	0.00	0.000	0.000	0.00	0.00	0.00	1.00
sodium	137.60	136.80	133.80	111.000	138.40	142.000	104.00	133.800	134.000	114.00	139.00	131.00	138.00
white_blood_cell_count	7800.00	6000.00	7500.00	6700.000	7300.00	7800.000	10280.00	6900.000	9600.000	12100.00	9260.00	4500.00	12200.00
red_blood_cells: normal	0.00	0.00	1.00	1.000	1.00	0.000	0.00	1.000	1.000	0.00	0.00	0.00	0.00
diabetes_mellitus: yes	1.00	0.00	1.00	0.000	0.00	1.000	0.00	1.000	1.000	1.00	1.00	1.00	1.00
blood_glucose_random	121.00	113.00	423.00	117.000	106.00	74.000	100.00	410.000	138.000	70.00	490.00	380.00	208.00

In [25]:
```
X_trans # final datset
```
Out[25]:

	blood_urea	serum_creatinine	appetite: poor	anemia: yes	potassium	sugar	bacteria: present	albumin	hemoglobin	age	...	pedal_edema: yes	pus_cell: normal	pus_cell_
0	36.0	1.2	0.0	0.0	4.20	0.0	0.0	1.0	15.4	48.0	...	0.0	1.0	
1	18.0	0.8	0.0	0.0	3.92	0.0	0.0	4.0	11.3	7.0	...	0.0	1.0	
2	53.0	1.8	1.0	1.0	4.20	3.0	0.0	2.0	9.6	62.0	...	0.0	1.0	
3	56.0	3.8	1.0	1.0	2.50	0.0	0.0	4.0	11.2	48.0	...	1.0	0.0	
4	26.0	1.4	0.0	0.0	3.98	0.0	0.0	2.0	11.6	51.0	...	0.0	1.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
395	49.0	0.5	0.0	0.0	4.90	0.0	0.0	0.0	15.7	55.0	...	0.0	1.0	
396	31.0	1.2	0.0	0.0	3.50	0.0	0.0	0.0	16.5	42.0	...	0.0	1.0	
397	26.0	0.6	0.0	0.0	4.40	0.0	0.0	0.0	15.8	12.0	...	0.0	1.0	
398	50.0	1.0	0.0	0.0	4.90	0.0	0.0	0.0	14.2	17.0	...	0.0	1.0	
399	18.0	1.1	0.0	0.0	3.50	0.0	0.0	0.0	15.8	58.0	...	0.0	1.0	

400 rows × 24 columns

Modelling

In [26]:
```
X=X_trans
y=ckd_df["classification"]

X_prod=X_trans
print(X.shape)
print(y.shape)
print(X_prod.shape)
```

```
(400, 24)
(400,)
(400, 24)
```

Predictive Models with hyperparameter tuning Section

In [27]:
```
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

from sklearn.model_selection import GridSearchCV
```

In [42]:
```
def display_confusion_matrix(y_test,y_pred):

    cm = confusion_matrix(y_test, y_pred)
    group_names = ["True Neg","False Pos","False Neg","True Pos"]
    group_counts = ["{0:0.0f}".format(value) for value in cm.flatten()]
    group_percentages = ["{0:.2%}".format(value) for value in cm.flatten()/np.sum(cm)]

    labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in zip(group_names,group_counts,group_percentages)]
    labels = np.asarray(labels).reshape(2,2)

    sns.heatmap(cm, annot=labels, fmt="", cmap="Blues")
    print(classification_report(y_test, y_pred))
```

In [35]:
```
def plot_roc_curve(fpr, tpr):
    plt.plot(fpr, tpr, label='ROC')
    plt.plot([0, 1], linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend()
    plt.show()
```

In [36]:
```
##Split train and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 4658)

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(320, 24)
(80, 24)
(320,)
(80,)
```

StandardScaler data with PCA implementation

In [37]:
```
# performing preprocessing part
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

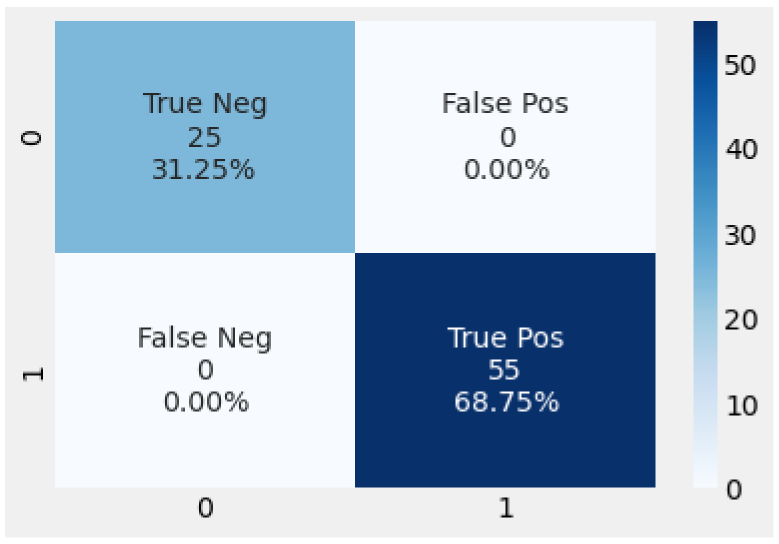
In [38]:
```
# Applying PCA function on training and testing set of X component
from sklearn.decomposition import PCA

pca = PCA(n_components = 5)

X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

explained_variance = pca.explained_variance_ratio_
explained_variance
```

Out[38]:
```
array([0.30943744, 0.07796278, 0.06483376, 0.05539189, 0.0515917 ])
```

k Nearest Neighbors

In [43]:
```
from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n_jobs=-1)
params = {'n_neighbors':[5,6,7,8,9,10],
          'leaf_size':[1,2,3,5],
          'weights':['uniform', 'distance'],
          'algorithm':['auto', 'ball_tree','kd_tree','brute'],
          'n_jobs':[-1]}

start_knn = time.time()
model1 = GridSearchCV(model, param_grid=params, n_jobs=1)
model1.fit(X_train, y_train)
end_knn = time.time()
final_knn = end_knn - start_knn
final_knn = round(final_knn,3)
final_knn

# Print the tuned parameters and score
print("Tuned Logistic Regression Parameters: {}".format(model1.best_params_))
print("Best score is {}".format(model1.best_score_))
print("Best estimator is {} \n\n".format(model1.best_estimator_))

y_pred = model1.predict(X_test)
display_confusion_matrix(y_test, y_pred)
accuracy_knn=accuracy_score(y_test, y_pred)
print("\nAccuracy of k Nearest Neighbors is \t:", accuracy_knn)
print("Best Accuracy of k Nearest Neighbors is : {}".format(model1.best_score_))
print("Computation time {} - Sec".format(final_knn))
```

```
Tuned Logistic Regression Parameters: {'algorithm': 'auto', 'leaf_size': 1, 'n_jobs': -1, 'n_neighbors': 6,
'weights': 'distance'}
Best score is 0.984375
Best estimator is KNeighborsClassifier(leaf_size=1, n_jobs=-1, n_neighbors=6, weights='distance')


              precision    recall  f1-score   support

           0       1.00      1.00      1.00        25
           1       1.00      1.00      1.00        55

    accuracy                           1.00        80
   macro avg       1.00      1.00      1.00        80
weighted avg       1.00      1.00      1.00        80


Accuracy of k Nearest Neighbors is      : 1.0
Best Accuracy of k Nearest Neighbors is : 0.984375
Computation time 12.225 - Sec
```



In [46]:
```
model.get_params().keys()
```

Out[46]:
```
dict_keys(['algorithm', 'leaf_size', 'metric', 'metric_params', 'n_jobs', 'n_neighbors', 'p', 'weights'])
```

Advantages of KNN

1. No Training Period: KNN is called Lazy Learner (Instance based learning). It does not learn anything in the training period. It does not derive any discriminative function from the training data. In other words, there is no training period for it. It stores the training dataset and learns from it only at the time of making real time predictions. This makes the KNN algorithm much faster than other algorithms that require training e.g. SVM, Linear Regression etc.
2. Since the KNN algorithm requires no training before making predictions, new data can be added seamlessly which will not impact the accuracy of the algorithm.
3. KNN is very easy to implement. There are only two parameters required to implement KNN
   i.e. the value of K and the distance function (e.g. Euclidean or Manhattan etc.)

Disadvantages of KNN

1. Does not work well with large dataset: In large datasets, the cost of calculating the distance between the new point and each existing points is huge which degrades the performance of the algorithm.
2. Does not work well with high dimensions: The KNN algorithm doesn't work well with high dimensional data because with large number of dimensions, it becomes difficult for the algorithm to calculate the distance in each dimension.
3. Need feature scaling: We need to do feature scaling (standardization and normalization) before applying KNN algorithm to any dataset. If we don't do so, KNN may generate wrong predictions.
4. Sensitive to noisy data, missing values and outliers: KNN is sensitive to noise in the dataset. We need to manually impute missing values and remove outliers.

In [ ]:

```

```