MANOJ KUMAR - 2048015

Lab 5 & 6 - Chronic Kidney Disease

Create a program to implement ANN, SVM and Logistic regression for binary classification using respective datasets related to your own doamin. Find out the inference related to following:

1. Time complexity
2. Generalizing capacity of each technique
3. Hyper parameter tuning and
4. Advantages and disadvantages of each technique

NOTE: Prepare a detailed report (Word document) on comparative study.

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', 'd
m', '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', 'sodi
um',
       'potassium', 'hemoglobin', 'packed_cell_volume',
       'white_blood_cell_count', 'red_blood_cell_count', 'hypertensio
n',
       '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	
id	0	1	2	3	4	5	
age	48	7	62	48	51	60	
blood_pressure	80	50	80	70	80	90	
specific_gravity	1.02	1.02	1.01	1.005	1.01	1.015	
albumin	1	4	2	4	2	3	
sugar	0	0	3	0	0	0	
red_blood_cells	NaN	NaN	normal	normal	normal	NaN	
pus_cell	normal	normal	normal	abnormal	normal	NaN	ı
pus_cell_clumps	notpresent	notpresent	notpresent	present	notpresent	notpresent	notp
bacteria	notpresent	notpresent	notpresent	notpresent	notpresent	notpresent	notp
blood_glucose_random	121	NaN	423	117	106	74	
blood_urea	36	18	53	56	26	25	
serum_creatinine	1.2	0.8	1.8	3.8	1.4	1.1	
sodium	NaN	NaN	NaN	111	NaN	142	
potassium	NaN	NaN	NaN	2.5	NaN	3.2	
hemoglobin	15.4	11.3	9.6	11.2	11.6	12.2	
packed_cell_volume	44	38	31	32	35	39	
white_blood_cell_count	7800	6000	7500	6700	7300	7800	
red_blood_cell_count	5.2	NaN	NaN	3.9	4.6	4.4	
hypertension	yes	no	no	yes	no	yes	
diabetes_mellitus	yes	no	yes	no	no	yes	
coronary_artery_disease	no	no	no	no	no	no	
appetite	good	good	poor	poor	good	good	
pedal_edema	no	no	no	yes	no	yes	
anemia	no	no	yes	yes	no	no	
classification	ckd	ckd	ckd	ckd	ckd	ckd	

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":
```

In [7]:

```
#Replace incorrect values
ckd_df['diabetes_mellitus'] =ckd_df['diabetes_mellitus'].replace(to_replace={'\tno':
ckd_df['coronary_artery_disease'] = ckd_df['coronary_artery_disease'].replace(to_rep
ckd_df['white_blood_cell_count'] = ckd_df['white_blood_cell_count'].replace(to_repla
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.  9
8.
 157.  76.  99. 114. 263. 173.  95. 108. 156. 264. 123.  93. 107. 15
9.
 140. 171. 270.  92. 137. 204.  79. 207. 124. 144.  91. 162. 246. 25
3.
 141. 182.  86. 150. 146. 425. 112. 250. 360. 163. 129. 133. 102. 15
```

```
8.
 165. 132. 104. 127. 415. 169. 251. 109. 280. 210. 219. 295.  94. 17
2.
 101. 298. 153.  88. 226. 143. 115.  89. 297. 233. 294. 323. 125.  9
0.
 308. 118. 224. 128. 122. 214. 213. 268. 256.  84. 105. 288. 139.  7
8.
 273. 242. 424. 303. 148. 160. 192. 307. 220. 447. 309.  22. 111. 26
1.
 215. 234. 131. 352.  80. 239. 110. 130. 184. 252. 113. 230. 341. 25
5.
 103. 238. 248. 120. 241. 269. 201. 203. 463. 176.  82. 119.  97.  9
6.
  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.   7
5.
  45.   28.  155.   33.   39.  153.   29.   65.  103.   70.   80.   2
0.
 202.   77.   89.   24.   17.   32.  114.   66.   38.  164.  142.   9
6.
 391.   15.  111.   73.   19.   92.   35.   16.  139.   48.   85.   9
8.
 186.   37.   47.   52.   82.   51.  106.   22.  217.   88.  118.   5
0.1
  71.   34.   40.   21.  219.   30.  125.  166.   49.  208.  176.   6
8.
 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.  15
0.
  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  1
5.
  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  1
3.3
  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.  12
7.
 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' '3
7' '30'
 '34' '40' '45' '27' '48' '52' '14' '22' '18' '42' '17' '46' '23' '1
9'
 '25' '41' '26' '15' '21' '43' '20' '47' '9' '49' '50' '53' '51' '5
4']
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' '1
0300'
 '9800' '9100' '7900' '6400' '8600' '18900' '21600' '4300' '8500' '11
300'
 '7200' '7700' '14600' '6300' '7100' '11800' '9400' '5500' '5800' '13
200'
 '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]:

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

Out[9]:

	0	1	2	3	4	5	
id	0	1	2	3	4	5	
age	48	7	62	48	51	60	
blood_pressure	80	50	80	70	80	90	
specific_gravity	1.02	1.02	1.01	1.005	1.01	1.015	
albumin	1	4	2	4	2	3	
sugar	0	0	3	0	0	0	
red_blood_cells	NaN	NaN	normal	normal	normal	NaN	
pus_cell	normal	normal	normal	abnormal	normal	NaN	n
pus_cell_clumps	notpresent	notpresent	notpresent	present	notpresent	notpresent	notp
bacteria	notpresent	notpresent	notpresent	notpresent	notpresent	notpresent	notp
blood_glucose_random	121	NaN	423	117	106	74	
blood_urea	36	18	53	56	26	25	
serum_creatinine	1.2	0.8	1.8	3.8	1.4	1.1	
sodium	NaN	NaN	NaN	111	NaN	142	
potassium	NaN	NaN	NaN	2.5	NaN	3.2	
hemoglobin	15.4	11.3	9.6	11.2	11.6	12.2	
packed_cell_volume	44	38	31	32	35	39	
white_blood_cell_count	7800	6000	7500	6700	7300	7800	
red_blood_cell_count	5.2	NaN	NaN	3.9	4.6	4.4	
hypertension	yes	no	no	yes	no	yes	
diabetes_mellitus	yes	no	yes	no	no	yes	
coronary_artery_disease	no	no	no	no	no	no	
appetite	good	good	poor	poor	good	good	
pedal_edema	no	no	no	yes	no	yes	
anemia	no	no	yes	yes	no	no	
classification	1	1	1	1	1	1	

In [10]:

```
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' '10
300'
 '9800' '9100' '7900' '6400' '8600' '18900' '21600' '4300' '8500' '113
00'
 '7200' '7700' '14600' '6300' '7100' '11800' '9400' '5500' '5800' '132
00'
 '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' '5
900'
 '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']
```

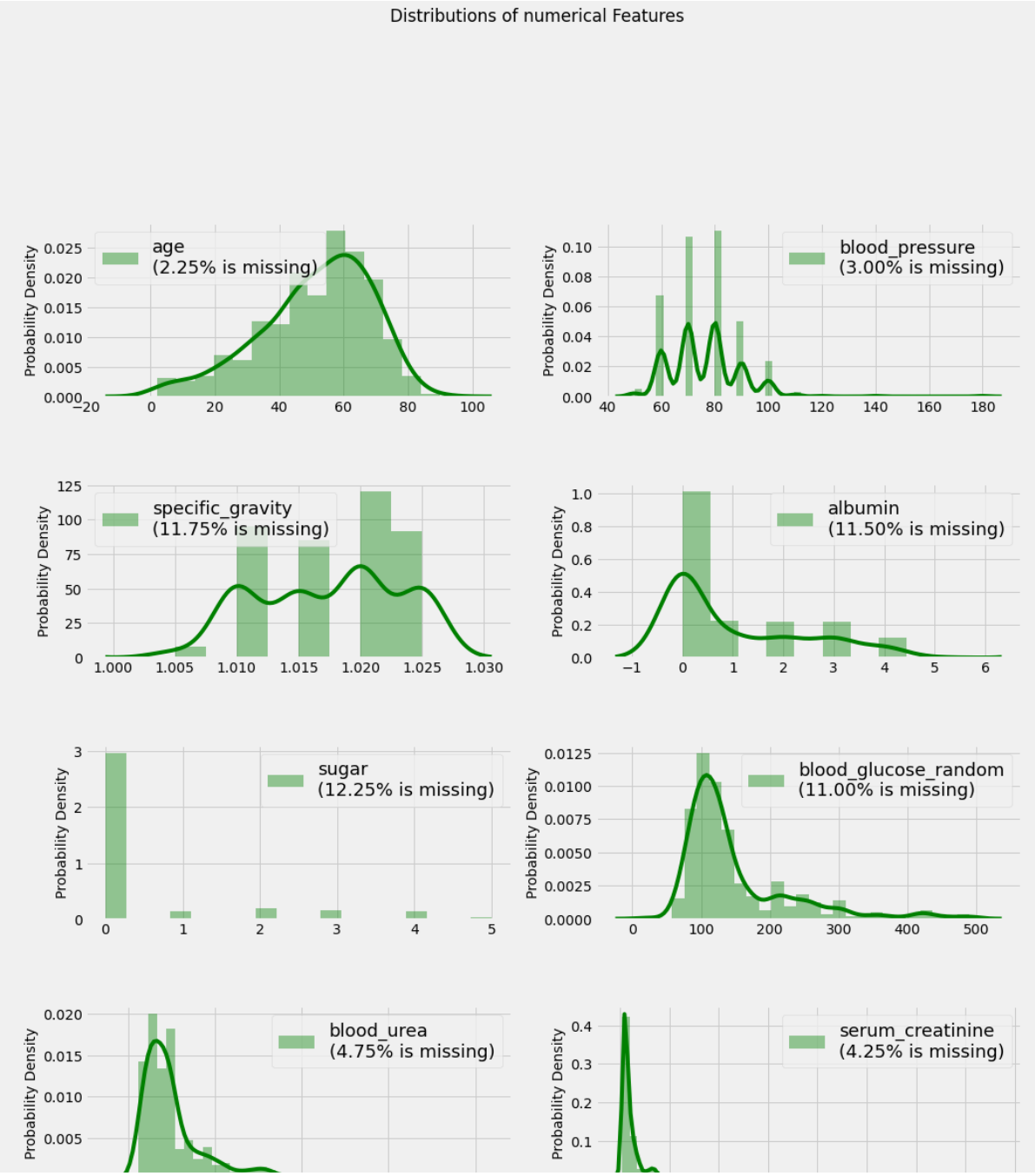
In [13]:

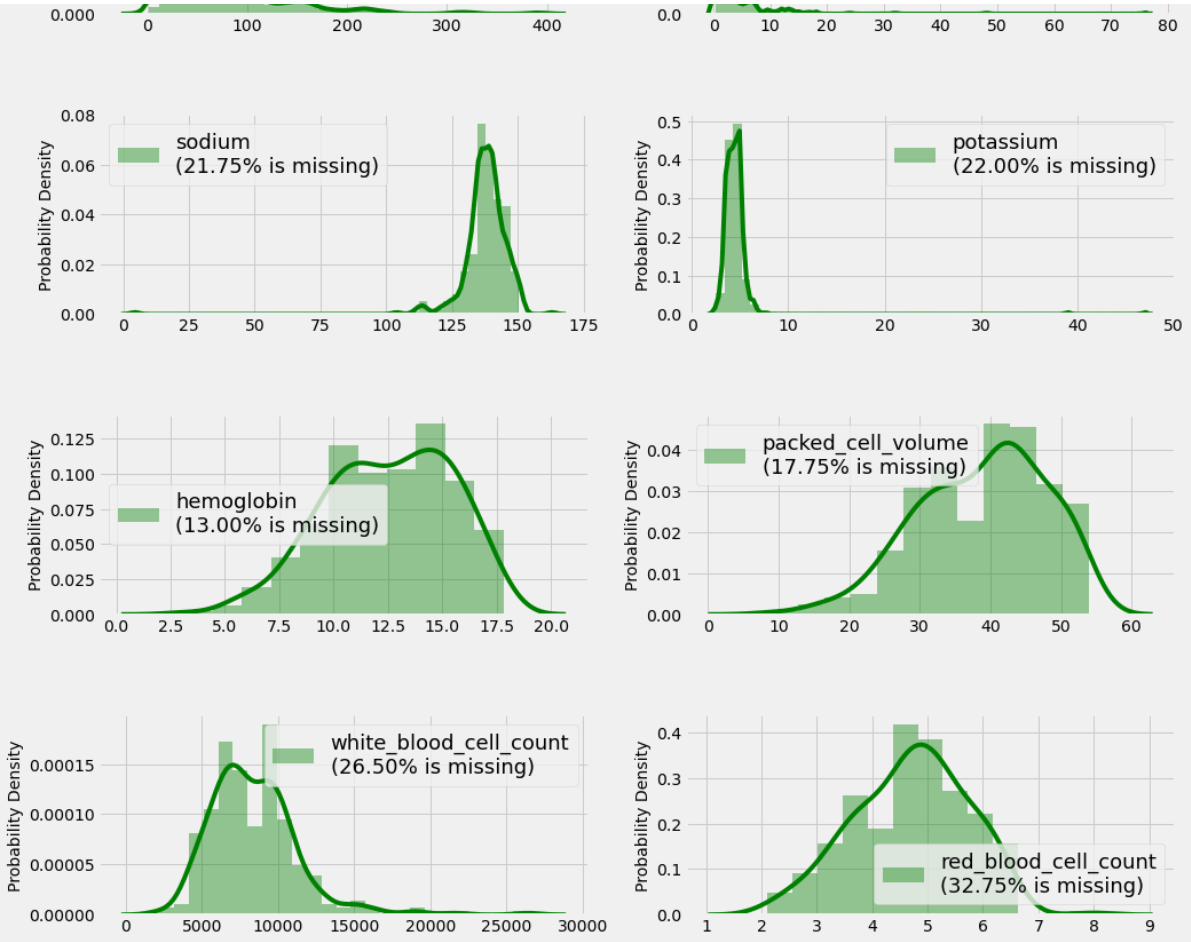
```
import warnings
warnings.simplefilter('ignore')

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

Checking distribution of the numerical features

In [14]:

```
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="green", 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()
```

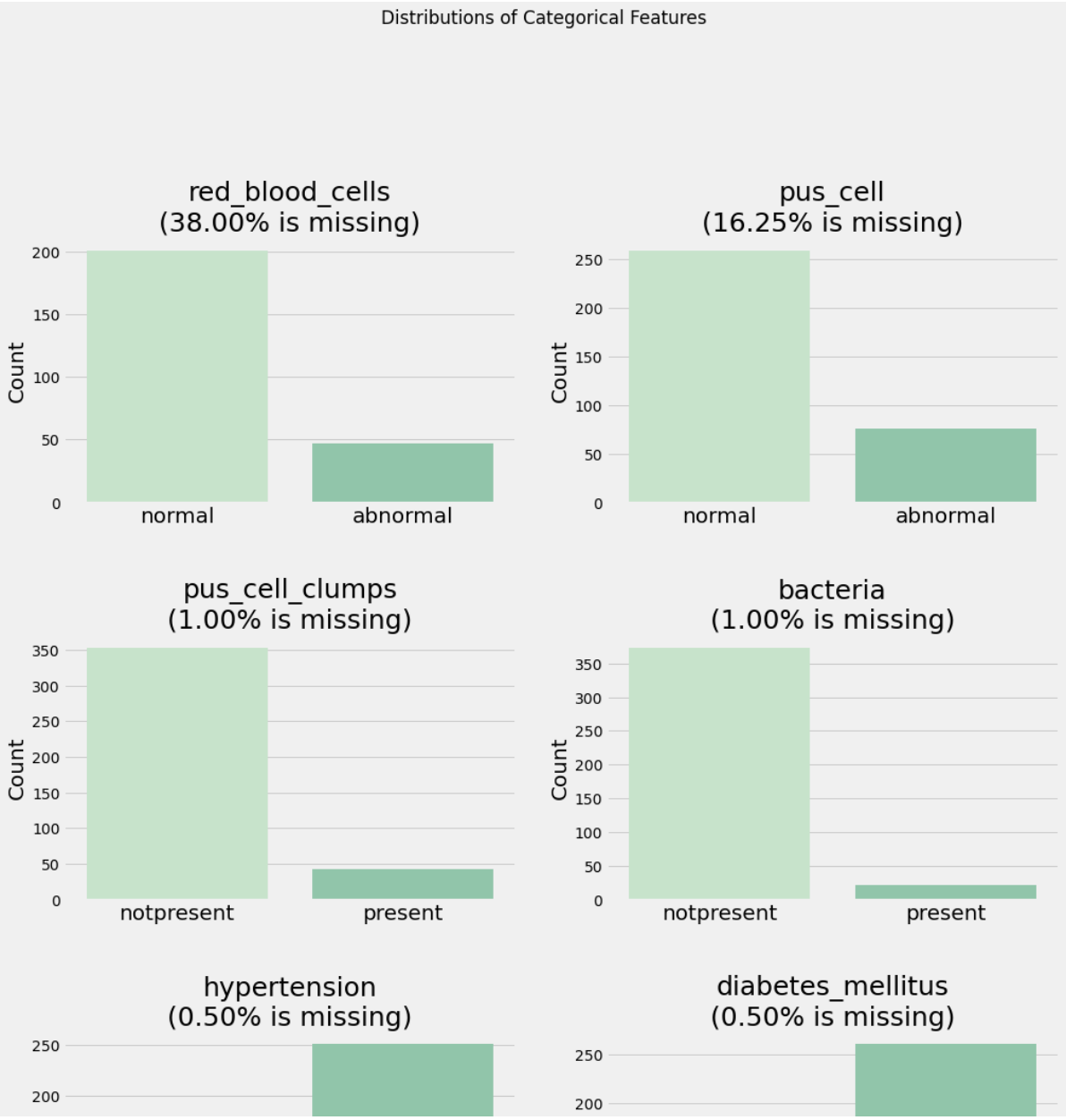


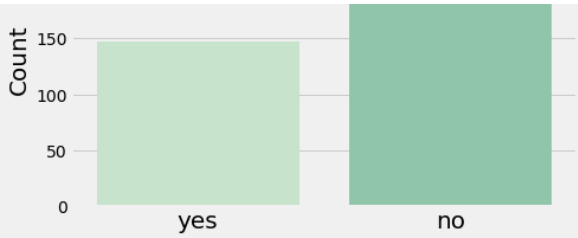


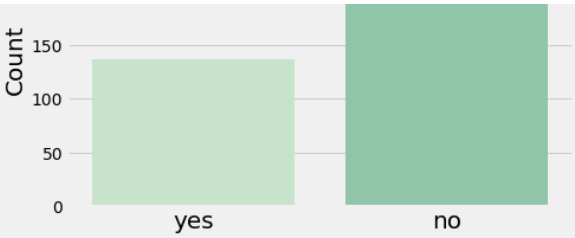
Checking distribution of the Categorical features

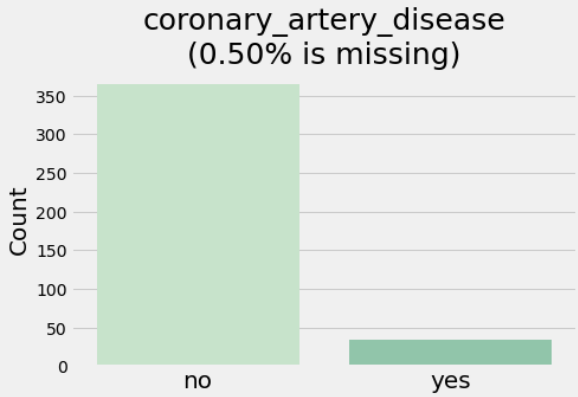
In [15]:

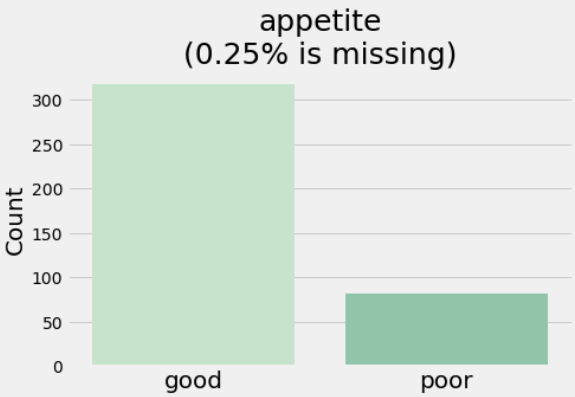
```
style.use('fivethirtyeight')

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=0.85,hue=1), ax=

    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()
```

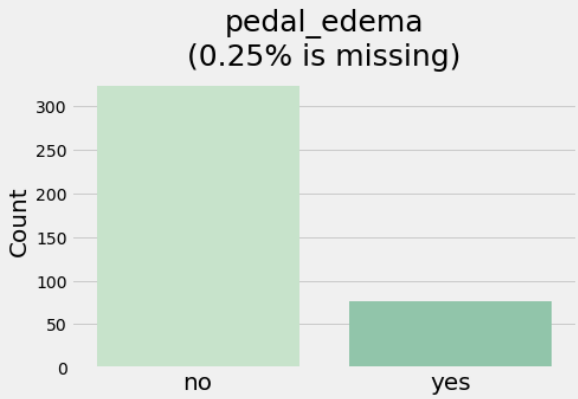


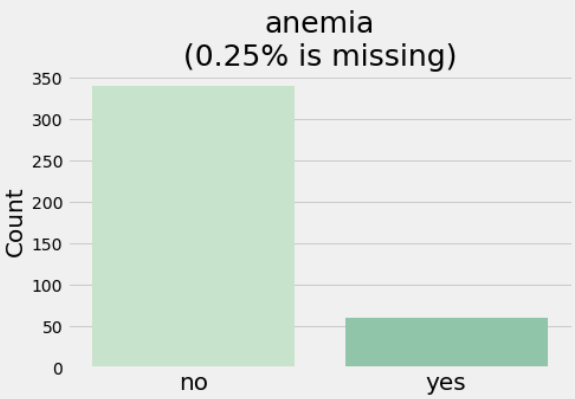

Count
150
100
50
0
yes
no


Count
150
100
50
0
yes
no


coronary_artery_disease
(0.50% is missing)
Count
350
300
250
200
150
100
50
0
no
yes

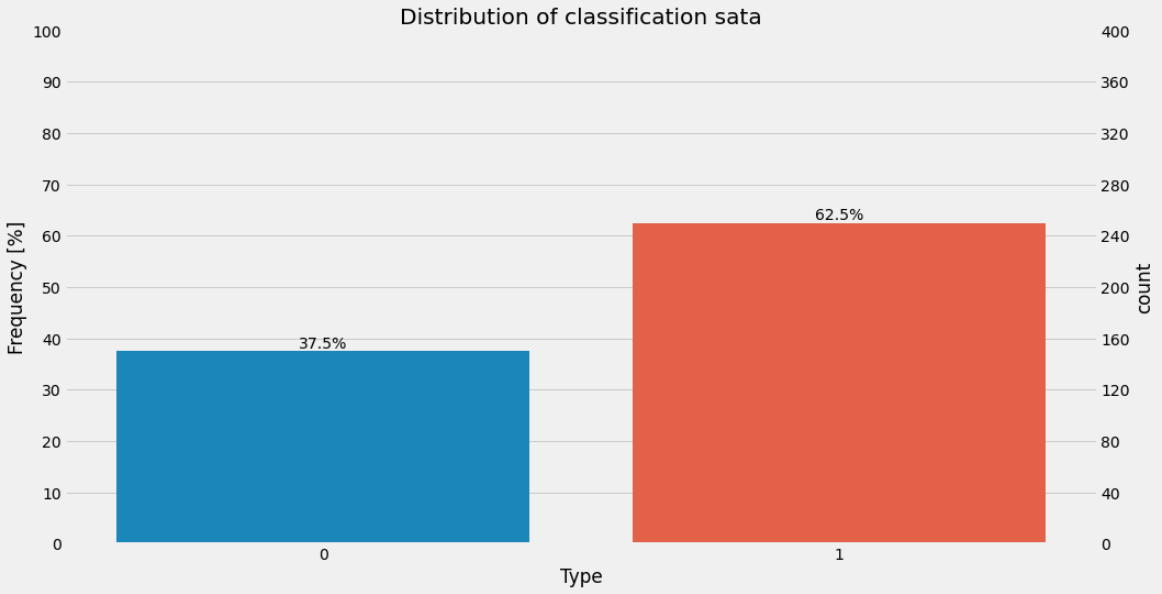

appetite
(0.25% is missing)
Count
300
250
200
150
100
50
0
good
poor


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]:

```
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 bar
ax2.grid(None)
```



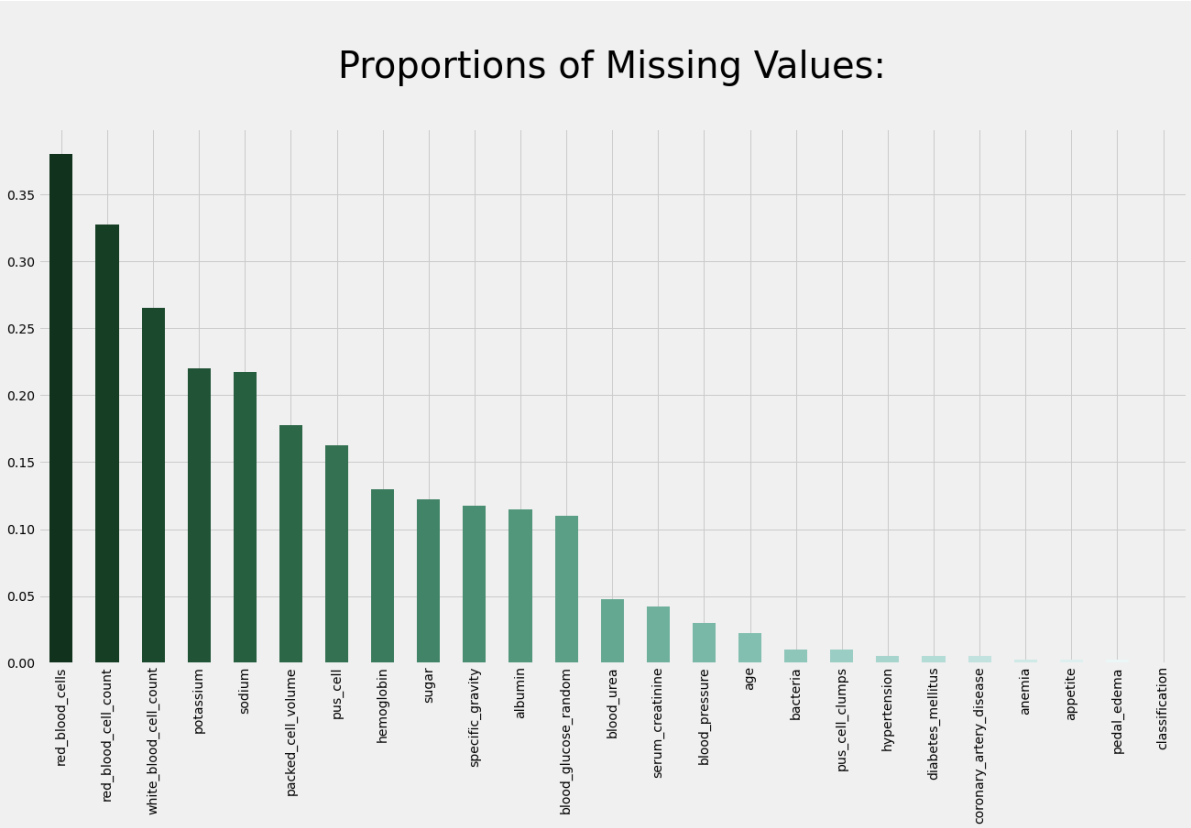
In [17]:

```
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]:

```
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	
age	48	7	62	48	51	60	
blood_pressure	80	50	80	70	80	90	
specific_gravity	1.02	1.02	1.01	1.005	1.01	1.015	
albumin	1	4	2	4	2	3	
sugar	0	0	3	0	0	0	
red_blood_cells	NaN	NaN	normal	normal	normal	NaN	
pus_cell	normal	normal	normal	abnormal	normal	NaN	ı
pus_cell_clumps	notpresent	notpresent	notpresent	present	notpresent	notpresent	notp
bacteria	notpresent	notpresent	notpresent	notpresent	notpresent	notpresent	notp
blood_glucose_random	121	NaN	423	117	106	74	
blood_urea	36	18	53	56	26	25	
serum_creatinine	1.2	0.8	1.8	3.8	1.4	1.1	
sodium	NaN	NaN	NaN	111	NaN	142	
potassium	NaN	NaN	NaN	2.5	NaN	3.2	
hemoglobin	15.4	11.3	9.6	11.2	11.6	12.2	
packed_cell_volume	44	38	31	32	35	39	
white_blood_cell_count	7800	6000	7500	6700	7300	7800	
red_blood_cell_count	5.2	NaN	NaN	3.9	4.6	4.4	
hypertension	yes	no	no	yes	no	yes	
diabetes_mellitus	yes	no	yes	no	no	yes	
coronary_artery_disease	no	no	no	no	no	no	
appetite	good	good	poor	poor	good	good	
pedal_edema	no	no	no	yes	no	yes	
anemia	no	no	yes	yes	no	no	
classification	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	
age	48.00	7.00	62.00	48.000	51.00	60.000	68.00	24.0(
blood_pressure	80.00	50.00	80.00	70.000	80.00	90.000	70.00	Na
specific_gravity	1.02	1.02	1.01	1.005	1.01	1.015	1.01	1.0
albumin	1.00	4.00	2.00	4.000	2.00	3.000	0.00	2.0(
sugar	0.00	0.00	3.00	0.000	0.00	0.000	0.00	4.0(
blood_glucose_random	121.00	NaN	423.00	117.000	106.00	74.000	100.00	410.0(
blood_urea	36.00	18.00	53.00	56.000	26.00	25.000	54.00	31.0(
serum_creatinine	1.20	0.80	1.80	3.800	1.40	1.100	24.00	1.1(
sodium	NaN	NaN	NaN	111.000	NaN	142.000	104.00	Na
potassium	NaN	NaN	NaN	2.500	NaN	3.200	4.00	Na
hemoglobin	15.40	11.30	9.60	11.200	11.60	12.200	12.40	12.4(
packed_cell_volume	44.00	38.00	31.00	32.000	35.00	39.000	36.00	44.0(
white_blood_cell_count	7800.00	6000.00	7500.00	6700.000	7300.00	7800.000	NaN	6900.0(
red_blood_cell_count	5.20	NaN	NaN	3.900	4.60	4.400	NaN	5.0(
classification	1.00	1.00	1.00	1.000	1.00	1.000	1.00	1.0(
red_blood_cells: normal	0.00	0.00	1.00	1.000	1.00	0.000	0.00	1.0(
pus_cell: normal	1.00	1.00	1.00	0.000	1.00	0.000	1.00	0.0(
pus_cell_clumps: present	0.00	0.00	0.00	1.000	0.00	0.000	0.00	0.0(
bacteria: present	0.00	0.00	0.00	0.000	0.00	0.000	0.00	0.0(
hypertension: yes	1.00	0.00	0.00	1.000	0.00	1.000	0.00	0.0(
diabetes_mellitus: yes	1.00	0.00	1.00	0.000	0.00	1.000	0.00	1.0(
coronary_artery_disease: yes	0.00	0.00	0.00	0.000	0.00	0.000	0.00	0.0(
appetite: poor	0.00	0.00	1.00	1.000	0.00	0.000	0.00	0.0(
pedal_edema: yes	0.00	0.00	0.00	1.000	0.00	1.000	0.00	1.0(
anemia: yes	0.00	0.00	1.00	1.000	0.00	0.000	0.00	0.0(

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_glucose_random', 'hemoglobin', 'white_blood_cell_count', 'pus_
cell_clumps: present', 'albumin', 'red_blood_cell_count', 'anemia: ye
s', 'pus_cell: normal', 'pedal_edema: yes', 'appetite: poor', 'bacteri
a: present', 'specific_gravity', 'red_blood_cells: normal', 'packed_ce
ll_volume', 'age', 'sodium', 'coronary_artery_disease: yes', 'blood_pr
essure', 'diabetes_mellitus: yes', 'serum_creatinine', 'sugar', 'potas
sium', 'hypertension: yes', 'blood_urea']
```

In [22]:

```
imputer.fit(onehotdata[impute_columns])
```

Out[22]:

```
KNNImputer()
```

In [23]:

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

In [24]:

```
X_trans.head(13).T
```

Out[24]:

	0	1	2	3	4	5	6	
blood_glucose_random	121.00	113.00	423.00	117.000	106.00	74.000	100.00	410
hemoglobin	15.40	11.30	9.60	11.200	11.60	12.200	12.40	12
white_blood_cell_count	7800.00	6000.00	7500.00	6700.000	7300.00	7800.000	10280.00	6900
pus_cell_clumps: present	0.00	0.00	0.00	1.000	0.00	0.000	0.00	0
albumin	1.00	4.00	2.00	4.000	2.00	3.000	0.00	2
red_blood_cell_count	5.20	4.96	3.80	3.900	4.60	4.400	4.64	5
anemia: yes	0.00	0.00	1.00	1.000	0.00	0.000	0.00	0
pus_cell: normal	1.00	1.00	1.00	0.000	1.00	0.000	1.00	0
pedal_edema: yes	0.00	0.00	0.00	1.000	0.00	1.000	0.00	1
appetite: poor	0.00	0.00	1.00	1.000	0.00	0.000	0.00	0
bacteria: present	0.00	0.00	0.00	0.000	0.00	0.000	0.00	0
specific_gravity	1.02	1.02	1.01	1.005	1.01	1.015	1.01	1
red_blood_cells: normal	0.00	0.00	1.00	1.000	1.00	0.000	0.00	1
packed_cell_volume	44.00	38.00	31.00	32.000	35.00	39.000	36.00	44
age	48.00	7.00	62.00	48.000	51.00	60.000	68.00	24
sodium	137.60	136.80	133.80	111.000	138.40	142.000	104.00	133
coronary_artery_disease: yes	0.00	0.00	0.00	0.000	0.00	0.000	0.00	0
blood_pressure	80.00	50.00	80.00	70.000	80.00	90.000	70.00	74
diabetes_mellitus: yes	1.00	0.00	1.00	0.000	0.00	1.000	0.00	1
serum_creatinine	1.20	0.80	1.80	3.800	1.40	1.100	24.00	1
sugar	0.00	0.00	3.00	0.000	0.00	0.000	0.00	4
potassium	4.20	3.92	4.20	2.500	3.98	3.200	4.00	4
hypertension: yes	1.00	0.00	0.00	1.000	0.00	1.000	0.00	0
blood_urea	36.00	18.00	53.00	56.000	26.00	25.000	54.00	31

In [25]:

```
X_trans # final datset
```

Out[25]:

	blood_glucose_random	hemoglobin	white_blood_cell_count	pus_cell_clumps: present	albumin	red_
0	121.0	15.4	7800.0	0.0	1.0	
1	113.0	11.3	6000.0	0.0	4.0	
2	423.0	9.6	7500.0	0.0	2.0	
3	117.0	11.2	6700.0	1.0	4.0	
4	106.0	11.6	7300.0	0.0	2.0	
...	...	...	...	...	...	
395	140.0	15.7	6700.0	0.0	0.0	
396	75.0	16.5	7800.0	0.0	0.0	
397	100.0	15.8	6600.0	0.0	0.0	
398	114.0	14.2	7200.0	0.0	0.0	
399	131.0	15.8	6800.0	0.0	0.0	

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 [28]:

```
def display_confusion_matrix(y_test,y_pred):

    cm = confusion_matrix(y_test, y_pred_lr)
    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,grc
    labels = np.asarray(labels).reshape(2,2)

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

In [29]:

```
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 [30]:

```
##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_s

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

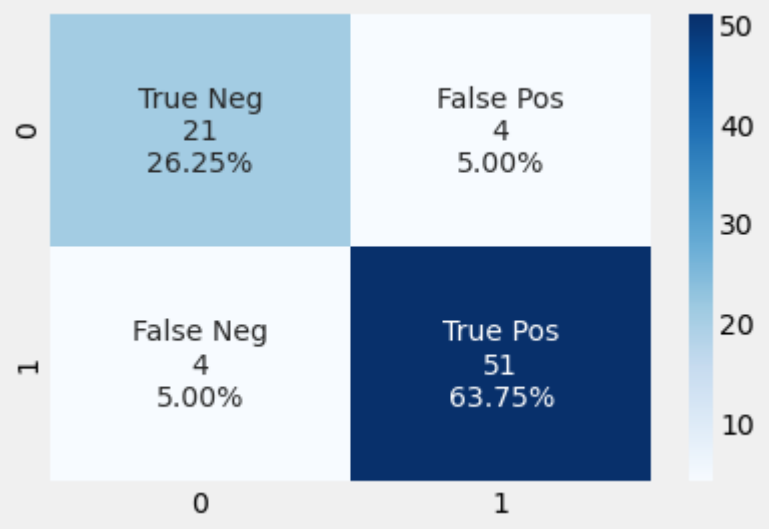
```
(320, 24)
(80, 24)
(320,)
(80,)
```

Logistic Regression Hyper parameter tuning

In [31]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

c_space = np.logspace(-5, 8, 15)
param_grid = {'C': c_space}

start_lr = time.time()
lr = GridSearchCV(LogisticRegression(),
                  param_grid,
                  cv = 5)
lr.fit(X_train, y_train)
end_lr = time.time()
final_lr = end_lr - start_lr
final_lr = round(final_lr,3)
final_lr

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

y_pred_lr = lr.predict(X_test)
display_confusion_matrix(y_test, y_pred_lr)
accuracy_lr=accuracy_score(y_test, y_pred_lr)
print("\nAccuracy of Logistic Regression is :", accuracy_lr)
print("Computation time {} - Sec".format(final_lr))
```

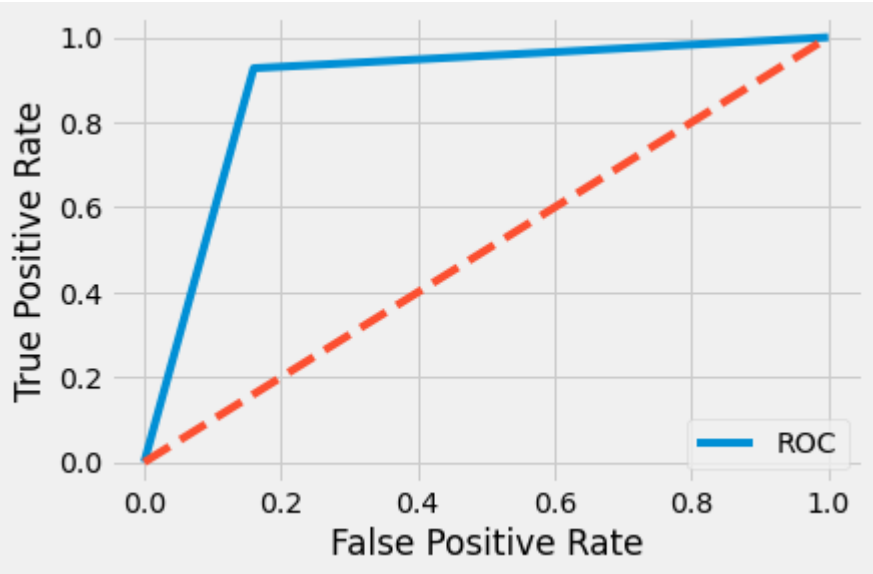
```
Tuned Logistic Regression Parameters: {'C': 0.006105402296585327}
Best score is 0.896875
Best estimator is LogisticRegression(C=0.006105402296585327)


              precision    recall  f1-score   support

           0       0.84      0.84      0.84        25
           1       0.93      0.93      0.93        55

    accuracy                           0.90        80
   macro avg       0.88      0.88      0.88        80
weighted avg       0.90      0.90      0.90        80


Accuracy of Logistic Regression is : 0.9
Computation time 3.894 - Sec
```



In [32]:

```
auc = roc_auc_score(y_test, y_pred_lr)
print('AUC: %.2f' % auc)

fpr, tpr, thresholds = roc_curve(y_test, y_pred_lr)
plot_roc_curve(fpr, tpr)
```

```
AUC: 0.88
```

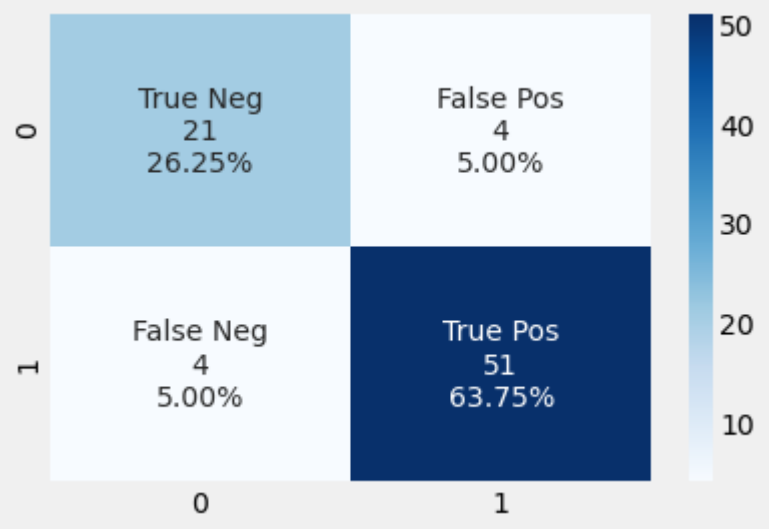


Decision Tree Hyper parameter tuning

In [33]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import RandomizedSearchCV

hyperparam_combs = {
    'max_depth': [4, 6, 8, 10, 12],
    'criterion': ['gini', 'entropy'],
    'min_samples_split': [2, 10, 20, 30, 40],
    'max_features': [0.2, 0.4, 0.6, 0.8, 1],
    'max_leaf_nodes': [8, 16, 32, 64, 128],
    'class_weight': [{0: 1, 1: 1}, {0: 1, 1: 2}, {0: 1, 1: 3}, {0: 1, 1: 4}, {0: 1,
}

start_dt = time.time()
clf = RandomizedSearchCV(DecisionTreeClassifier(),
                         hyperparam_combs,
                         scoring='f1',
                         random_state=1,
                         n_iter=20)

dt_model = clf.fit(X_train, y_train)
end_dt = time.time()
final_dt = end_dt - start_dt
final_dt = round(final_dt,3)
final_dt

# Print the tuned parameters and score
print("Tuned Decision Tree Parameters: {}".format(dt_model.best_params_))
print("Best score is {}".format(dt_model.best_score_))
print("Best estimator is {}".format(dt_model.best_estimator_))

y_pred_dt = dt_model.predict(X_test)
display_confusion_matrix(y_test, y_pred_dt)
accuracy_dt=accuracy_score(y_test, y_pred_dt)
print("Accuracy of Decision Tree is :", accuracy_dt)
print("Computation time {} - Sec".format(final_dt))
```

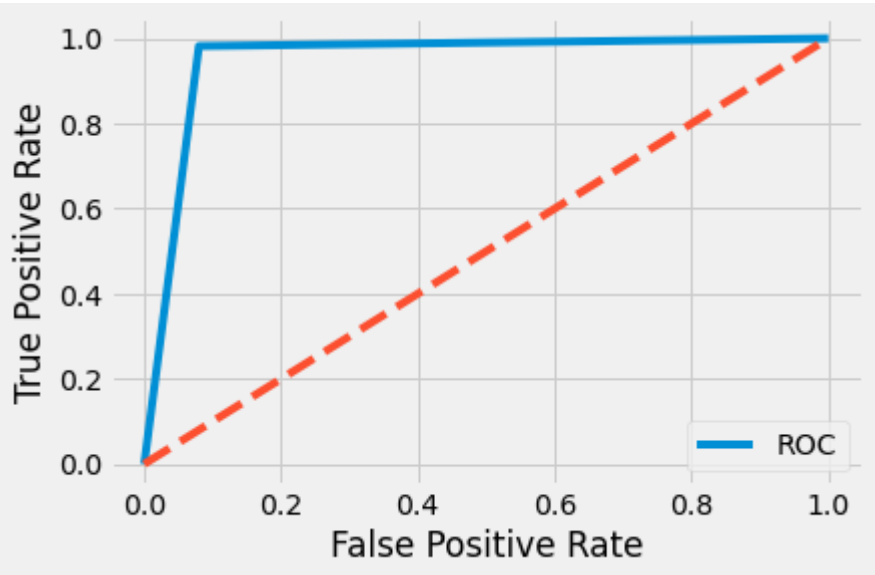
```
Tuned Decision Tree Parameters: {'min_samples_split': 2, 'max_leaf_nod
es': 128, 'max_features': 0.2, 'max_depth': 10, 'criterion': 'gini',
'class_weight': {0: 1, 1: 3}}
Best score is 0.973536420904842
Best estimator is DecisionTreeClassifier(class_weight={0: 1, 1: 3}, ma
x_depth=10,
                       max_features=0.2, max_leaf_nodes=128)
              precision    recall  f1-score   support

           0       0.96      0.92      0.94        25
           1       0.96      0.98      0.97        55

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

Accuracy of Decision Tree is : 0.9625
Computation time 0.605 - Sec
```



In [34]:

```
auc = roc_auc_score(y_test, y_pred_dt)
print('AUC: %.2f' % auc)

fpr, tpr, thresholds = roc_curve(y_test, y_pred_dt)
plot_roc_curve(fpr, tpr)
```

```
AUC: 0.95
```

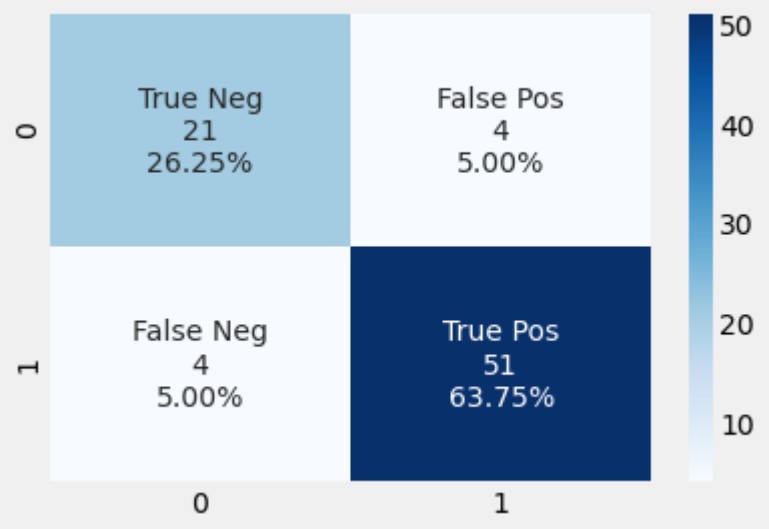


Random Forest Hyper parameter tuning

In [35]:

```
from sklearn.ensemble import RandomForestClassifier
param_grid = {"n_estimators": np.arange(2, 300, 2),
              "max_depth": np.arange(1, 28, 1),
              "min_samples_split": np.arange(1,150,1),
              "min_samples_leaf": np.arange(1,60,1),
              "max_leaf_nodes": np.arange(2,60,1),
              "min_weight_fraction_leaf": np.arange(0.1,0.4, 0.1)}

start_rf = time.time()
rf = RandomizedSearchCV(RandomForestClassifier(),
                        param_grid,
                        scoring='f1',
                        random_state=4658,
                        n_iter=20)

rf_model = rf.fit(X_train, y_train)
end_rf = time.time()
final_rf = end_rf - start_rf
final_rf = round(final_rf,3)
final_rf

# Print the tuned parameters and score
print("Tuned Random Tree Parameters: {}".format(rf_model.best_params_))
print("Best score is {}".format(rf_model.best_score_))
print("Best estimator is {}".format(rf_model.best_estimator_))

y_pred_rf = rf_model.predict(X_test)
display_confusion_matrix(y_test, y_pred_rf)
accuracy_rf=accuracy_score(y_test, y_pred_rf)
print("Accuracy of Random Forests model is :", accuracy_rf)
print("Computation time {} - Sec".format(final_rf))
```

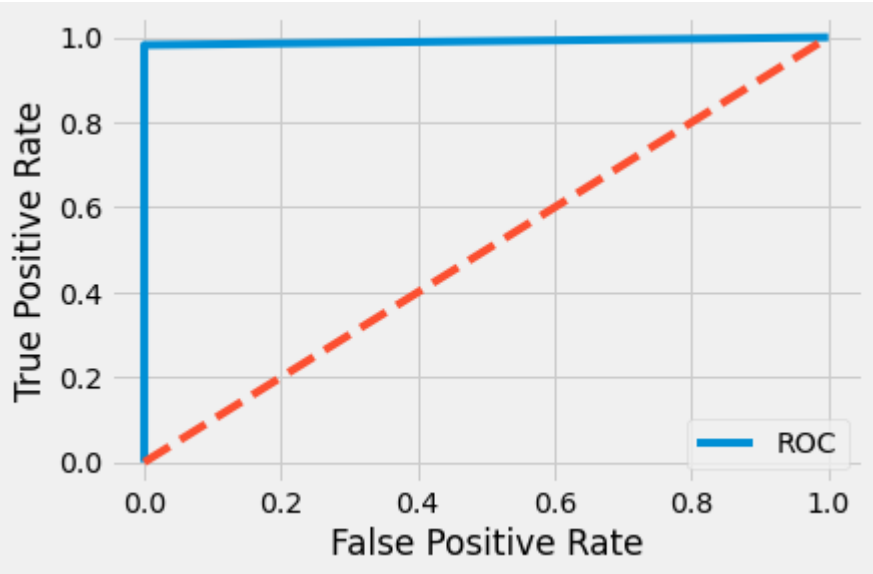
```
Tuned Random Tree Parameters: {'n_estimators': 250, 'min_weight_fracti
on_leaf': 0.1, 'min_samples_split': 34, 'min_samples_leaf': 44, 'max_l
eaf_nodes': 56, 'max_depth': 8}
Best score is 0.9740908214592426
Best estimator is RandomForestClassifier(max_depth=8, max_leaf_nodes=5
6, min_samples_leaf=44,
                       min_samples_split=34, min_weight_fraction_leaf=
0.1,
                       n_estimators=250)
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        25
           1       1.00      0.98      0.99        55

    accuracy                           0.99        80
   macro avg       0.98      0.99      0.99        80
weighted avg       0.99      0.99      0.99        80

Accuracy of Random Forests model is : 0.9875
Computation time 22.908 - Sec
```



In [36]:

```
auc = roc_auc_score(y_test, y_pred_rf)
print('AUC: %.2f' % auc)

fpr, tpr, thresholds = roc_curve(y_test, y_pred_rf)
plot_roc_curve(fpr, tpr)
```

```
AUC: 0.99
```



Support Vector Machine Hyper parameter tuning

In [37]:

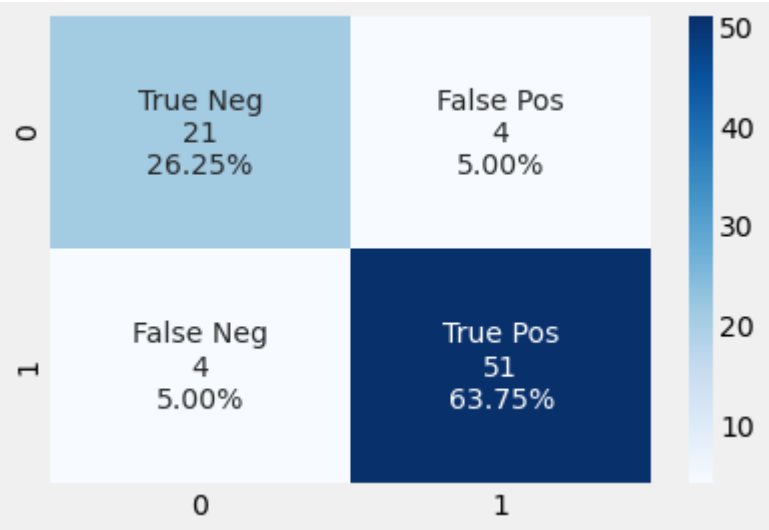
```
from sklearn.svm import SVC

# defining parameter range
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001]}

start_svm = time.time()
svm = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3)
# fitting the model for grid search
svm.fit(X_train, y_train)

end_svm = time.time()
final_svm = end_svm - start_svm
final_svm = round(final_svm,3)
final_svm

# Print the tuned parameters and score
print("Tuned Support Vector Machine Parameters: {}".format(svm.best_params_))
print("Best score is {}".format(svm.best_score_))
print("Best estimator is {}".format(svm.best_estimator_))
```

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits
[CV] C=0.1, gamma=1
.................................................
[CV] ..................... C=0.1, gamma=1, score=0.609, total=   0.0
s
[CV] C=0.1, gamma=1
.................................................
[CV] ..................... C=0.1, gamma=1, score=0.609, total=   0.0
s
[CV] C=0.1, gamma=1
.................................................
[CV] ..................... C=0.1, gamma=1, score=0.609, total=   0.0
s
[CV] C=0.1, gamma=1
.................................................
[CV] ..................... C=0.1, gamma=1, score=0.609, total=   0.0
s
[CV] C=0.1, gamma=1
.................................................
```

In [38]:

```
y_pred_svm = svm.predict(X_test)
display_confusion_matrix(y_test, y_pred_svm)
accuracy_svm=accuracy_score(y_test, y_pred_svm)
print("Accuracy of Support Vector Machine is :", accuracy_svm)
print("Computation time {} - Sec".format(final_svm))
```

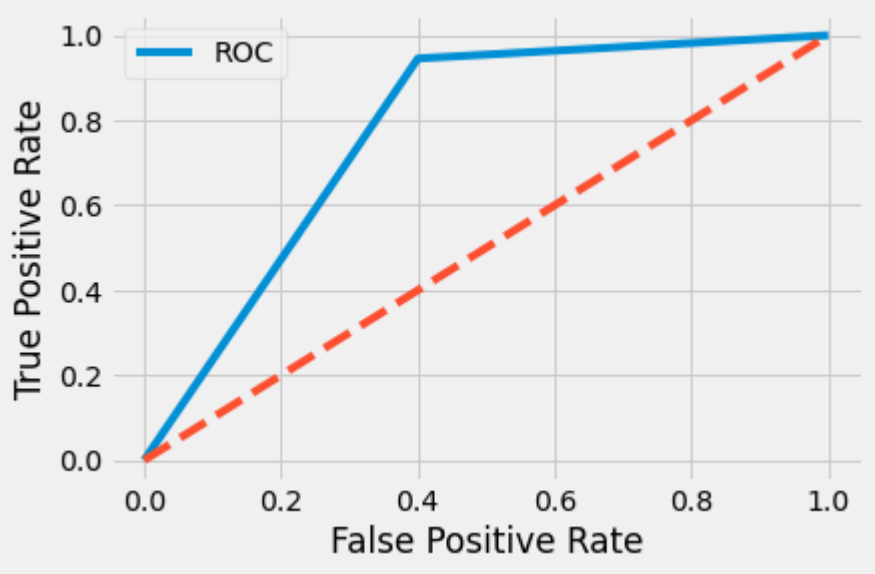
```
              precision    recall  f1-score   support

           0       0.83      0.60      0.70        25
           1       0.84      0.95      0.89        55

    accuracy                           0.84        80
   macro avg       0.84      0.77      0.79        80
weighted avg       0.84      0.84      0.83        80

Accuracy of Support Vector Machine is : 0.8375
Computation time 1.166 - Sec
```



In [39]:

```
auc = roc_auc_score(y_test, y_pred_svm)
print('AUC: %.2f' % auc)

fpr, tpr, thresholds = roc_curve(y_test, y_pred_svm)
plot_roc_curve(fpr, tpr)
```

```
AUC: 0.77
```



Artificial neural network

In [40]:

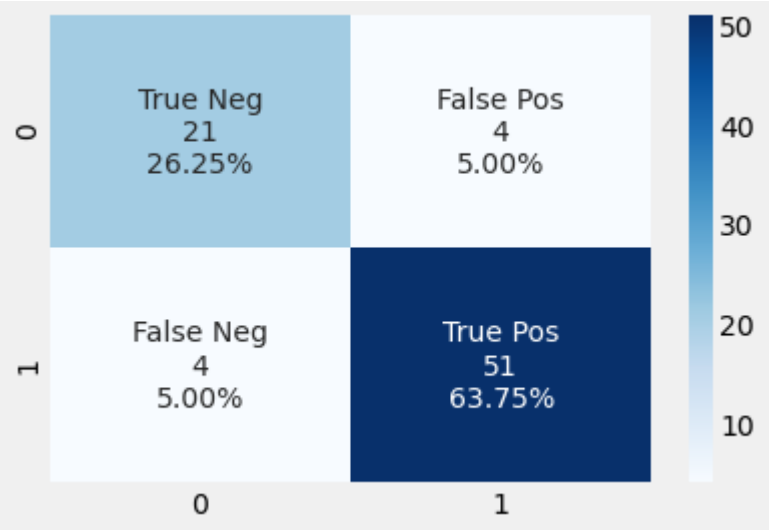
```
from sklearn.neural_network import MLPClassifier

# defining parameter range
param_grid = {
              'max_iter': [1000]
}

start_mlp = time.time()
mlp = GridSearchCV(MLPClassifier(), param_grid, refit = True, verbose = 3)
# fitting the model for grid search
mlp.fit(X_train, y_train.values.ravel())

end_mlp = time.time()
final_mlp = end_mlp - start_mlp
final_mlp = round(final_mlp,3)
final_mlp

# Print the tuned parameters and score
print("Tuned Artificial neural network Parameters: {}".format(mlp.best_params_))
print("Best score is {}".format(mlp.best_score_))
print("Best estimator is {}".format(mlp.best_estimator_))
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV] max_iter=1000 ...................................................
[CV] ....................... max_iter=1000, score=0.750, total=   0.1s
[CV] max_iter=1000 ...................................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurren
t workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remainin
g:    0.0s

[CV] ....................... max_iter=1000, score=0.859, total=   0.2s
[CV] max_iter=1000 ...................................................
[CV] ....................... max_iter=1000, score=0.641, total=   0.2s
[CV] max_iter=1000 ...................................................

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.3s remainin
g:    0.0s

[CV] ....................... max_iter=1000, score=0.703, total=   0.1s
[CV] max_iter=1000 ...................................................
[CV] ....................... max_iter=1000, score=0.672, total=   0.1s

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.7s finished

Tuned Artificial neural network Parameters: {'max_iter': 1000}
Best score is 0.725
Best estimator is MLPClassifier(max_iter=1000)
```

In [41]:

```
y_pred_mlp = mlp.predict(X_test)
display_confusion_matrix(y_test, y_pred_mlp)
accuracy_mlp=accuracy_score(y_test, y_pred_mlp)
print("Accuracy of Artificial neural network  is :", accuracy_mlp)
print("Computation time {} - Sec".format(final_mlp))
```

```
              precision    recall  f1-score   support

           0       0.94      0.68      0.79        25
           1       0.87      0.98      0.92        55

    accuracy                           0.89        80
   macro avg       0.91      0.83      0.86        80
weighted avg       0.89      0.89      0.88        80

Accuracy of Artificial neural network  is : 0.8875
Computation time 0.926 - Sec
```



In [42]:

```
auc = roc_auc_score(y_test, y_pred_mlp)
print('AUC: %.2f' % auc)

fpr, tpr, thresholds = roc_curve(y_test, y_pred_mlp)
plot_roc_curve(fpr, tpr)
```

```
AUC: 0.83
```

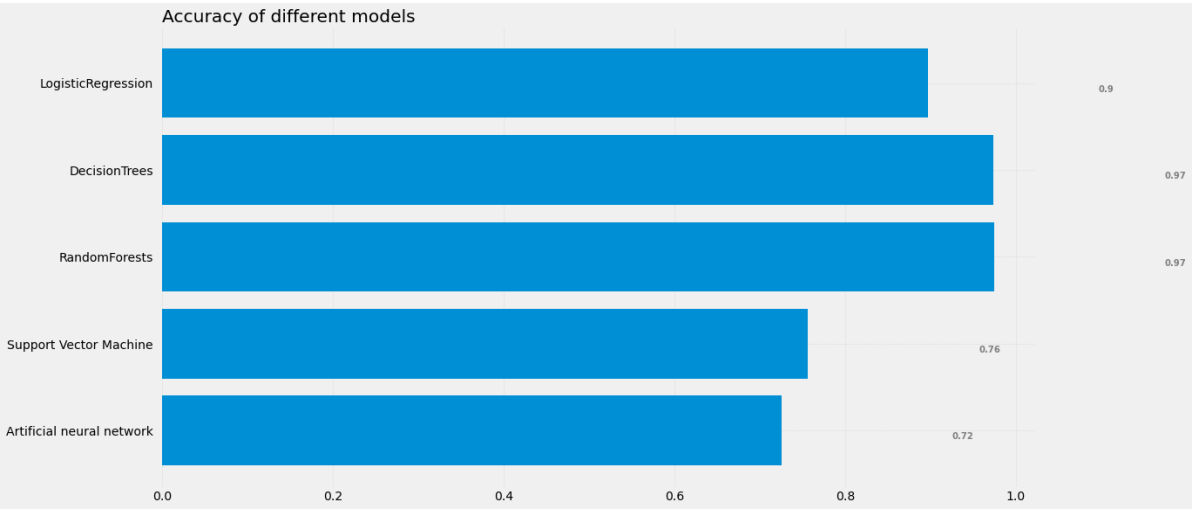
In [43]:

```
accuracies1 = [lr.best_score_,dt_model.best_score_,rf_model.best_score_, svm.best_sc
final_time1 = [final_lr,final_dt,final_rf, final_svm, final_mlp]
print(accuracies1)
print(final_time1)

models= ['LogisticRegression', 'DecisionTrees', 'RandomForests', 'Support Vector Mac
```

```
[0.896875, 0.973536420904842, 0.9740908214592426, 0.75625, 0.725]
[3.894, 0.605, 22.908, 1.166, 0.926]
```

In [44]:

```
# Figure Size
fig, ax = plt.subplots(figsize =(16, 9))
# Horizontal Bar Plot
ax.barh(models, accuracies1)

# Remove axes splines
for s in ['top', 'bottom', 'left', 'right']:
    ax.spines[s].set_visible(False)
# Remove x, y Ticks
ax.xaxis.set_ticks_position('none')
ax.yaxis.set_ticks_position('none')

# Add padding between axes and labels
ax.xaxis.set_tick_params(pad = 5)
ax.yaxis.set_tick_params(pad = 10)
# Add x, y gridlines
ax.grid(b = True, color ='grey',
        linestyle ='-.', linewidth = 0.5,
        alpha = 0.2)

# Show top values
ax.invert_yaxis()

# Add annotation to bars
for i in ax.patches:
    plt.text(i.get_width()+0.2, i.get_y()+0.5,
             str(round((i.get_width()), 2)),
             fontsize = 10, fontweight ='bold',
             color ='grey')
ax.set_title('Accuracy of different models', loc ='left')
plt.show()
```



StandardScaler data with PCA implementation

In [45]:

```
# performing preprocessing part
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

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

In [46]:

```
# Applying PCA function on training and testing set of X component
from sklearn.decomposition import PCA

pca = PCA(n_components = 2)

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

explained_variance = pca.explained_variance_ratio_
explained_variance
```

Out[46]:

```
array([0.30943656, 0.07796664])
```

Logistic Regression Hyper parameter tuning

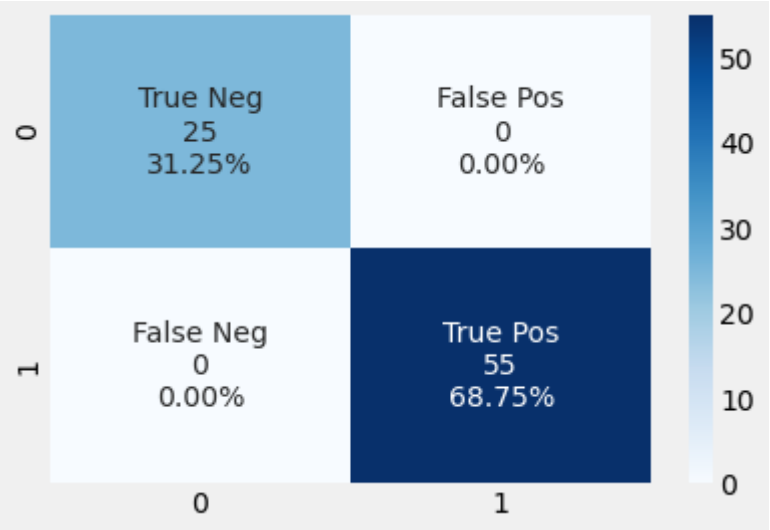
In [47]:

```
c_space = np.logspace(-5, 8, 15)
param_grid = {'C': c_space}

start_lr = time.time()
lr = GridSearchCV(LogisticRegression(),
                  param_grid,
                  cv = 5)
lr.fit(X_train, y_train)
end_lr = time.time()
final_lr = end_lr - start_lr
final_lr = round(final_lr,3)
final_lr

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

y_pred_lr = lr.predict(X_test)
display_confusion_matrix(y_test, y_pred_lr)
accuracy_lr=lr.best_score_
print("\nAccuracy of Logistic Regression is :", accuracy_lr)
print("Computation time {} - Sec".format(final_lr))
```

```
Tuned Logistic Regression Parameters: {'C': 3.727593720314938}
Best score is 0.978125
Best estimator is LogisticRegression(C=3.727593720314938)
```

	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

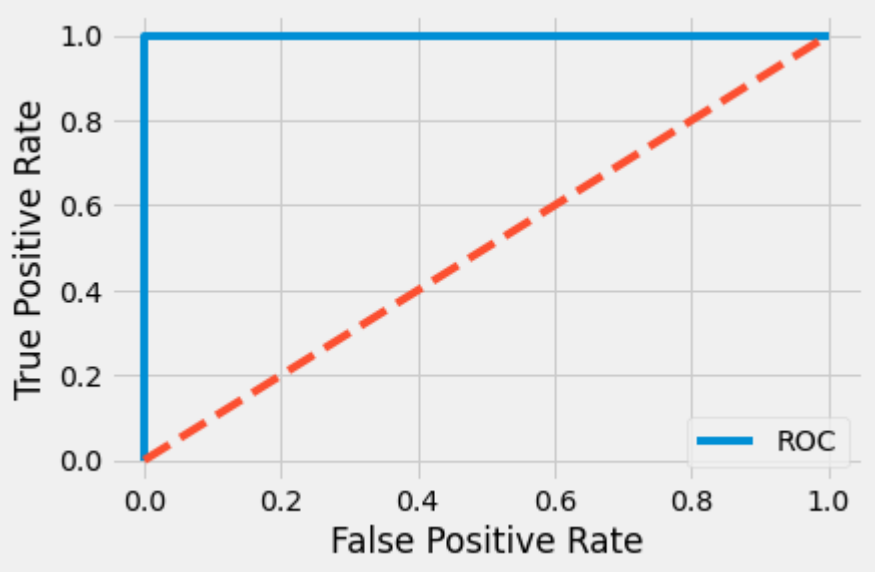
```
Accuracy of Logistic Regression is : 0.978125
Computation time 0.509 - Sec
```



In [48]:

```
auc = roc_auc_score(y_test, y_pred_lr)
print('AUC: %.2f' % auc)

fpr, tpr, thresholds = roc_curve(y_test, y_pred_lr)
plot_roc_curve(fpr, tpr)
```

```
AUC: 1.00
```

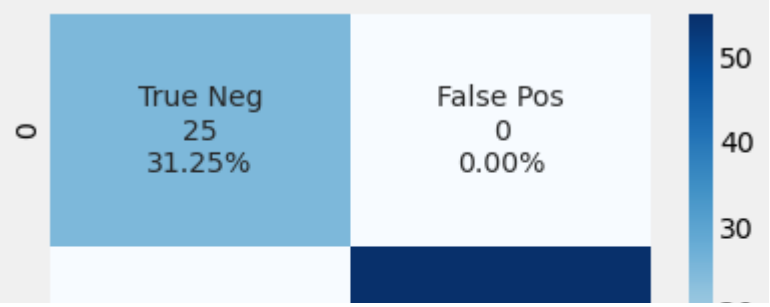


Decision Tree Hyper parameter tuning

In [49]:

```
hyperparam_combs = {
    'max_depth': [4, 6, 8, 10, 12],
    'criterion': ['gini', 'entropy'],
    'min_samples_split': [2, 10, 20, 30, 40],
    'max_features': [0.2, 0.4, 0.6, 0.8, 1],
    'max_leaf_nodes': [8, 16, 32, 64, 128],
    'class_weight': [{0: 1, 1: 1}, {0: 1, 1: 2}, {0: 1, 1: 3}, {0: 1, 1: 4}, {0: 1,
}

start_dt = time.time()
clf = RandomizedSearchCV(DecisionTreeClassifier(),
                         hyperparam_combs,
                         scoring='f1',
                         random_state=1,
                         n_iter=20)

dt_model = clf.fit(X_train, y_train)
end_dt = time.time()
final_dt = end_dt - start_dt
final_dt = round(final_dt,3)
final_dt

# Print the tuned parameters and score
print("Tuned Decision Tree Parameters: {}".format(dt_model.best_params_))
print("Best score is {}".format(dt_model.best_score_))
print("Best estimator is {}".format(dt_model.best_estimator_))

y_pred_dt = dt_model.predict(X_test)
display_confusion_matrix(y_test, y_pred_dt)
accuracy_dt=dt_model.best_score_
print("Accuracy of Decision Tree is :", accuracy_dt)
print("Computation time {} - Sec".format(final_dt))
```

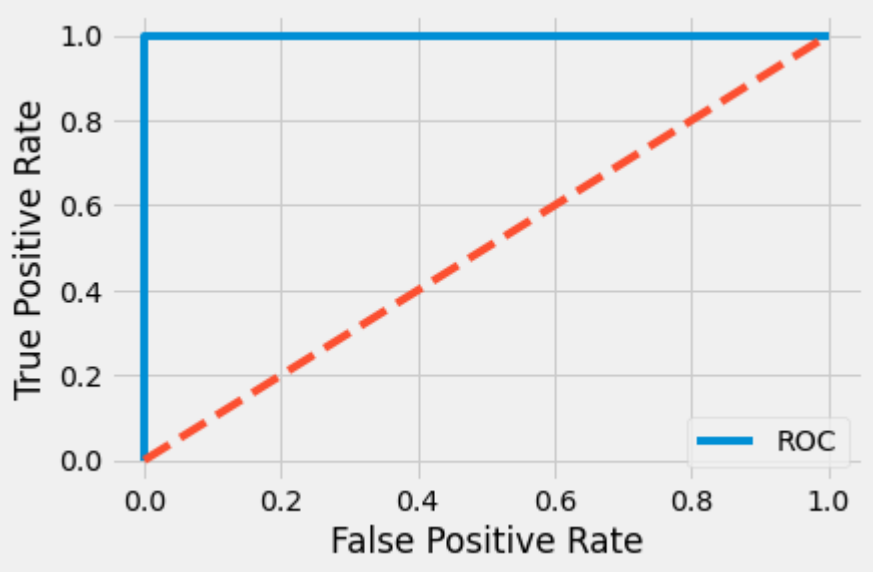
```
Tuned Decision Tree Parameters: {'min_samples_split': 20, 'max_leaf_no
des': 64, 'max_features': 0.6, 'max_depth': 6, 'criterion': 'gini', 'c
lass_weight': {0: 1, 1: 5}}
Best score is 0.9872768581629341
Best estimator is DecisionTreeClassifier(class_weight={0: 1, 1: 5}, ma
x_depth=6, max_features=0.6,
                       max_leaf_nodes=64, min_samples_split=20)
              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 Decision Tree is : 0.9872768581629341
Computation time 0.273 - Sec
```



In [50]:

```
auc = roc_auc_score(y_test, y_pred_dt)
print('AUC: %.2f' % auc)

fpr, tpr, thresholds = roc_curve(y_test, y_pred_dt)
plot_roc_curve(fpr, tpr)
```

```
AUC: 1.00
```

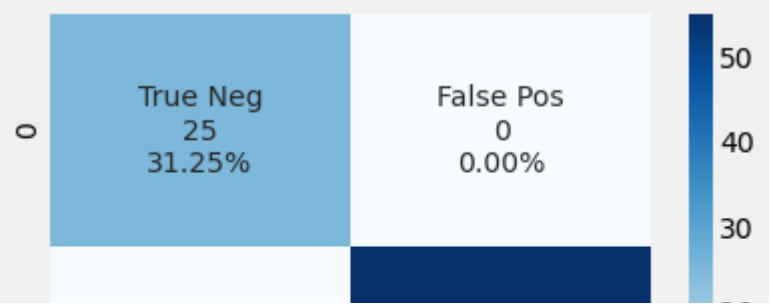


Random Forest Hyper parameter tuning

In [51]:

```
param_grid = {"n_estimators": np.arange(2, 300, 2),
              "max_depth": np.arange(1, 28, 1),
              "min_samples_split": np.arange(1,150,1),
              "min_samples_leaf": np.arange(1,60,1),
              "max_leaf_nodes": np.arange(2,60,1),
              "min_weight_fraction_leaf": np.arange(0.1,0.4, 0.1)}

start_rf = time.time()
rf = RandomizedSearchCV(RandomForestClassifier(),
                        param_grid,
                        scoring='f1',
                        random_state=4658,
                        n_iter=20)

rf_model = rf.fit(X_train, y_train)
end_rf = time.time()
final_rf = end_rf - start_rf
final_rf = round(final_rf,3)
final_rf

# Print the tuned parameters and score
print("Tuned Random Tree Parameters: {}".format(rf_model.best_params_))
print("Best score is {}".format(rf_model.best_score_))
print("Best estimator is {}".format(rf_model.best_estimator_))

y_pred_rf = rf_model.predict(X_test)
display_confusion_matrix(y_test, y_pred_rf)
accuracy_rf=rf_model.best_score_
print("Accuracy of Random Forests model is :", accuracy_rf)
print("Computation time {} - Sec".format(final_rf))
```

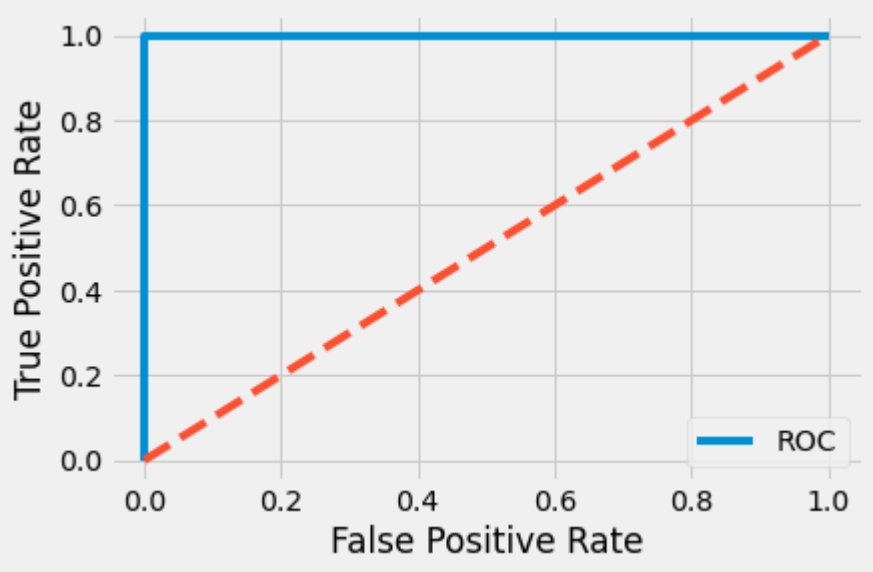
```
Tuned Random Tree Parameters: {'n_estimators': 292, 'min_weight_fracti
on_leaf': 0.30000000000000004, 'min_samples_split': 6, 'min_samples_le
af': 33, 'max_leaf_nodes': 11, 'max_depth': 11}
Best score is 0.9872768581629341
Best estimator is RandomForestClassifier(max_depth=11, max_leaf_nodes=
11, min_samples_leaf=33,
                       min_samples_split=6,
                       min_weight_fraction_leaf=0.30000000000000004,
                       n_estimators=292)
              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 Random Forests model is : 0.9872768581629341
Computation time 22.332 - Sec
```



In [52]:

```
auc = roc_auc_score(y_test, y_pred_rf)
print('AUC: %.2f' % auc)

fpr, tpr, thresholds = roc_curve(y_test, y_pred_rf)
plot_roc_curve(fpr, tpr)
```

```
AUC: 1.00
```



Support Vector Machine Hyper parameter tuning

In [53]:

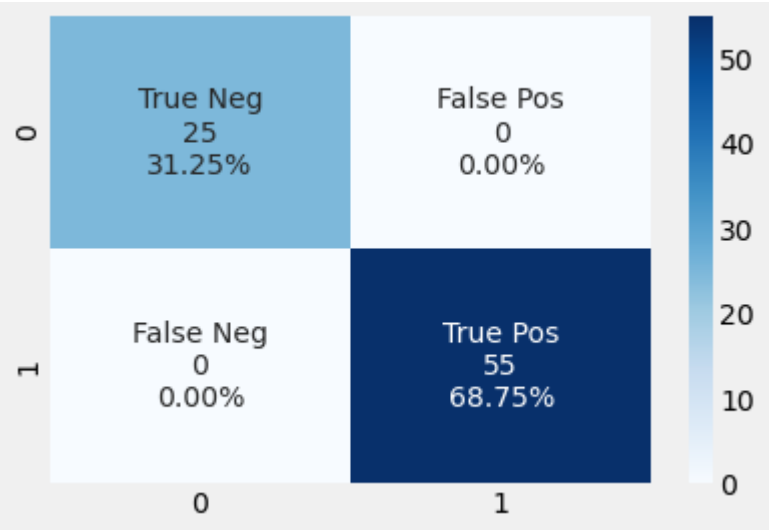
```
# defining parameter range
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001]}

start_svm = time.time()
svm = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3)
# fitting the model for grid search
svm.fit(X_train, y_train)

end_svm = time.time()
final_svm = end_svm - start_svm
final_svm = round(final_svm,3)
final_svm

# Print the tuned parameters and score
print("Tuned Support Vector Machine Parameters: {}".format(svm.best_params_))
print("Best score is {}".format(svm.best_score_))
print("Best estimator is {}".format(svm.best_estimator_))
```

```
[CV] .................. C=0.1, gamma=0.001, score=0.609, total=   0.0
s
[CV] C=0.1, gamma=0.001
.........................................
[CV] .................. C=0.1, gamma=0.001, score=0.609, total=   0.0
s
[CV] C=0.1, gamma=0.001
.........................................
[CV] .................. C=0.1, gamma=0.001, score=0.609, total=   0.0
s
[CV] C=0.1, gamma=0.001
.........................................
[CV] .................. C=0.1, gamma=0.001, score=0.609, total=   0.0
s
[CV] C=0.1, gamma=0.0001
.........................................
[CV] ................. C=0.1, gamma=0.0001, score=0.609, total=   0.0
s
[CV] C=0.1, gamma=0.0001
```

In [54]:

```
y_pred_svm = svm.predict(X_test)
display_confusion_matrix(y_test, y_pred_svm)
accuracy_svm=svm.best_score_
print("Accuracy of Support Vector Machine is :", accuracy_svm)
print("Computation time {} - Sec".format(final_svm))
```

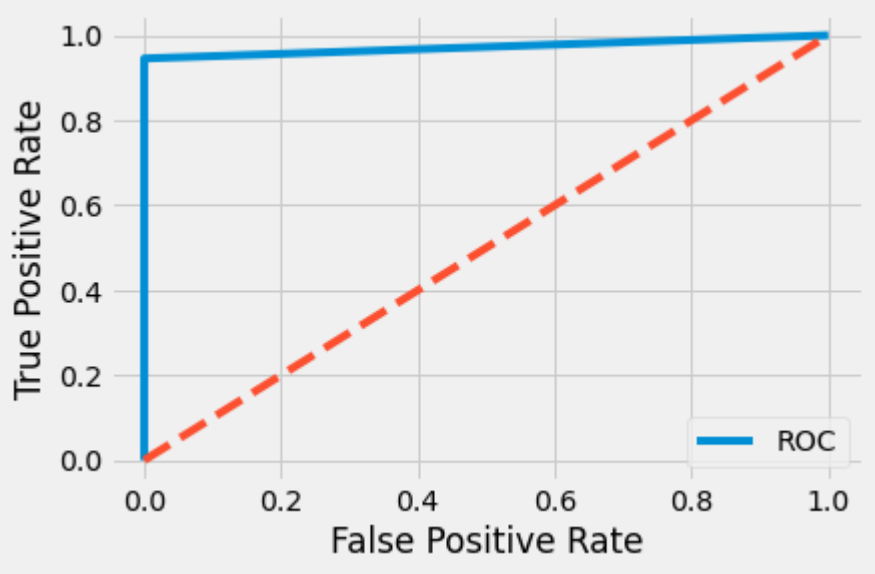
```
              precision    recall  f1-score   support

           0       0.89      1.00      0.94        25
           1       1.00      0.95      0.97        55

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

Accuracy of Support Vector Machine is : 0.9875
Computation time 0.384 - Sec
```



In [55]:

```
auc = roc_auc_score(y_test, y_pred_svm)
print('AUC: %.2f' % auc)

fpr, tpr, thresholds = roc_curve(y_test, y_pred_svm)
plot_roc_curve(fpr, tpr)
```

```
AUC: 0.97
```



Artificial neural network

In [57]:

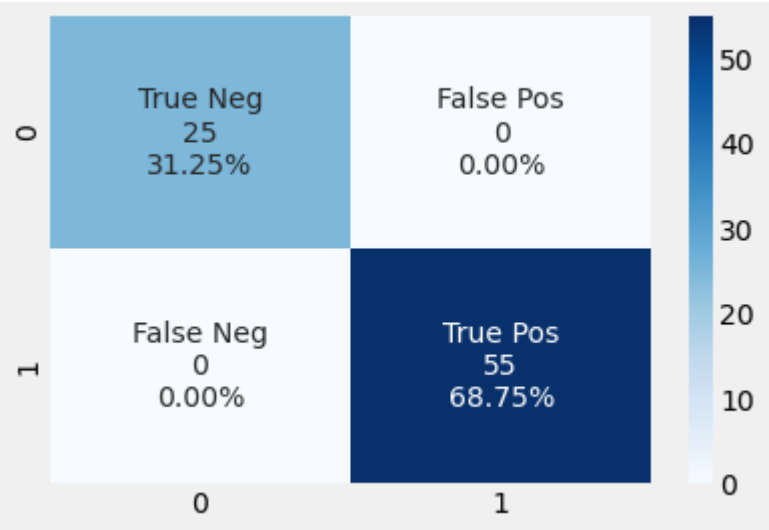
```
# defining parameter range
param_grid = {
               'max_iter': [1000],
             }

start_mlp = time.time()
mlp = GridSearchCV(MLPClassifier(), param_grid, refit = True, verbose = 3)
# fitting the model for grid search
mlp.fit(X_train, y_train.values.ravel())

end_mlp = time.time()
final_mlp = end_mlp - start_mlp
final_mlp = round(final_mlp,3)
final_mlp

# Print the tuned parameters and score
print("Tuned Artificial neural network Parameters: {}".format(mlp.best_params_))
print("Best score is {}".format(mlp.best_score_))
print("Best estimator is {}".format(mlp.best_estimator_))
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV] max_iter=1000 ...................................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurren
t workers.

[CV] ....................... max_iter=1000, score=0.969, total=   0.6s
[CV] max_iter=1000 ...................................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.6s remainin
g:    0.0s

[CV] ....................... max_iter=1000, score=0.984, total=   0.7s
[CV] max_iter=1000 ...................................................

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    1.3s remainin
g:    0.0s

[CV] ....................... max_iter=1000, score=0.984, total=   0.6s
[CV] max_iter=1000 ...................................................
[CV] ....................... max_iter=1000, score=0.969, total=   0.6s
[CV] max_iter=1000 ...................................................
[CV] ....................... max_iter=1000, score=1.000, total=   0.6s

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    3.2s finished

Tuned Artificial neural network Parameters: {'max_iter': 1000}
Best score is 0.98125
Best estimator is MLPClassifier(max_iter=1000)
```

In [58]:

```
y_pred_mlp = mlp.predict(X_test)
display_confusion_matrix(y_test, y_pred_mlp)
accuracy_mlp=mlp.best_score_
print("Accuracy of Artificial neural network  is :", accuracy_mlp)
print("Computation time {} - Sec".format(final_mlp))
```

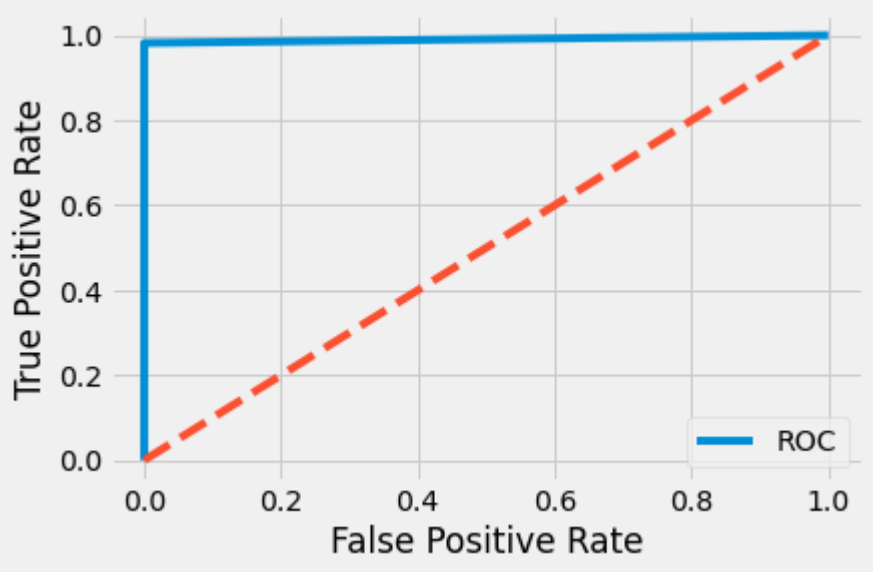
```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        25
           1       1.00      0.98      0.99        55

    accuracy                           0.99        80
   macro avg       0.98      0.99      0.99        80
weighted avg       0.99      0.99      0.99        80

Accuracy of Artificial neural network  is : 0.98125
Computation time 3.76 - Sec
```



In [59]:

```
auc = roc_auc_score(y_test, y_pred_mlp)
print('AUC: %.2f' % auc)

fpr, tpr, thresholds = roc_curve(y_test, y_pred_mlp)
plot_roc_curve(fpr, tpr)
```

```
AUC: 0.99
```



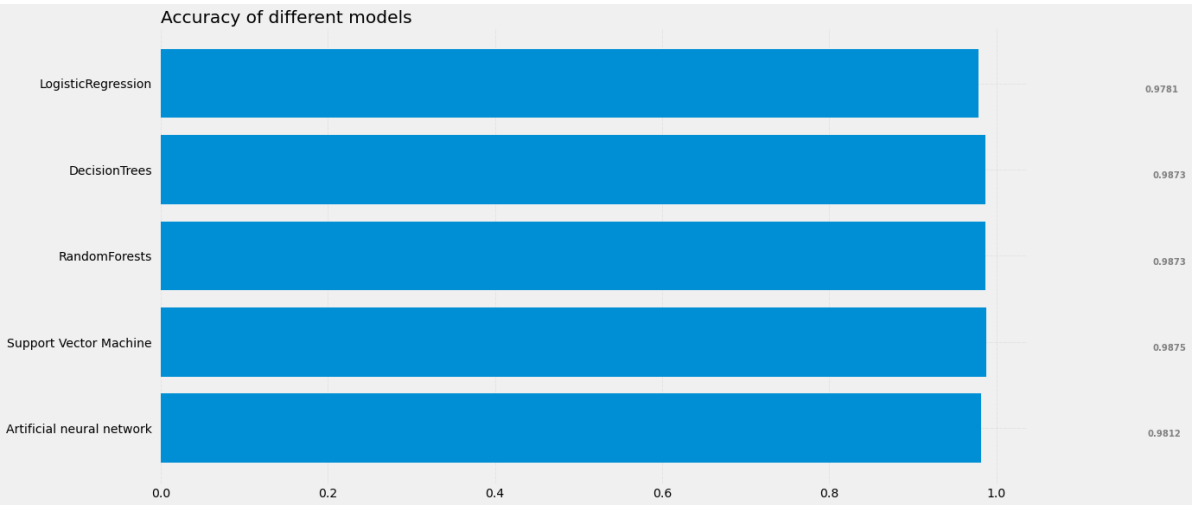
In [62]:

```
accuracies2 = [lr.best_score_,dt_model.best_score_,rf_model.best_score_, svm.best_sc
final_time2 = [final_lr,
               final_dt,
               final_rf,
               final_svm,
               final_mlp]
print(accuracies2)
print(final_time2)
```

```
[0.978125, 0.9872768581629341, 0.9872768581629341, 0.9875, 0.98125]
[0.509, 0.273, 22.332, 0.384, 3.76]
```

In [63]:

```
# Figure Size
fig, ax = plt.subplots(figsize =(16, 9))
# Horizontal Bar Plot
ax.barh(models, accuracies2)

# Remove axes splines
for s in ['top', 'bottom', 'left', 'right']:
    ax.spines[s].set_visible(False)
# Remove x, y Ticks
ax.xaxis.set_ticks_position('none')
ax.yaxis.set_ticks_position('none')

# Add padding between axes and labels
ax.xaxis.set_tick_params(pad = 5)
ax.yaxis.set_tick_params(pad = 10)
# Add x, y gridlines
ax.grid(b = True, color ='grey',
        linestyle ='-.', linewidth = 0.5,
        alpha = 0.2)

# Show top values
ax.invert_yaxis()

# Add annotation to bars
for i in ax.patches:
    plt.text(i.get_width()+0.2, i.get_y()+0.5,
             str(round((i.get_width()), 4)),
             fontsize = 10, fontweight ='bold',
             color ='grey')
ax.set_title('Accuracy of different models', loc ='left')
plt.show()
```



In [ ]: