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', '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	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	ckd	ckd	ckd	ckd	ckd	ckd	ckd	ckd	ckd	ckd	ckd

Data DeepDive

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

In [ ]:
```
#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())
```

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

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

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

In [10]:
```
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[10]:
```
['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']
```

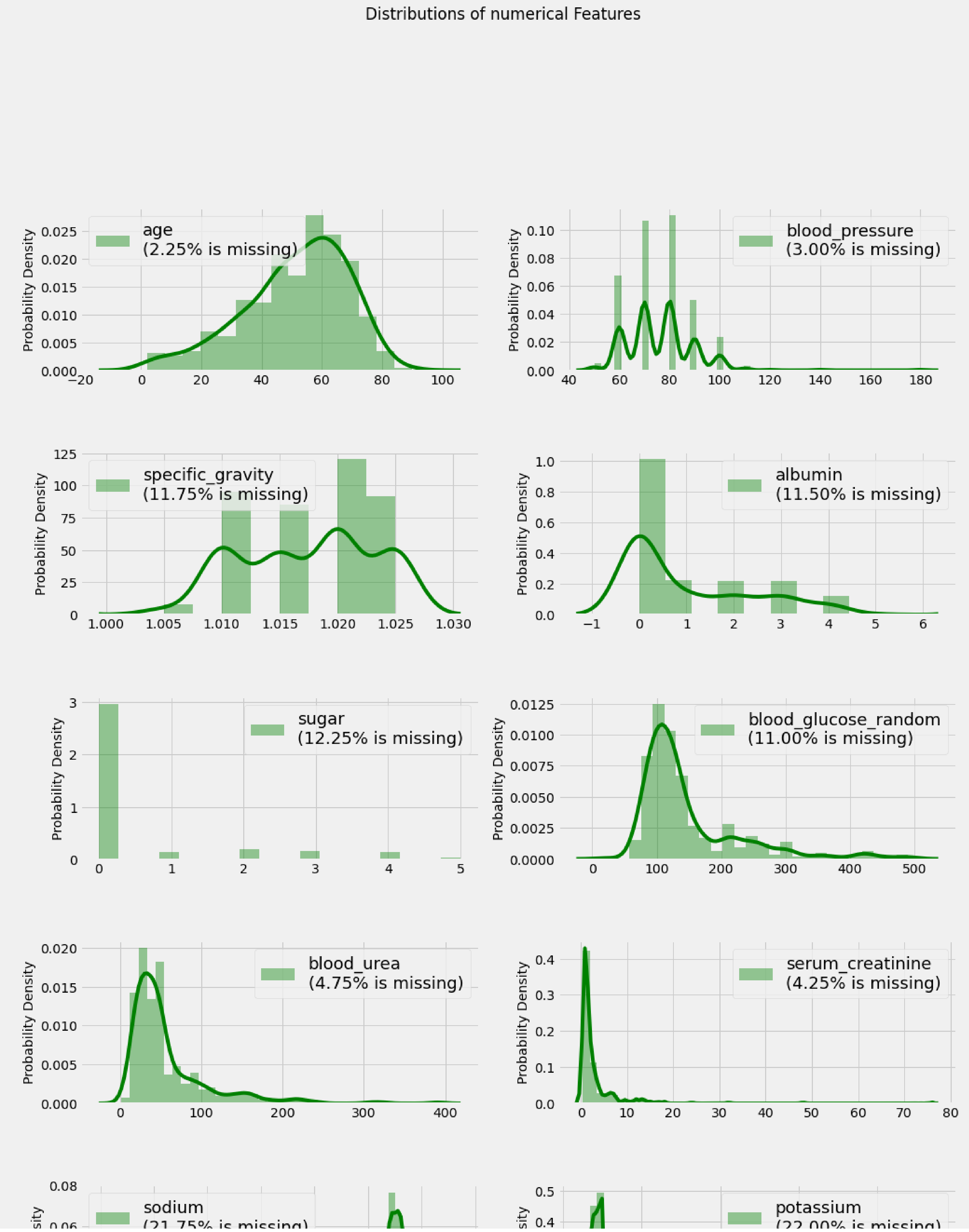
In [11]:
```
ckd_df.drop('id',axis=1,inplace=True)

categoricals=[]

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

categoricals
```

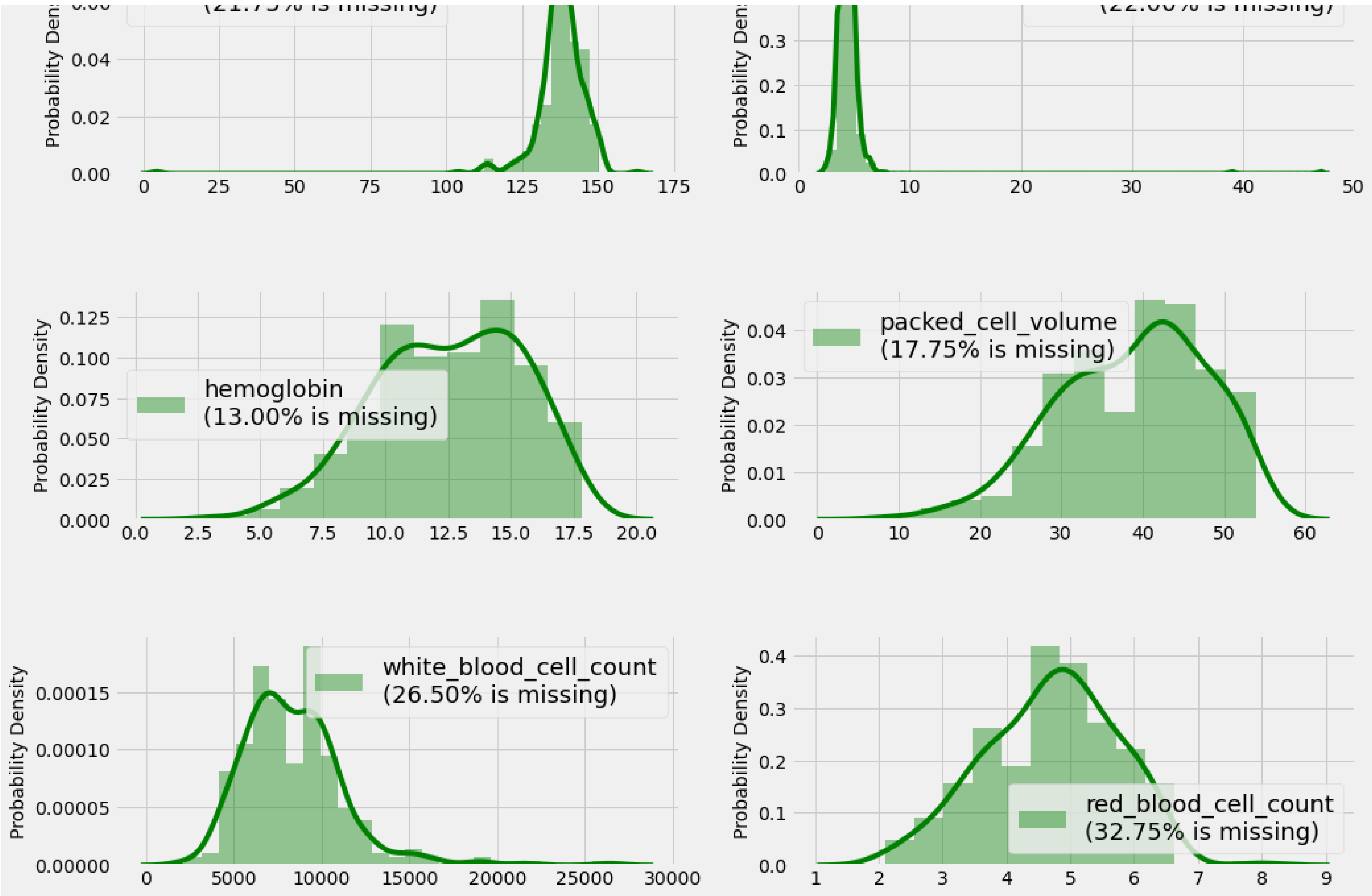
Out[11]:
```
['red_blood_cells',
 'pus_cell',
 'pus_cell_clumps',
 'bacteria',
 'hypertension',
 'diabetes_mellitus',
 'coronary_artery_disease',
 'appetite',
 'pedal_edema',
 'anemia']
```

In [12]:
```
import warnings
warnings.simplefilter('ignore')

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

Checking distribution of the numerical features

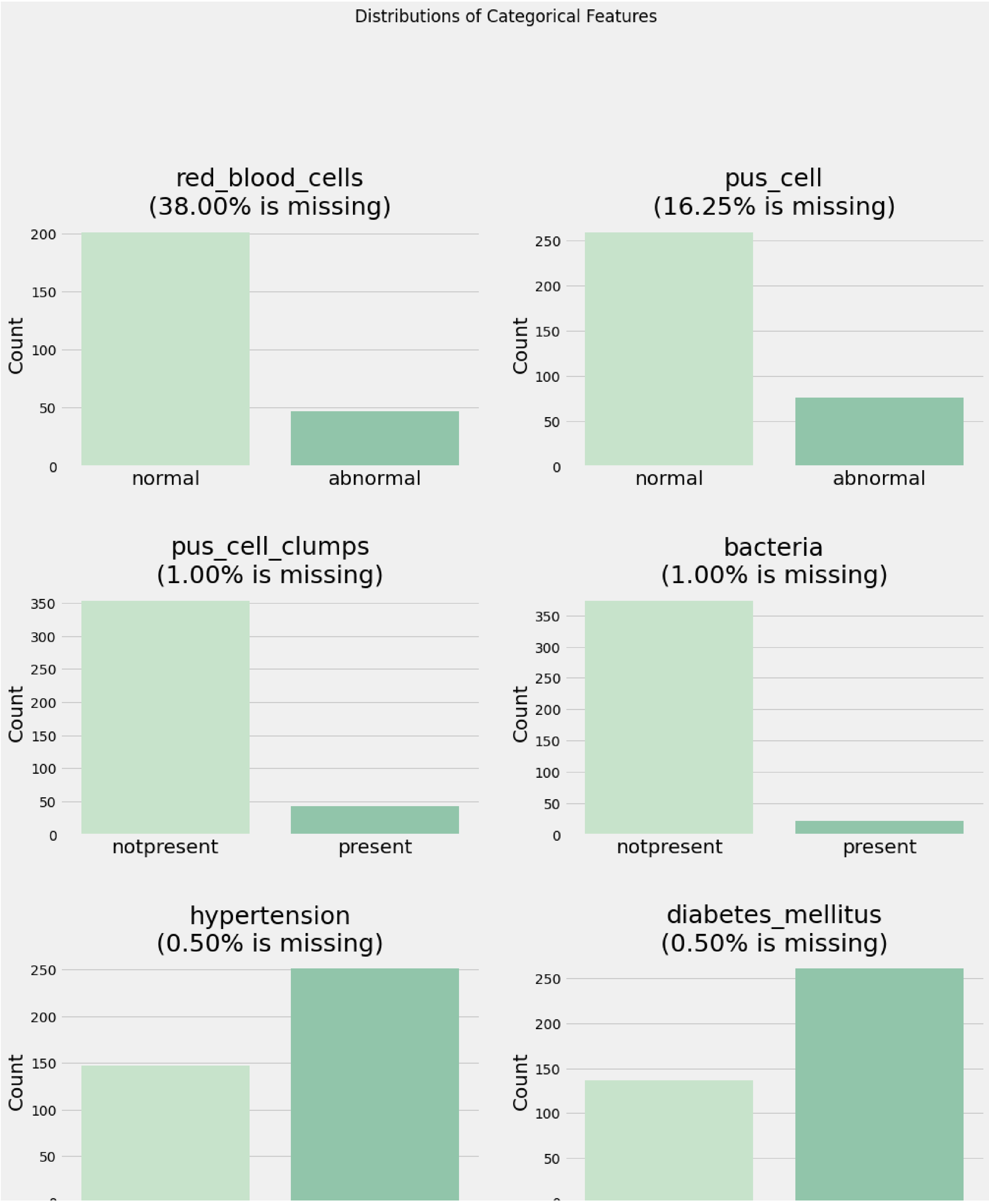
In [13]:
```
fig, axes = plt.subplots(nrows=7, ncols=2, figsize=(15,30))
fig.subplots_adjust(hspace=0.5)
fig.suptitle('Distributions of numerical Features')

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()
```





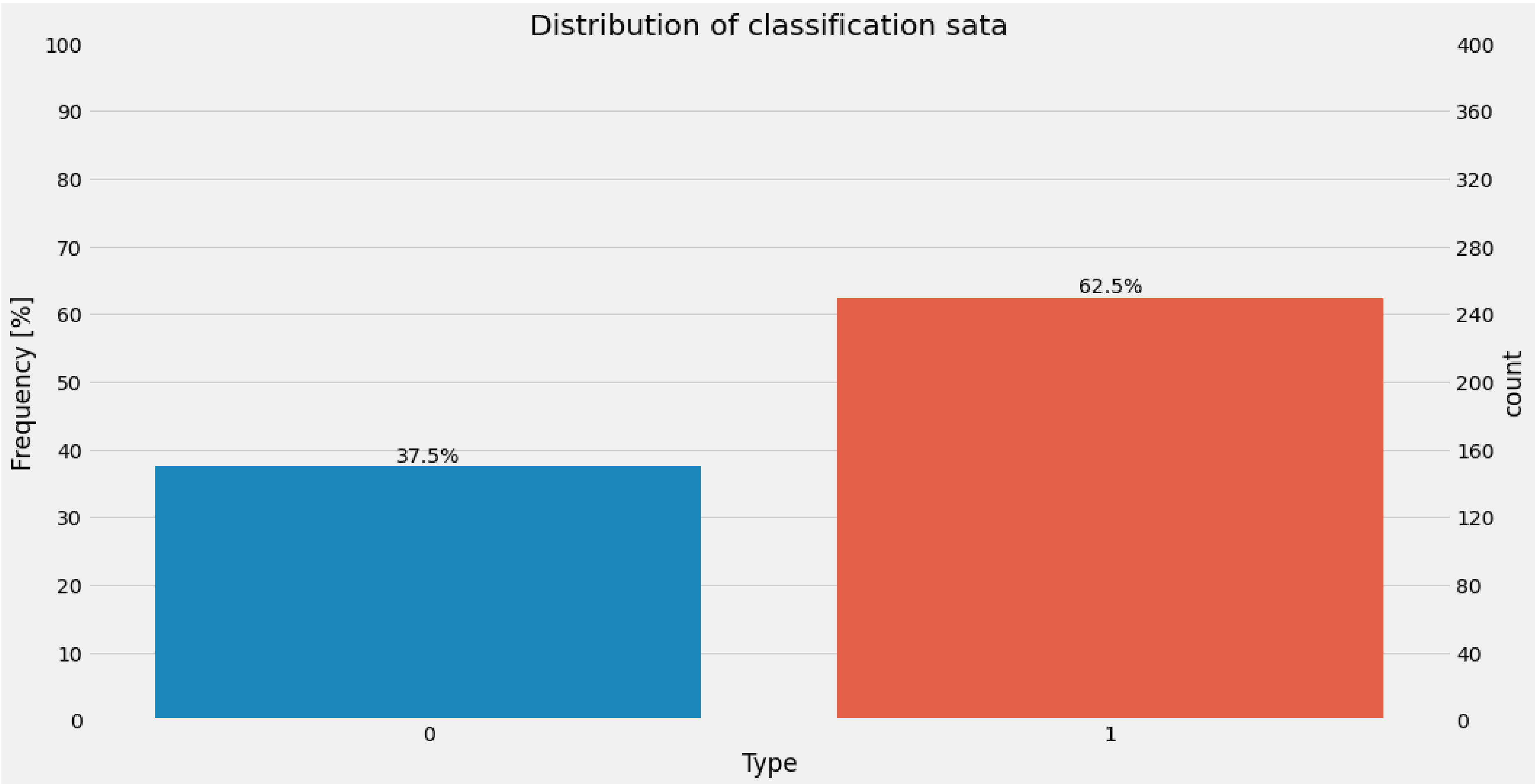
Checking distribution of the Categorical features

In [14]:
```
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])

    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 [15]:
```
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)
```

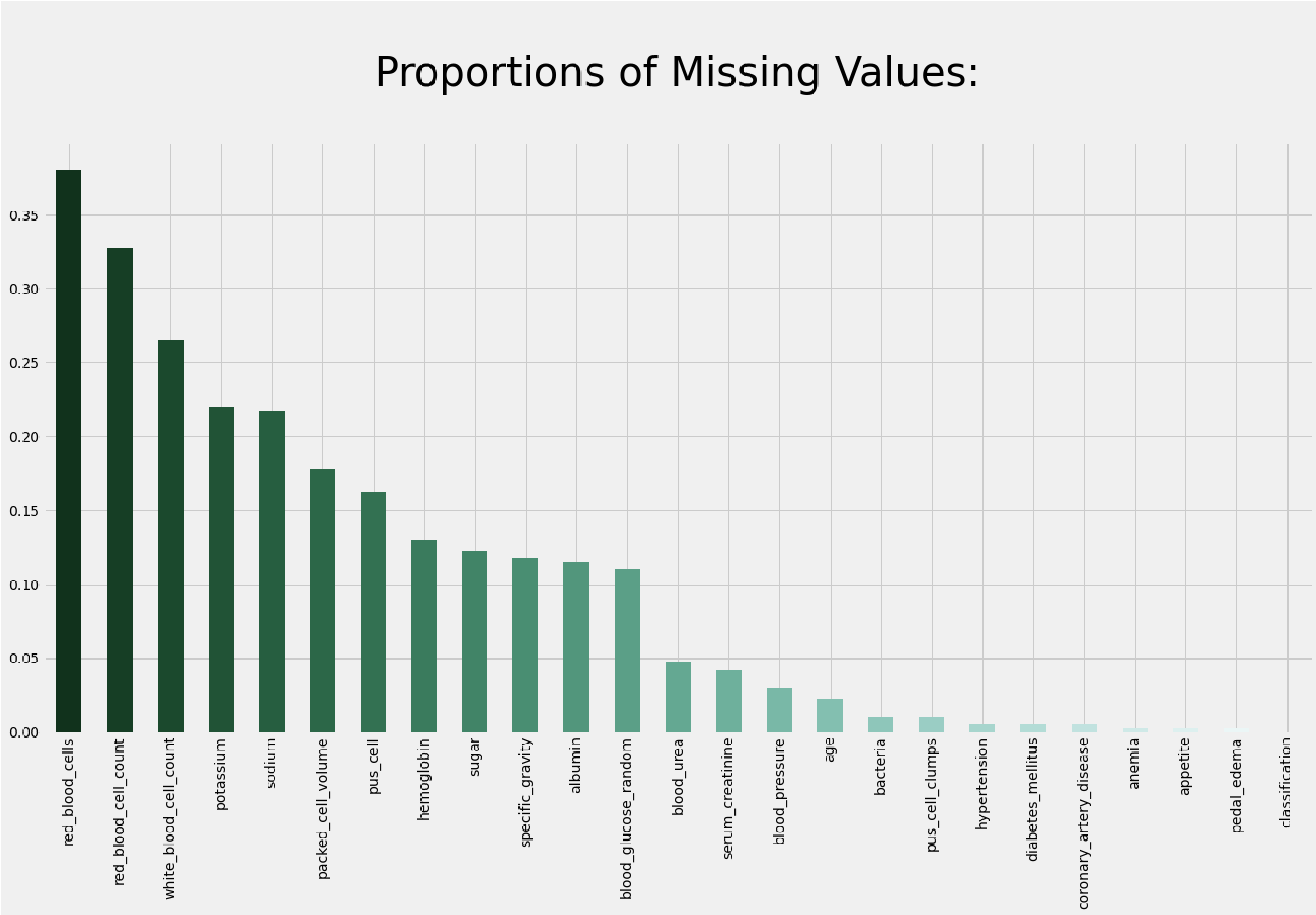


In [16]:
```
for i in range(ckd_df.shape[0]):
    if ckd_df.iloc[i,24]=='ckd':
        ckd_df.iloc[i,24]='1'
    if ckd_df.iloc[i,24]=='notckd':
        ckd_df.iloc[i,24]='0'
```

In [17]:
```python
g = sns.pairplot(ckd_df, vars = numeric ,hue = 'classification')
g.map_diag(sns.distplot)
g.add_legend()
g.fig.suptitle('FacetGrid plot', fontsize = 20)
g.fig.subplots_adjust(top= 0.9);
```

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()
```



One-Hot Encoding

In [19]:
```python
onehotdata=pd.get_dummies(ckd_df,drop_first=True,prefix_sep=': ')
onehotdata.head(13).T
```

Out[19]:

	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 [20]:
```python
# 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)
```

```
['sodium', 'serum_creatinine', 'sugar', 'diabetes_mellitus: yes', 'albumin', 'pus_cell_clumps: present', 'cor
onary_artery_disease: yes', 'appetite: poor', 'red_blood_cells: normal', 'anemia: yes', 'pus_cell: normal',
'red_blood_cell_count', 'bacteria: present', 'blood_urea', 'pedal_edema: yes', 'potassium', 'age', 'white_blo
od_cell_count', 'hypertension: yes', 'hemoglobin', 'specific_gravity', 'blood_pressure', 'blood_glucose_rando
m', 'packed_cell_volume']
```

In [21]:
```python
imputer.fit(onehotdata[impute_columns])
```

Out[21]:
```
KNNImputer()
```

In [22]:
```python
X_trans=pd.DataFrame(imputer.transform(onehotdata[impute_columns]), columns=impute_columns)
```

In [23]:
```
X_trans.head(13).T
```
Out[23]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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

Modelling

In [24]:
```
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 [25]:
```
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 [26]:
```
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,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 [27]:
```python
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()
```

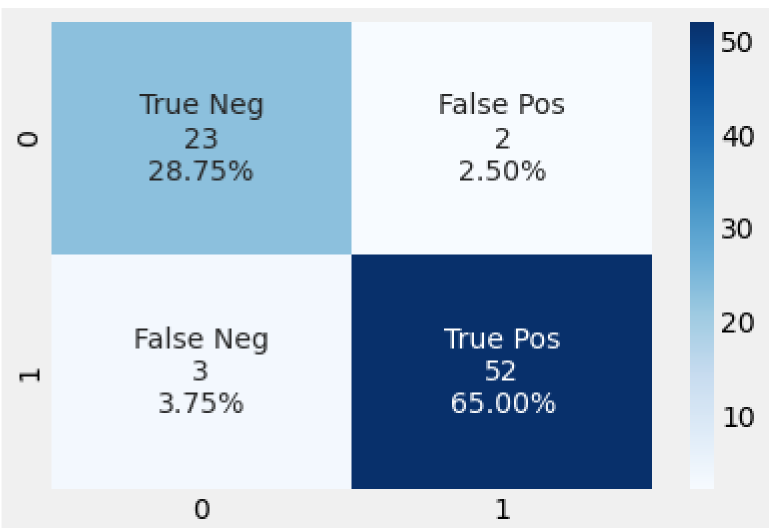
In [28]:
```python
##Split train and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 4658)

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

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

Logistic Regression Hyper parameter tuning

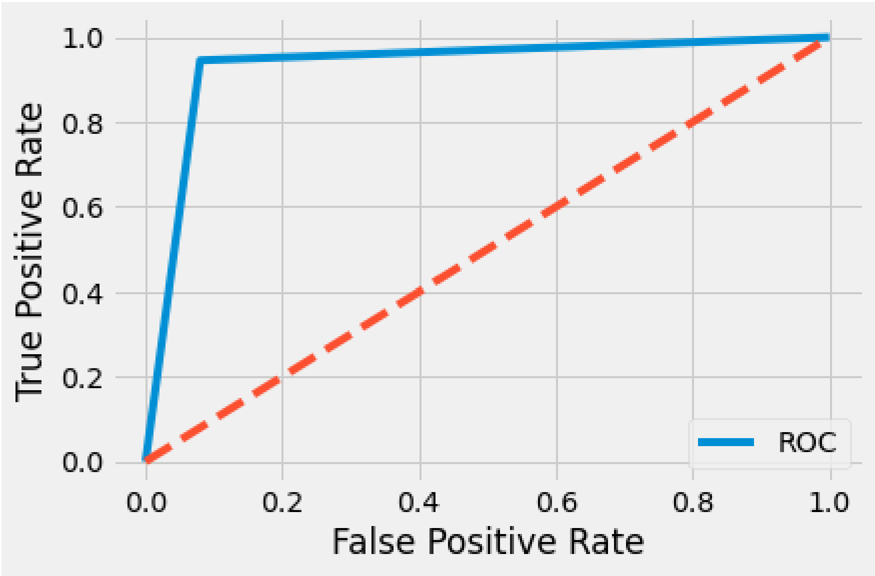
In [29]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

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': 0.05179474679231213}
Best score is 0.9125
Best estimator is LogisticRegression(C=0.05179474679231213)


              precision    recall  f1-score   support

           0       0.88      0.92      0.90        25
           1       0.96      0.95      0.95        55

    accuracy                           0.94        80
   macro avg       0.92      0.93      0.93        80
weighted avg       0.94      0.94      0.94        80


Accuracy of Logistic Regression is : 0.9375
Computation time 3.255 - Sec
```



In [30]:
```
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)
```

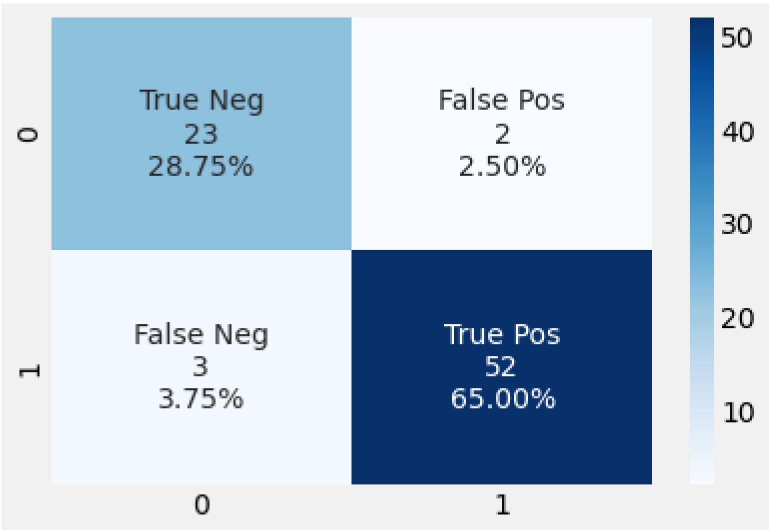
```
AUC: 0.93
```



Decision Tree Hyper parameter tuning

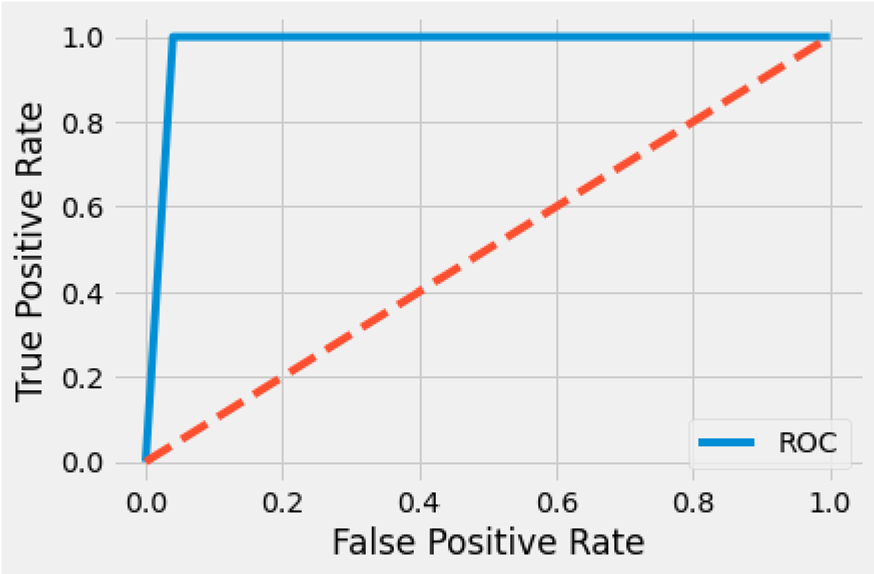
In [31]:
```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import RandomizedSearchCV

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, 1: 5}]
}

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_nodes': 16, 'max_features': 0.4, 'max_dept
h': 8, 'criterion': 'entropy', 'class_weight': {0: 1, 1: 1}}
Best score is 0.9689451476793248
Best estimator is DecisionTreeClassifier(class_weight={0: 1, 1: 1}, criterion='entropy',
                       max_depth=8, max_features=0.4, max_leaf_nodes=16)
              precision    recall  f1-score   support

           0       1.00      0.96      0.98        25
           1       0.98      1.00      0.99        55

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

Accuracy of Decision Tree is : 0.9875
Computation time 0.699 - Sec
```



In [32]:
```
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)
```

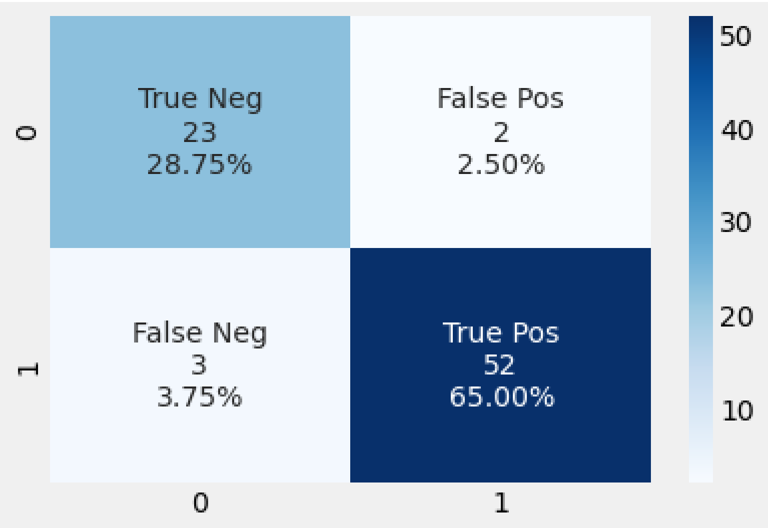
```
AUC: 0.98
```



Random Forest Hyper parameter tuning

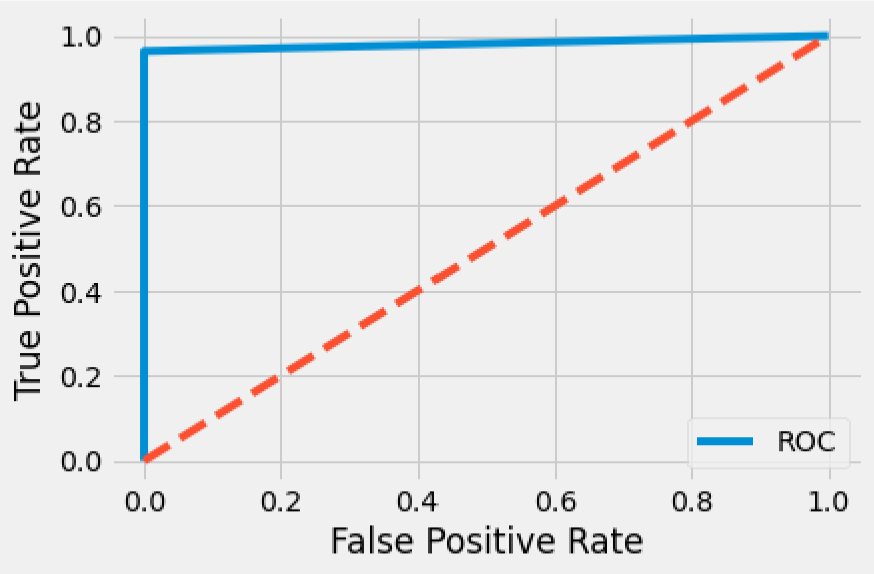
In [33]:
```python
from sklearn.ensemble import RandomForestClassifier
param_grid = {"n_estimators": np.arange(2, 300, 2),
              "max_depth": np.arange(1, 28, 1),
              "min_samples_split": np.arange(1,150,1),
              "min_samples_leaf": np.arange(1,60,1),
              "max_leaf_nodes": 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': 168, 'min_weight_fraction_leaf': 0.2, 'min_samples_split': 41,
'min_samples_leaf': 28, 'max_leaf_nodes': 52, 'max_depth': 17}
Best score is 0.9742257742257742
Best estimator is RandomForestClassifier(max_depth=17, max_leaf_nodes=52, min_samples_leaf=28,
                       min_samples_split=41, min_weight_fraction_leaf=0.2,
                       n_estimators=168)
              precision    recall  f1-score   support

           0       0.93      1.00      0.96        25
           1       1.00      0.96      0.98        55

    accuracy                           0.97        80
   macro avg       0.96      0.98      0.97        80
weighted avg       0.98      0.97      0.98        80

Accuracy of Random Forests model is : 0.975
Computation time 21.179 - Sec
```



In [34]:
```python
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: 0.98
```



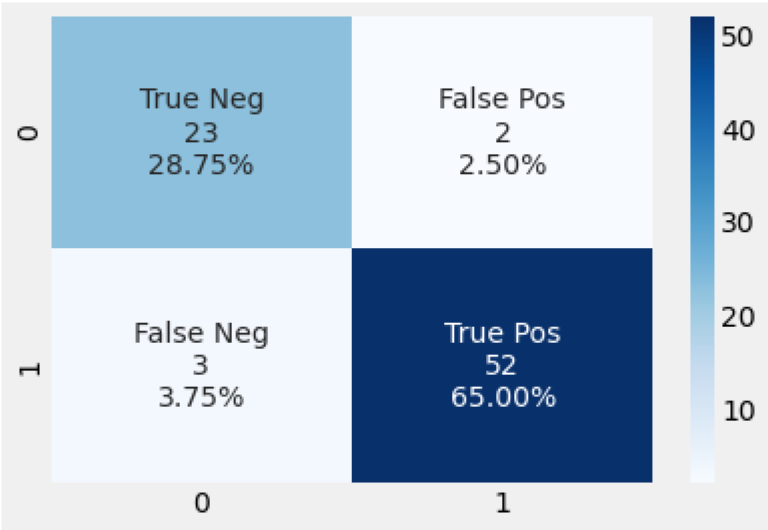
Support Vector Machine Hyper parameter tuning

In [35]:
```python
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],
              'kernel': ['rbf']}

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, kernel=rbf ......................................
[CV] ......... C=0.1, gamma=1, kernel=rbf, score=0.609, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf ......................................
[CV] ......... C=0.1, gamma=1, kernel=rbf, score=0.609, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf ......................................
[CV] ......... C=0.1, gamma=1, kernel=rbf, score=0.609, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf ......................................
[CV] ......... C=0.1, gamma=1, kernel=rbf, score=0.609, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf ......................................
[CV] ......... C=0.1, gamma=1, kernel=rbf, score=0.609, total=   0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf ....................................
[CV] ....... C=0.1, gamma=0.1, kernel=rbf, score=0.609, total=   0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf ....................................
[CV] ....... C=0.1, gamma=0.1, kernel=rbf, score=0.609, total=   0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf ....................................
[CV] ....... C=0.1, gamma=0.1, kernel=rbf, score=0.609, total=   0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf ....................................
[CV] ....... C=0.1, gamma=0.1, kernel=rbf, score=0.609, total=   0.0s
```

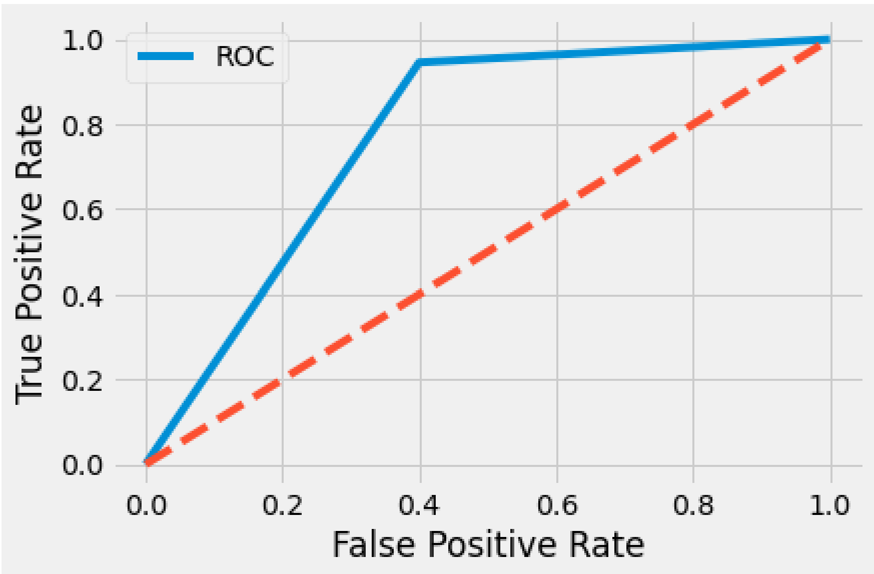
```
In [36]: y_pred_svm = svm.predict(X_test)
display_confusion_matrix(y_test, y_pred_svm)
accuracy_svm=accuracy_score(y_test, y_pred_svm)
print("Accuracy of Support Vector Machine is :", accuracy_svm)
print("Computation time {} - Sec".format(final_svm))
```

```
              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 1.123 - Sec
```




```
In [37]: 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
```



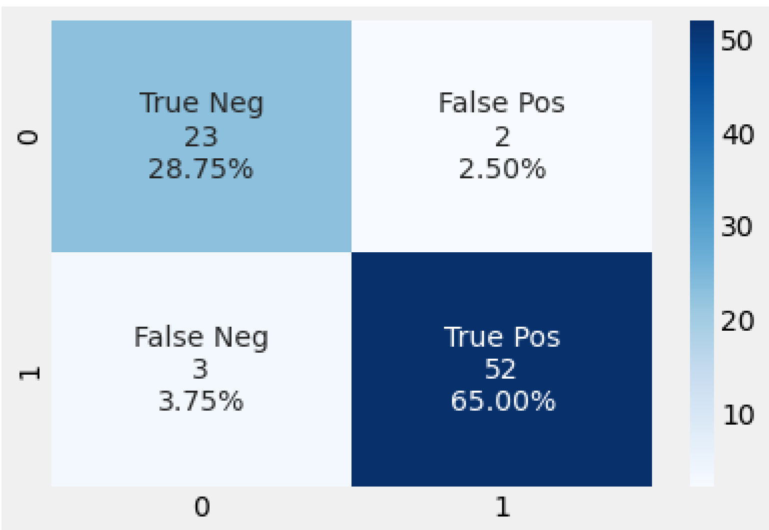
Artificial neural network

In [38]:
```
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=0.812, total=   0.2s
[CV] max_iter=1000 ...................................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.2s remaining:    0.0s

[CV] ....................... max_iter=1000, score=0.703, total=   0.2s
[CV] max_iter=1000 ...................................................
[CV] ....................... max_iter=1000, score=0.828, total=   0.1s
[CV] max_iter=1000 ...................................................

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.4s remaining:    0.0s

[CV] ....................... max_iter=1000, score=0.844, total=   0.1s
[CV] max_iter=1000 ...................................................
[CV] ....................... max_iter=1000, score=0.594, total=   0.1s
Tuned Artificial neural network Parameters: {'max_iter': 1000}
Best score is 0.75625
Best estimator is MLPClassifier(max_iter=1000)

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

In [39]:
```
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       0.54      1.00      0.70        25
           1       1.00      0.62      0.76        55

    accuracy                           0.74        80
   macro avg       0.77      0.81      0.73        80
weighted avg       0.86      0.74      0.75        80

Accuracy of Artificial neural network  is : 0.7375
Computation time 0.836 - Sec
```



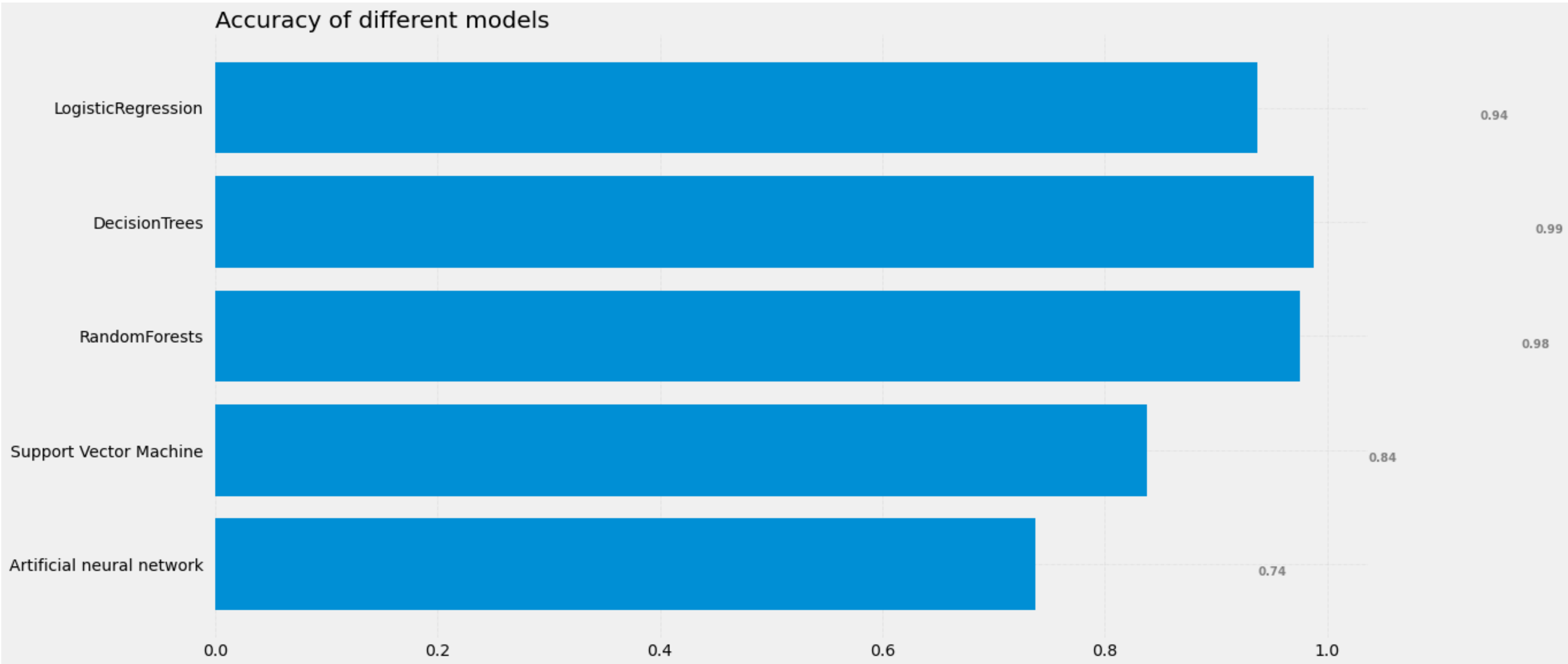
In [40]:
```
auc = roc_auc_score(y_test, y_pred_mlp)
print('AUC: %.2f' % auc)

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

```
AUC: 0.81
```

In [41]:
```
accuracies1 = [accuracy_lr,accuracy_dt,accuracy_rf, accuracy_svm, accuracy_mlp]
final_time1 = [final_lr,final_dt,final_rf, final_svm, final_mlp]
print(accuracies1)
print(final_time1)

models= ['LogisticRegression', 'DecisionTrees', 'RandomForests', 'Support Vector Machine', 'Artificial neural ne
```

```
[0.9375, 0.9875, 0.975, 0.8375, 0.7375]
[3.255, 0.699, 21.179, 1.123, 0.836]
```

In [42]:
```python
# 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 [43]:
```python
# performing preprocessing part
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

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

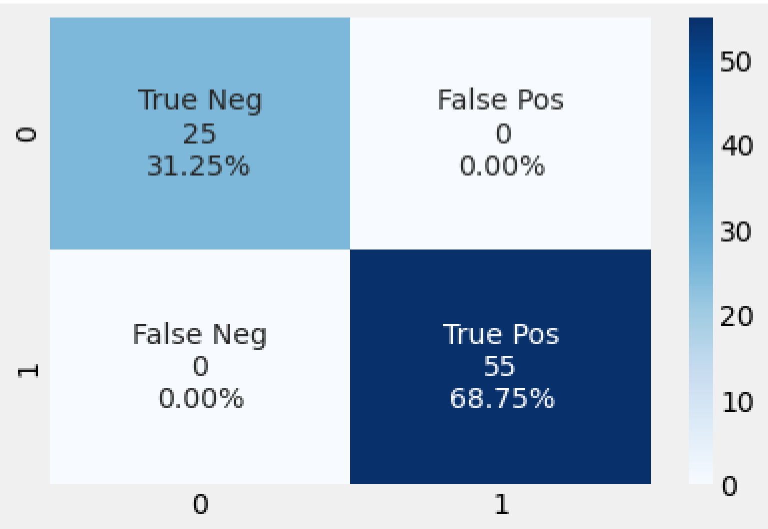
In [44]:
```python
# Applying PCA function on training and testing set of X component
from sklearn.decomposition import PCA

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[44]:
```
array([0.30943656, 0.07796664])
```

Logistic Regression Hyper parameter tuning

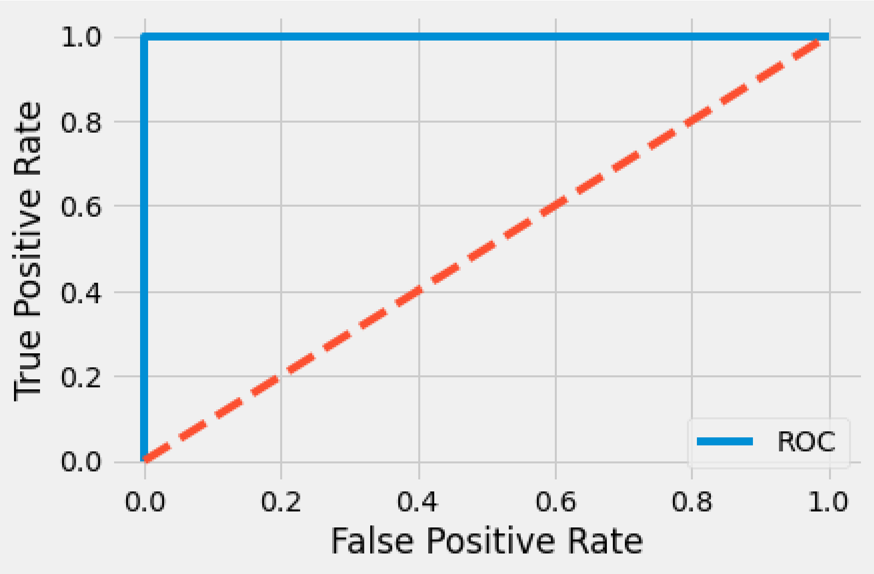
In [45]:
```python
c_space = np.logspace(-5, 8, 15)
param_grid = {'C': c_space}

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': 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 : 1.0
Computation time 0.407 - Sec
```



In [46]:
```
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)
```

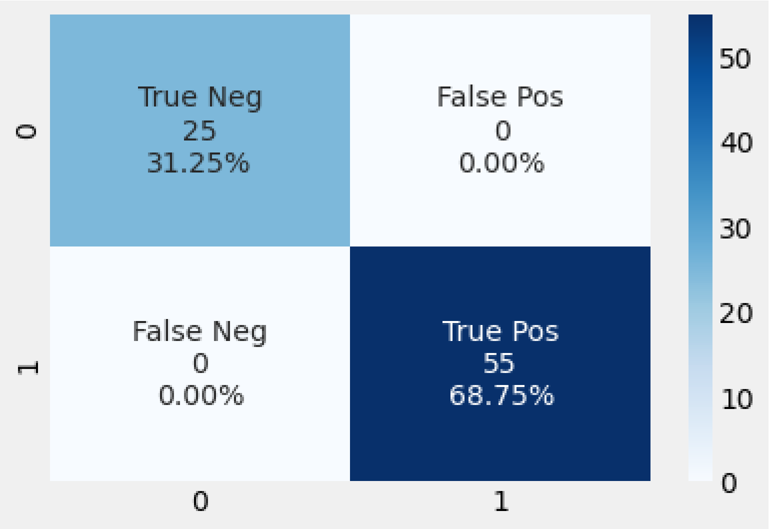
```
AUC: 1.00
```



Decision Tree Hyper parameter tuning

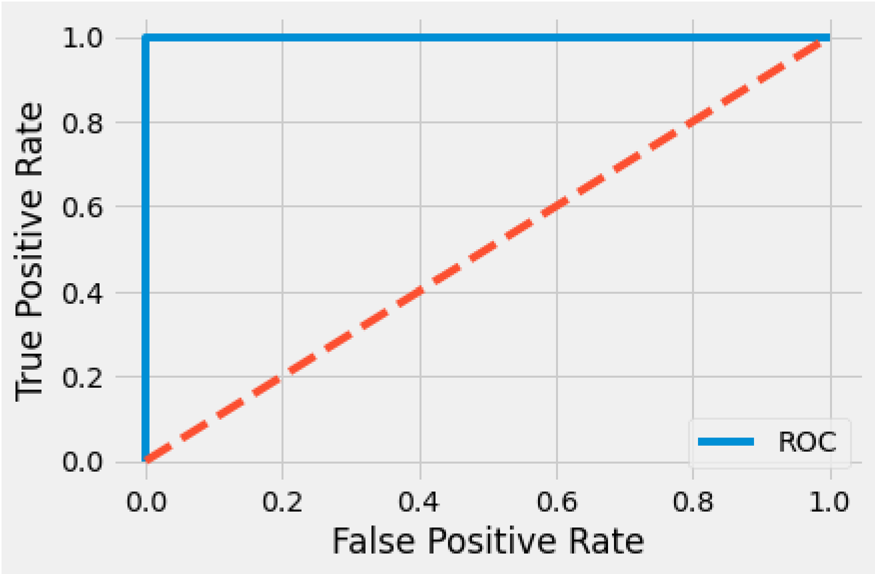
In [47]:
```python
hyperparam_combs = {
    'max_depth': [4, 6, 8, 10, 12],
    'criterion': ['gini', 'entropy'],
    'min_samples_split': [2, 10, 20, 30, 40],
    'max_features': [0.2, 0.4, 0.6, 0.8, 1],
    'max_leaf_nodes': [8, 16, 32, 64, 128],
    'class_weight': [{0: 1, 1: 1}, {0: 1, 1: 2}, {0: 1, 1: 3}, {0: 1, 1: 4}, {0: 1, 1: 5}]
}

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': 20, 'max_leaf_nodes': 64, 'max_features': 0.6, 'max_dep
th': 6, 'criterion': 'gini', 'class_weight': {0: 1, 1: 5}}
Best score is 0.9872768581629341
Best estimator is DecisionTreeClassifier(class_weight={0: 1, 1: 5}, max_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 : 1.0
Computation time 0.225 - Sec
```



In [48]:
```
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)
```

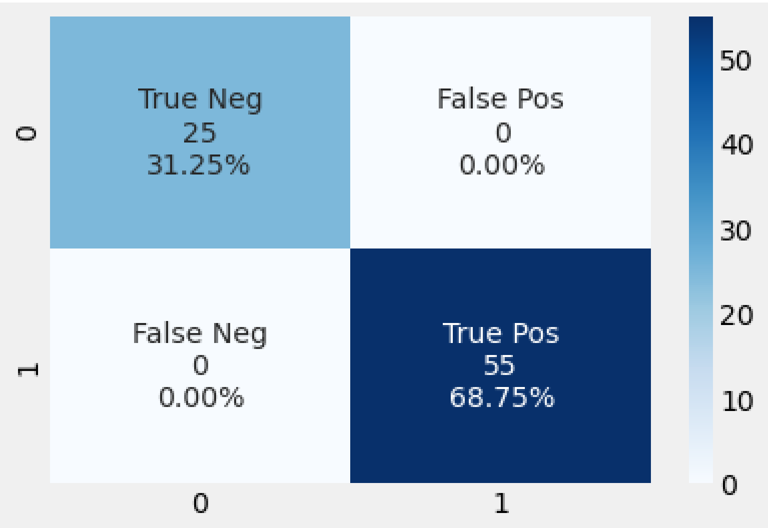
```
AUC: 1.00
```



Random Forest Hyper parameter tuning

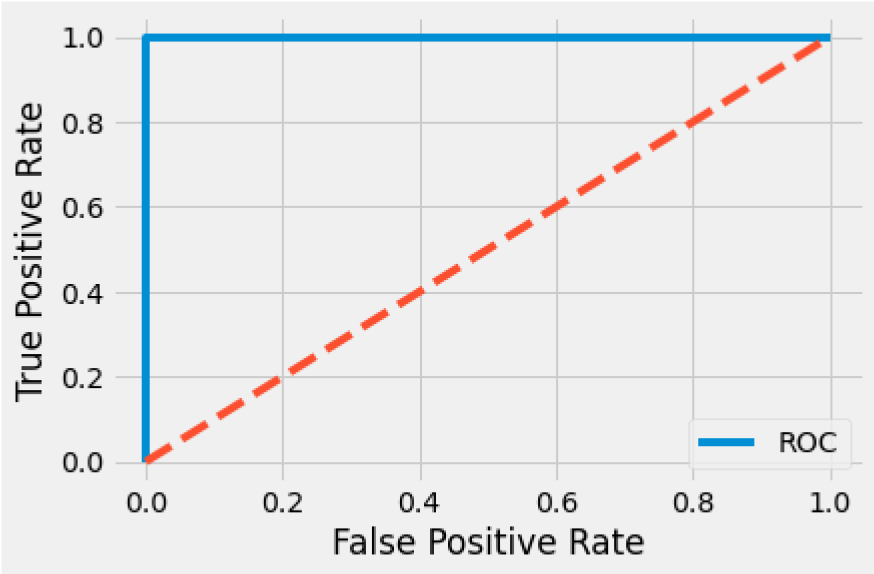
In [49]:
```python
param_grid = {"n_estimators": np.arange(2, 300, 2),
              "max_depth": np.arange(1, 28, 1),
              "min_samples_split": np.arange(1,150,1),
              "min_samples_leaf": np.arange(1,60,1),
              "max_leaf_nodes": np.arange(2,60,1),
              "min_weight_fraction_leaf": 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': 240, 'min_weight_fraction_leaf': 0.1, 'min_samples_split': 52,
'min_samples_leaf': 38, 'max_leaf_nodes': 39, 'max_depth': 26}
Best score is 0.9872768581629341
Best estimator is RandomForestClassifier(max_depth=26, max_leaf_nodes=39, min_samples_leaf=38,
                       min_samples_split=52, min_weight_fraction_leaf=0.1,
                       n_estimators=240)
              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 : 1.0
Computation time 20.392 - Sec
```



In [50]:
```python
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
```



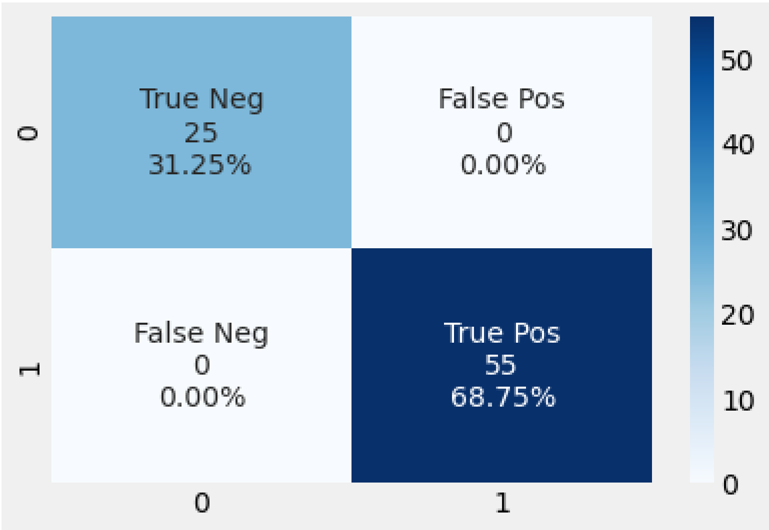
Support Vector Machine Hyper parameter tuning

In [51]:
```python
# defining parameter range
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf']}

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, kernel=rbf ......................................
[CV] ......... C=0.1, gamma=1, kernel=rbf, score=0.984, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf ......................................
[CV] ......... C=0.1, gamma=1, kernel=rbf, score=0.984, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf ......................................
[CV] ......... C=0.1, gamma=1, kernel=rbf, score=0.984, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf ......................................
[CV] ......... C=0.1, gamma=1, kernel=rbf, score=0.969, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf ......................................
[CV] ......... C=0.1, gamma=1, kernel=rbf, score=1.000, total=   0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf ....................................
[CV] ....... C=0.1, gamma=0.1, kernel=rbf, score=0.953, total=   0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf ....................................
[CV] ....... C=0.1, gamma=0.1, kernel=rbf, score=0.984, total=   0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf ....................................
[CV] ....... C=0.1, gamma=0.1, kernel=rbf, score=0.969, total=   0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf ....................................
[CV] ....... C=0.1, gamma=0.1, kernel=rbf, score=0.953, total=   0.0s
```

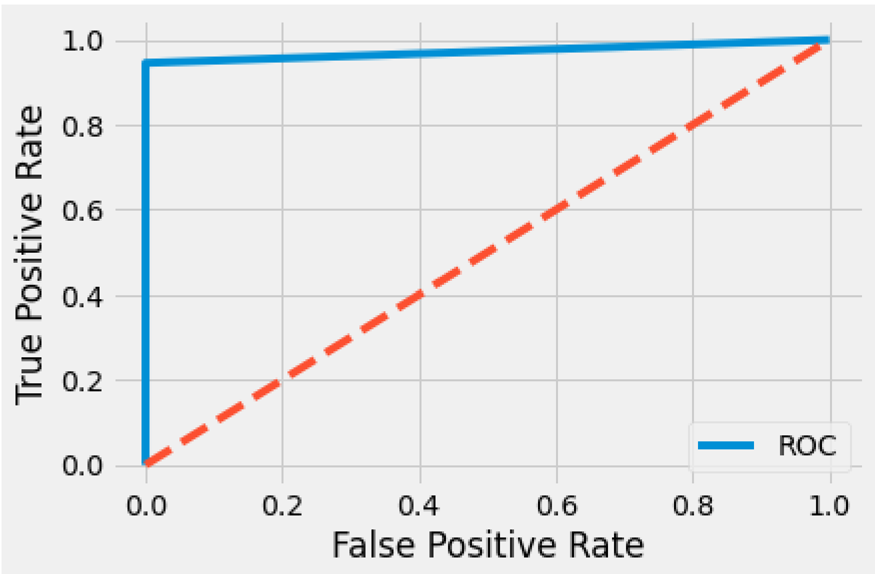
In [52]:
```
y_pred_svm = svm.predict(X_test)
display_confusion_matrix(y_test, y_pred_svm)
accuracy_svm=accuracy_score(y_test, y_pred_svm)
print("Accuracy of Support Vector Machine is :", accuracy_svm)
print("Computation time {} - Sec".format(final_svm))
```

```
              precision    recall  f1-score   support

           0       0.89      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 : 0.9625
Computation time 0.359 - Sec
```



In [53]:
```
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
```



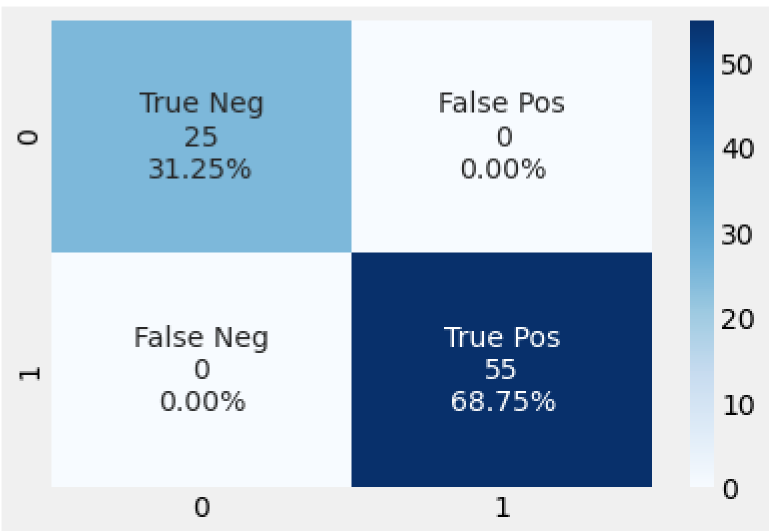
Artificial neural network

In [54]:
```
# 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 concurrent workers.

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

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

[CV] ....................... max_iter=1000, score=0.969, 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=   0.5s

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    2.9s finished

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

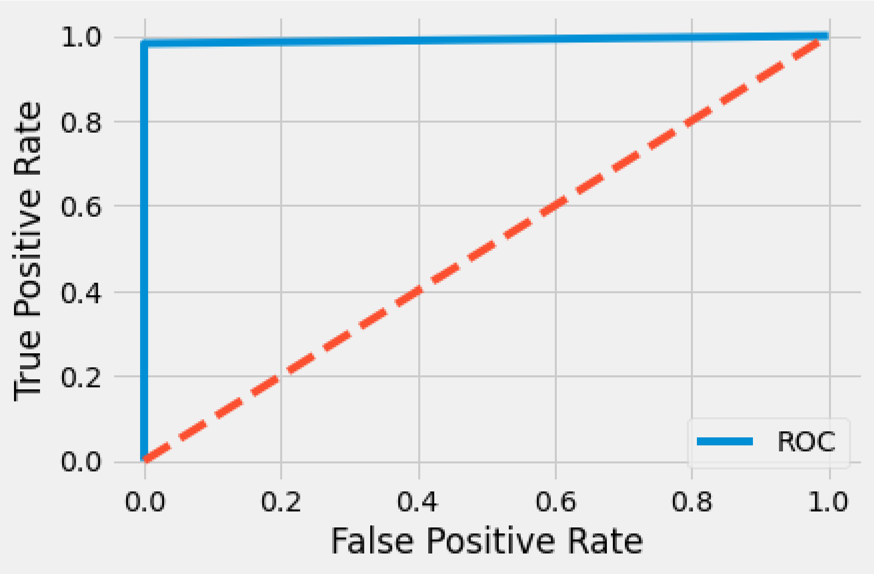
In [55]:
```
y_pred_mlp = mlp.predict(X_test)
display_confusion_matrix(y_test, y_pred_mlp)
accuracy_mlp=accuracy_score(y_test, y_pred_mlp)
print("Accuracy of Artificial neural network  is :", accuracy_mlp)
print("Computation time {} - Sec".format(final_mlp))
```

```
              precision    recall  f1-score   support

           0       0.96      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 : 0.9875
Computation time 3.391 - Sec
```



In [56]:
```
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 [57]:
```
accuracies2 = [accuracy_lr,accuracy_dt,accuracy_rf, accuracy_svm, accuracy_mlp]
final_time2 = [final_lr,final_dt,final_rf, final_svm, final_mlp]
print(accuracies2)
print(final_time2)
```

```
[1.0, 1.0, 1.0, 0.9625, 0.9875]
[0.407, 0.225, 20.392, 0.359, 3.391]
```

In [58]:
```
# 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()), 2)),
            fontsize = 10, fontweight ='bold',
            color ='grey')
ax.set_title('Accuracy of different models', loc ='left')
plt.show()
```