

Alzheimer_Prediction_with_Demographics_Data_NestedCV

November 4, 2021

```
[1]: import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sn

from numpy import mean

from sklearn import preprocessing
from sklearn.metrics import log_loss
from sklearn.metrics import confusion_matrix
import sklearn.metrics as metrics
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import classification_report

from sklearn.model_selection import GridSearchCV, cross_val_score, KFold

%matplotlib inline
```

Load The data files Display data frame head

```
[2]: data_frame = pd.read_excel('C:/Users/amran/Downloads/
    ↳oasis_longitudinal_demographics.xlsx')

data_frame.head()
```

```
[2]:
```

	Subject ID	MRI ID	Group	Visit	MR Delay	M/F	Hand	Age	EDUC	\
0	OAS2_0001	OAS2_0001_MR1	Nondemented	1	0	M	R	87	14	
1	OAS2_0001	OAS2_0001_MR2	Nondemented	2	457	M	R	88	14	
2	OAS2_0002	OAS2_0002_MR1	Demented	1	0	M	R	75	12	
3	OAS2_0002	OAS2_0002_MR2	Demented	2	560	M	R	76	12	
4	OAS2_0002	OAS2_0002_MR3	Demented	3	1895	M	R	80	12	

	SES	MMSE	CDR	eTIV	nWBV	ASF
0	2.0	27.0	0.0	1986.550000	0.696106	0.883440
1	2.0	30.0	0.0	2004.479526	0.681062	0.875539
2	NaN	23.0	0.5	1678.290000	0.736336	1.045710
3	NaN	28.0	0.5	1737.620000	0.713402	1.010000

```
4 NaN 22.0 0.5 1697.911134 0.701236 1.033623
```

Data pre-processing and selection Lets first look at columns data types

```
[3]: data_frame.dtypes
```

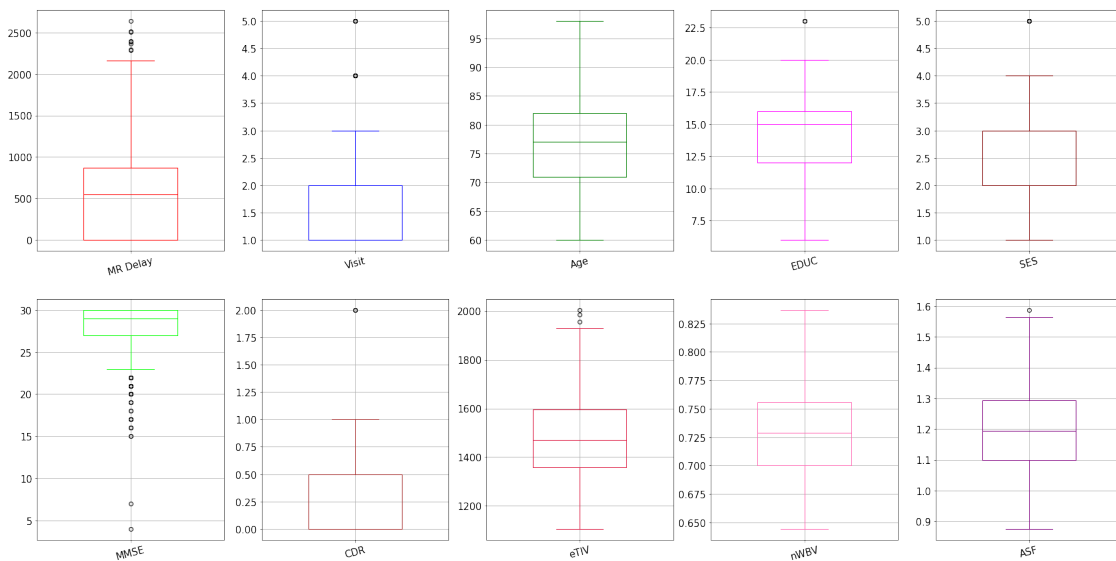
```
[3]: Subject ID      object
MRI ID             object
Group              object
Visit              int64
MR Delay           int64
M/F                object
Hand               object
Age                int64
EDUC               int64
SES                float64
MMSE               float64
CDR                float64
eTIV               float64
nWBV               float64
ASF                float64
dtype: object
```

```
[4]: data_frame.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 373 entries, 0 to 372
Data columns (total 15 columns):
#   Column          Non-Null Count  Dtype
---  -
0   Subject ID      373 non-null   object
1   MRI ID          373 non-null   object
2   Group           373 non-null   object
3   Visit           373 non-null   int64
4   MR Delay        373 non-null   int64
5   M/F             373 non-null   object
6   Hand            373 non-null   object
7   Age             373 non-null   int64
8   EDUC            373 non-null   int64
9   SES             354 non-null   float64
10  MMSE            371 non-null   float64
11  CDR             373 non-null   float64
12  eTIV            373 non-null   float64
13  nWBV            373 non-null   float64
14  ASF             373 non-null   float64
dtypes: float64(6), int64(4), object(5)
memory usage: 43.8+ KB
```

```
[5]: fig,axs = plt.subplots(2,5,figsize =(30,15))
boxplot_mrdelay = data_frame.boxplot(column=['MR Delay'], rot=15, widths=.
↳5,fontsize=15,color = 'red',ax=axs[0,0])
boxplot_visit =data_frame.boxplot(column=['Visit'], rot=15, fontsize=15,widths=.
↳5,color = 'blue',ax=axs[0,1])
boxplot_Age =data_frame.boxplot(column=['Age'], rot=15, fontsize=15,widths=.
↳5,color = 'green',ax=axs[0,2])
boxplot_EDUC = data_frame.boxplot(column=['EDUC'], rot=15, fontsize=15,widths=.
↳5,color = 'magenta',ax=axs[0,3])
boxplot_SES = data_frame.boxplot(column=['SES'], rot=15, fontsize=15,widths=.
↳5,color = 'maroon',ax=axs[0,4])
boxplot_MMSE = data_frame.boxplot(column=['MMSE'], rot=15, fontsize=15,widths=.
↳5,color = 'lime',ax=axs[1,0])
boxplot_CDR = data_frame.boxplot(column=['CDR'], rot=15, fontsize=15,widths=.
↳5,color = 'brown',ax=axs[1,1])
boxplot_eTIV = data_frame.boxplot(column=['eTIV'], rot=15, fontsize=15,widths=.
↳5,color = 'crimson',ax=axs[1,2])
boxplot_nWBV = data_frame.boxplot(column=['nWBV'], rot=15, fontsize=15,widths=.
↳5,color = 'hotpink',ax=axs[1,3])
boxplot_ASF = data_frame.boxplot(column=['ASF'], rot=15, fontsize=15,widths=.
↳5,color = 'purple',ax=axs[1,4])

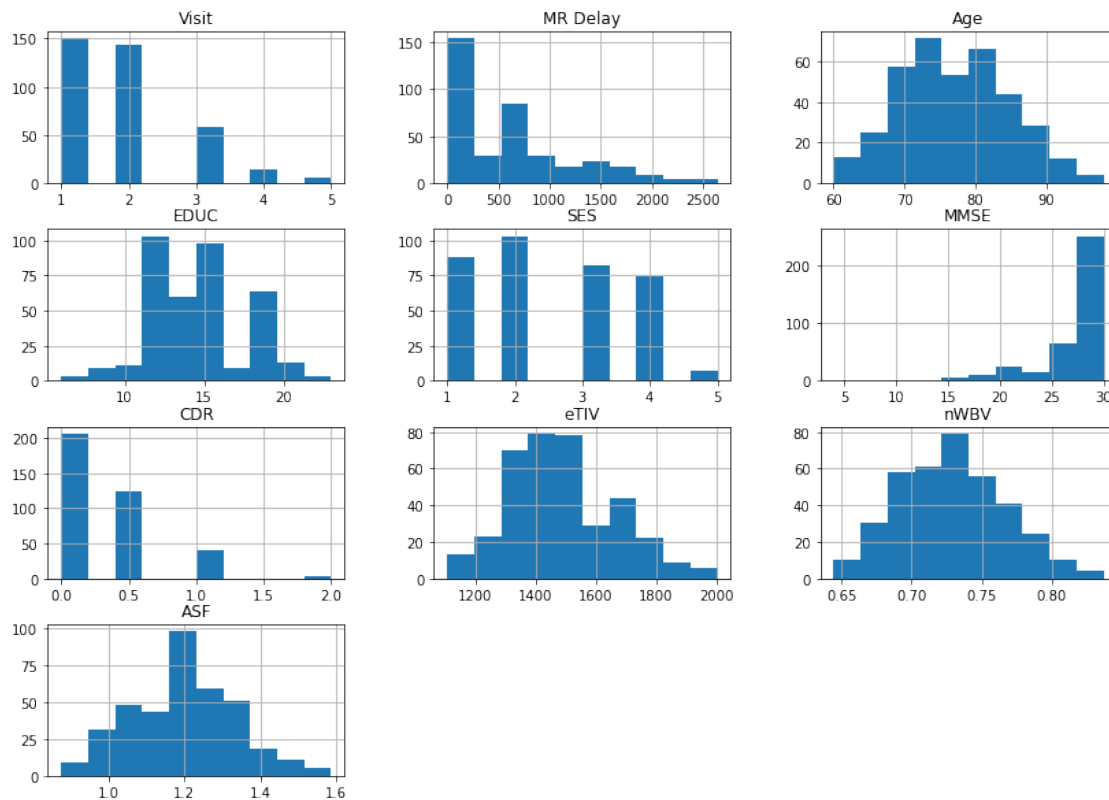
plt.show()
```



Histogram

```
[6]: data_frame.hist(figsize =(14,10))
```

```
[6]: array([[<AxesSubplot:title={'center':'Visit'}>,
<AxesSubplot:title={'center':'MR Delay'}>,
<AxesSubplot:title={'center':'Age'}>],
[<AxesSubplot:title={'center':'EDUC'}>,
<AxesSubplot:title={'center':'SES'}>,
<AxesSubplot:title={'center':'MMSE'}>],
[<AxesSubplot:title={'center':'CDR'}>,
<AxesSubplot:title={'center':'eTIV'}>,
<AxesSubplot:title={'center':'nWBV'}>],
[<AxesSubplot:title={'center':'ASF'}>, <AxesSubplot:>,
<AxesSubplot:>]], dtype=object)
```



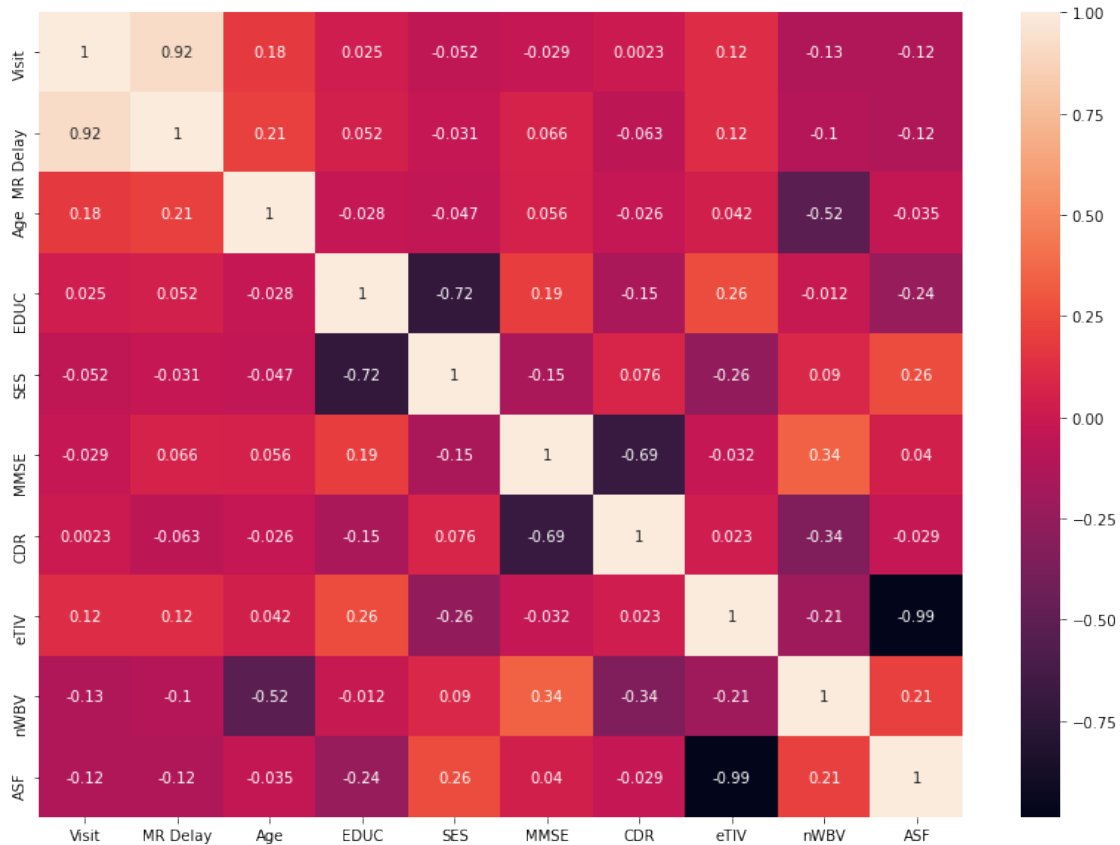
Correlation Heatmap

```
[7]: import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize = (14,10))

sns.heatmap(data_frame[data_frame.columns].corr(), annot=True)
```

```
[7]: <AxesSubplot:>
```



Dealing the missing values

```
[8]: missing_data = data_frame.isnull()
missing_data.head()
```

```
[8]:
```

	Subject ID	MRI ID	Group	Visit	MR Delay	M/F	Hand	Age	EDUC	\
0	False	False	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	False	

	SES	MMSE	CDR	eTIV	nWBV	ASF
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	True	False	False	False	False	False
3	True	False	False	False	False	False
4	True	False	False	False	False	False

Count missing values in each column

```
[9]: for column in missing_data.columns.values.tolist():  
      print(column)  
      print(missing_data[column].value_counts())  
      print(" ")
```

Subject ID
False 373
Name: Subject ID, dtype: int64

MRI ID
False 373
Name: MRI ID, dtype: int64

Group
False 373
Name: Group, dtype: int64

Visit
False 373
Name: Visit, dtype: int64

MR Delay
False 373
Name: MR Delay, dtype: int64

M/F
False 373
Name: M/F, dtype: int64

Hand
False 373
Name: Hand, dtype: int64

Age
False 373
Name: Age, dtype: int64

EDUC
False 373
Name: EDUC, dtype: int64

SES
False 354
True 19
Name: SES, dtype: int64

MMSE
False 371

True 2
Name: MMSE, dtype: int64

CDR
False 373
Name: CDR, dtype: int64

eTIV
False 373
Name: eTIV, dtype: int64

nWBV
False 373
Name: nWBV, dtype: int64

ASF
False 373
Name: ASF, dtype: int64

SES have 19 missing value and MMSE have 2 missing Value replace missing values

```
[10]: # check the details value of SES  
data_frame['SES'].value_counts()
```

```
[10]: 2.0    103  
      1.0     88  
      3.0     82  
      4.0     74  
      5.0      7  
      Name: SES, dtype: int64
```

```
[11]: #Here 2 is most common values. use the ".idxmax()" method to calculate for us  
      ↳ the most common type automatically:
```

```
data_frame['SES'].value_counts().idxmax()
```

```
[11]: 2.0
```

```
[12]: #replace null with most common values  
data_frame['SES'].fillna(2.0, inplace=True)
```

```
[13]: #check the details value of MMSE  
data_frame['MMSE'].value_counts()
```

```
[13]: 30.0    114  
      29.0     91  
      28.0     45
```

```

27.0    32
26.0    20
25.0    12
23.0    11
21.0    11
20.0     7
22.0     7
17.0     5
24.0     4
16.0     3
19.0     3
18.0     2
15.0     2
 7.0     1
 4.0     1
Name: MMSE, dtype: int64

```

```

[14]: #30 is the most common value. We can also use the ".idxmax()" method to
      ↪ calculate for us the most common type automatically:
      data_frame['MMSE'].value_counts().idxmax()

```

```

[14]: 30.0

```

```

[15]: #replace null with most common values
      data_frame['MMSE'].fillna(30,inplace=True)

```

```

[16]: missing_data = data_frame.isnull()
      missing_data.head()

```

```

[16]:   Subject ID  MRI ID  Group  Visit  MR Delay  M/F  Hand  Age  EDUC  \
0      False   False  False  False    False  False  False  False  False
1      False   False  False  False    False  False  False  False  False
2      False   False  False  False    False  False  False  False  False
3      False   False  False  False    False  False  False  False  False
4      False   False  False  False    False  False  False  False  False

      SES  MMSE  CDR  eTIV  nWBV  ASF
0  False  False  False  False  False  False
1  False  False  False  False  False  False
2  False  False  False  False  False  False
3  False  False  False  False  False  False
4  False  False  False  False  False  False

```

```

[17]: missing_data.info()

```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 373 entries, 0 to 372
Data columns (total 15 columns):

```


#	Column	Non-Null Count	Dtype
0	Subject ID	373 non-null	bool
1	MRI ID	373 non-null	bool
2	Group	373 non-null	bool
3	Visit	373 non-null	bool
4	MR Delay	373 non-null	bool
5	M/F	373 non-null	bool
6	Hand	373 non-null	bool
7	Age	373 non-null	bool
8	EDUC	373 non-null	bool
9	SES	373 non-null	bool
10	MMSE	373 non-null	bool
11	CDR	373 non-null	bool
12	eTIV	373 non-null	bool
13	nWBV	373 non-null	bool
14	ASF	373 non-null	bool

dtypes: bool(15)

memory usage: 5.6 KB

```
[18]: for column in missing_data.columns.values.tolist():
        print(column)
        print(missing_data[column].value_counts())
        print(" ")
```

Subject ID

False 373

Name: Subject ID, dtype: int64

MRI ID

False 373

Name: MRI ID, dtype: int64

Group

False 373

Name: Group, dtype: int64

Visit

False 373

Name: Visit, dtype: int64

MR Delay

False 373

Name: MR Delay, dtype: int64

M/F

False 373

Name: M/F, dtype: int64

```
Hand
False      373
Name: Hand, dtype: int64
```

```
Age
False      373
Name: Age, dtype: int64
```

```
EDUC
False      373
Name: EDUC, dtype: int64
```

```
SES
False      373
Name: SES, dtype: int64
```

```
MMSE
False      373
Name: MMSE, dtype: int64
```

```
CDR
False      373
Name: CDR, dtype: int64
```

```
eTIV
False      373
Name: eTIV, dtype: int64
```

```
nWBV
False      373
Name: nWBV, dtype: int64
```

```
ASF
False      373
Name: ASF, dtype: int64
```

Let's see how many of each class is in our data set

```
[19]: data_frame['Subject ID'].value_counts()
```

```
[19]: OAS2_0048      5
      OAS2_0073      5
      OAS2_0070      5
      OAS2_0127      5
      OAS2_0036      4
      ..
```

```

OAS2_0124    2
OAS2_0146    2
OAS2_0063    2
OAS2_0138    2
OAS2_0042    2
Name: Subject ID, Length: 150, dtype: int64

```

```
[20]: # check the total number of subject
data_frame['Subject ID'].nunique()
```

```
[20]: 150
```

```
[21]: data_frame['Group'].value_counts()
```

```

[21]: Nondemented    190
      Demented      146
      Converted      37
      Name: Group, dtype: int64

```

Check for male and female

```
[22]: data_frame.groupby(['M/F', 'Group'])['Subject ID'].nunique()
```

```

[22]: M/F  Group
      F    Converted    10
           Demented    28
           Nondemented  50
      M    Converted     4
           Demented    36
           Nondemented  22
      Name: Subject ID, dtype: int64

```

Group by Male female and Subject ID

```
[23]: data_frame_sub =data_frame.groupby(['M/F', 'Group'])
```

```
[24]: data_frame_sub= data_frame_sub.agg({'Subject ID': 'nunique'})
```

```
[25]: data_frame_sub.head()
```

```

[25]:          Subject ID
      M/F Group
      F    Converted    10
           Demented    28
           Nondemented  50
      M    Converted     4
           Demented    36

```

```
[26]: # Change M to 1 and F to 0
data_frame['M/F'] = data_frame['M/F'].replace(['F','M'], [0,1])

[27]: # Hand cloumn value to numaric data R to 1 and L to 0
data_frame['Hand'] = data_frame['Hand'].replace(['L','R'], [0,1]) # Hand column

[28]: # Group :Replace Converted to Demented
data_frame['Group'] = data_frame['Group'].replace(['Converted'], ['Demented'])

[29]: data_frame.head()
```

```
[29]:
```

	Subject ID	MRI ID	Group	Visit	MR Delay	M/F	Hand	Age	\
0	OAS2_0001	OAS2_0001_MR1	Nondemented	1	0	1	1	87	
1	OAS2_0001	OAS2_0001_MR2	Nondemented	2	457	1	1	88	
2	OAS2_0002	OAS2_0002_MR1	Demented	1	0	1	1	75	
3	OAS2_0002	OAS2_0002_MR2	Demented	2	560	1	1	76	
4	OAS2_0002	OAS2_0002_MR3	Demented	3	1895	1	1	80	

	EDUC	SES	MMSE	CDR	eTIV	nWBV	ASF
0	14	2.0	27.0	0.0	1986.550000	0.696106	0.883440
1	14	2.0	30.0	0.0	2004.479526	0.681062	0.875539
2	12	2.0	23.0	0.5	1678.290000	0.736336	1.045710
3	12	2.0	28.0	0.5	1737.620000	0.713402	1.010000
4	12	2.0	22.0	0.5	1697.911134	0.701236	1.033623

```
[30]: data_frame.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 373 entries, 0 to 372
Data columns (total 15 columns):
#   Column          Non-Null Count  Dtype
---  -
0   Subject ID      373 non-null    object
1   MRI ID          373 non-null    object
2   Group           373 non-null    object
3   Visit           373 non-null    int64
4   MR Delay        373 non-null    int64
5   M/F             373 non-null    int64
6   Hand            373 non-null    int64
7   Age             373 non-null    int64
8   EDUC            373 non-null    int64
9   SES             373 non-null    float64
10  MMSE            373 non-null    float64
11  CDR             373 non-null    float64
12  eTIV            373 non-null    float64
13  nWBV            373 non-null    float64
14  ASF             373 non-null    float64
dtypes: float64(6), int64(6), object(3)
```

memory usage: 43.8+ KB

New dataframe for male female group with Unique Subject ID

```
[31]: data_frame_new = data_frame.groupby(['M/F', 'Group'])
```

```
[32]: data_frame_new = data_frame_new.agg({'Subject ID': 'nunique'})
```

```
[33]: data_frame_new.head()
```

```
[33]:
```

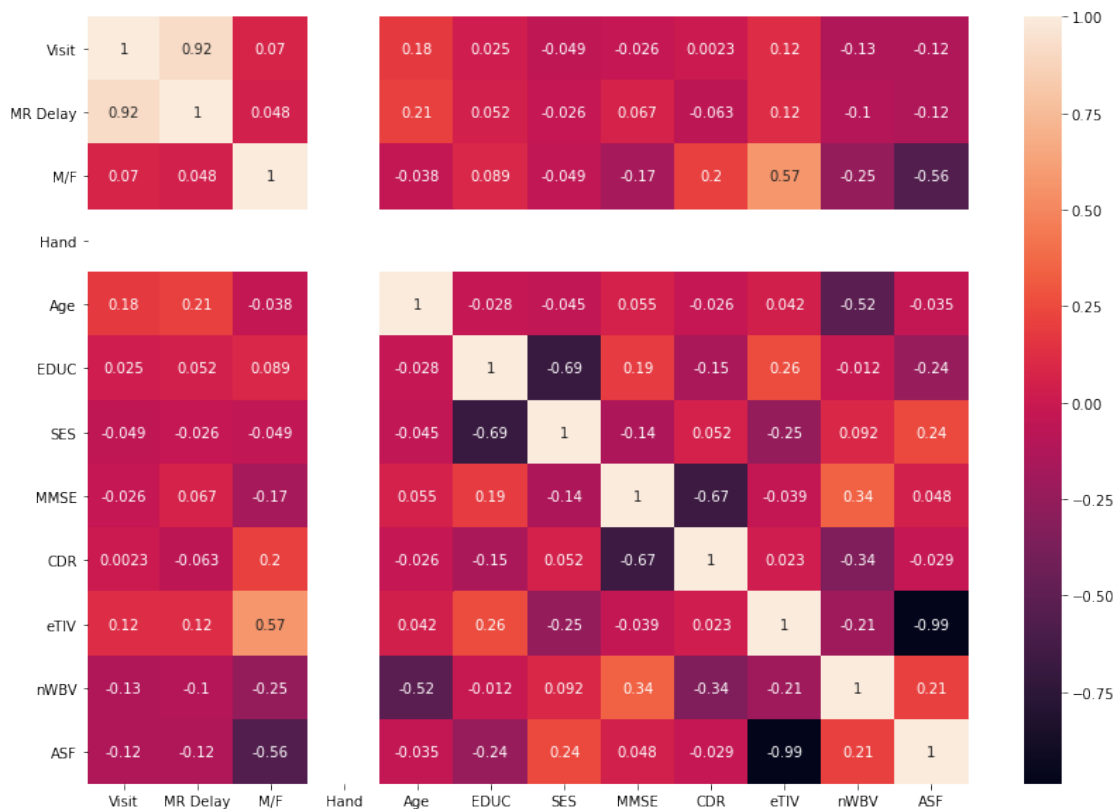
		Subject ID
M/F Group		
0	Demented	38
	Nondemented	50
1	Demented	40
	Nondemented	22

```
[34]: import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize = (14,10))

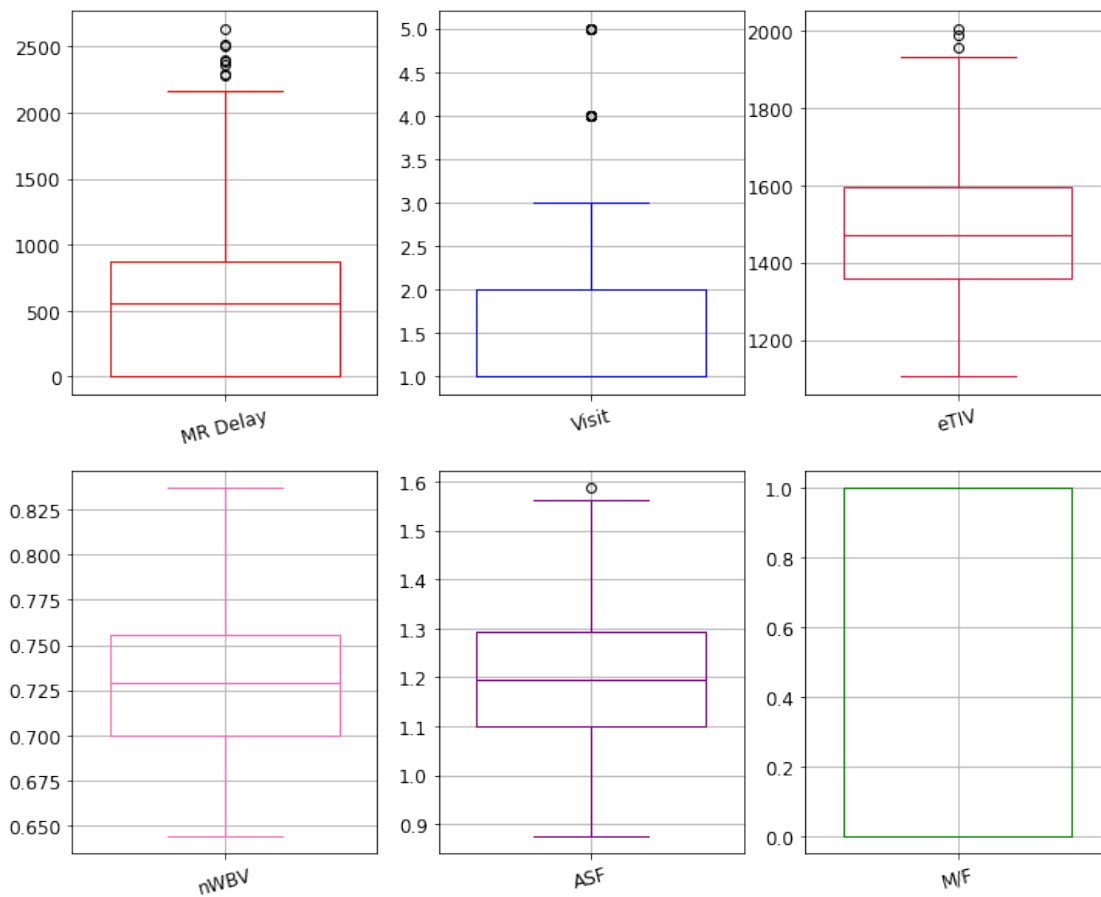
sns.heatmap(data_frame[data_frame.columns].corr(), annot=True)
```

```
[34]: <AxesSubplot:>
```



```
[35]: fig,axs = plt.subplots(2,3,figsize =(12,10))
boxplot_mrdelay = data_frame.boxplot(column=['MR Delay'], rot=15, widths=.
↪75,fontsize=12,color = 'red',ax=axs[0,0])
boxplot_visit = data_frame.boxplot(column=['Visit'], rot=15, fontsize=12,widths=.
↪75,color = 'blue',ax=axs[0,1])
boxplot_eTIV = data_frame.boxplot(column=['eTIV'], rot=15, fontsize=12,widths=.
↪75,color = 'crimson',ax=axs[0,2])
boxplot_nWBV = data_frame.boxplot(column=['nWBV'], rot=15, fontsize=12,widths=.
↪75,color = 'hotpink',ax=axs[1,0])
boxplot_ASF = data_frame.boxplot(column=['ASF'], rot=15, fontsize=12,widths=.
↪75,color = 'purple',ax=axs[1,1])
boxplot_MF = data_frame.boxplot(column=['M/F'], rot=15, fontsize=12,widths=.
↪75,color = 'green',ax=axs[1,2])

plt.show()
```



Target value, response variable or dependent variable

```
[36]: y_data = data_frame['Group']
```

```
[37]: y_data[0:5]
```

```
[37]: 0    Nondemented
      1    Nondemented
      2      Demented
      3      Demented
      4      Demented
      Name: Group, dtype: object
```

```
[38]: # Independent variable or regressor as X
      # drop unrelated values
      X_data = data_frame.drop(['Subject ID', 'MRI ID', 'Group'], axis=1)
      X_data.head()
```

```
[38]:   Visit  MR Delay  M/F  Hand  Age  EDUC  SES  MMSE  CDR      eTIV  \
0      1         0    1     1   87   14  2.0   27.0  0.0  1986.550000
1      2       457    1     1   88   14  2.0   30.0  0.0  2004.479526
2      1         0    1     1   75   12  2.0   23.0  0.5  1678.290000
3      2       560    1     1   76   12  2.0   28.0  0.5  1737.620000
4      3      1895    1     1   80   12  2.0   22.0  0.5  1697.911134

      nWBV      ASF
0  0.696106  0.883440
1  0.681062  0.875539
2  0.736336  1.045710
3  0.713402  1.010000
4  0.701236  1.033623
```

0.0.1 Train and Test Split

```
[39]: # import train_test_split library
      from sklearn.model_selection import train_test_split
      from sklearn.metrics import accuracy_score
      from sklearn.metrics import roc_auc_score
      from sklearn.metrics import roc_curve, auc

      X_train,X_test, y_train, y_test = train_test_split(X_data, y_data, test_size =0.33)
      print ('Train set:', X_train.shape, y_train.shape)
      print ('Test set:', X_test.shape, y_test.shape)
```

```
Train set: (249, 12) (249,)
```

```
Test set: (124, 12) (124,)
```

Normalize Data Data Standardization give data zero mean and unit variance

```
[40]: X_data= preprocessing.StandardScaler().fit(X_data).transform(X_data)
X_data[0:5]

[40]: array([[ -0.95706686, -0.93771494,  1.15379808,  0.          ,  1.30873772,
        -0.20813199, -0.3944662 , -0.09706416, -0.77765291,  2.83359462,
        -0.90181966, -2.26232493],
       [ 0.12799678, -0.21761337,  1.15379808,  0.          ,  1.43978716,
        -0.20813199, -0.3944662 ,  0.71958842, -0.77765291,  2.93552502,
        -1.30741435, -2.31961167],
       [-0.95706686, -0.93771494,  1.15379808,  0.          , -0.26385558,
        -0.90439416, -0.3944662 , -1.18593426,  0.55905002,  1.08111854,
         0.18280374, -1.08577527],
       [ 0.12799678, -0.05531476,  1.15379808,  0.          , -0.13280614,
        -0.90439416, -0.3944662 ,  0.17515337,  0.55905002,  1.41841305,
        -0.43550978, -1.34469305],
       [ 1.21306043,  2.04826424,  1.15379808,  0.          ,  0.39139163,
        -0.90439416, -0.3944662 , -1.45815179,  0.55905002,  1.19266583,
        -0.76351198, -1.17341289]])
```

0.0.2 Support Vector Machine (SVM)

```
[41]: # import SVM library

from sklearn import svm

[42]: svm_clf = svm.SVC(C=1, kernel='linear').fit(X_train, y_train)

[43]: yhat_svm = svm_clf.predict(X_test)

[44]: SVM_accuracy_score = accuracy_score(y_test,yhat_svm)*100
print("SVM_accuracy_score:", SVM_accuracy_score)

SVM_accuracy_score: 95.16129032258065
```

```
[45]: # Number of random trials
NUM_TRIALS = 10

# Arrays to store scores
svm_nested_scores = np.zeros(NUM_TRIALS)

# Set up possible values of parameters to optimize over
p_grid = {"C": [1, 10, 100], "gamma": [0.01, 0.1]}
```

```
[46]: # Loop for each trial
for i in range(NUM_TRIALS):

    # cross-validation techniques for the inner and outer loops,
    # independently of the dataset.
```



```
# E.g "GroupKFold", "LeaveOneOut", "LeaveOneGroupOut", etc.

inner_cv = KFold(n_splits=4, shuffle=True, random_state=i)
outer_cv = KFold(n_splits=4, shuffle=True, random_state=i)

# Nested CV with parameter optimization
Nes_svmclf = GridSearchCV(estimator=svm_clf, param_grid=p_grid, cv=inner_cv)

svm_nested_score = cross_val_score(svm_clf, X=X_data, y=y_data, cv=outer_cv)
```

```
[47]: svm_nested_score = svm_nested_score.mean()
      svm_nested_score
```

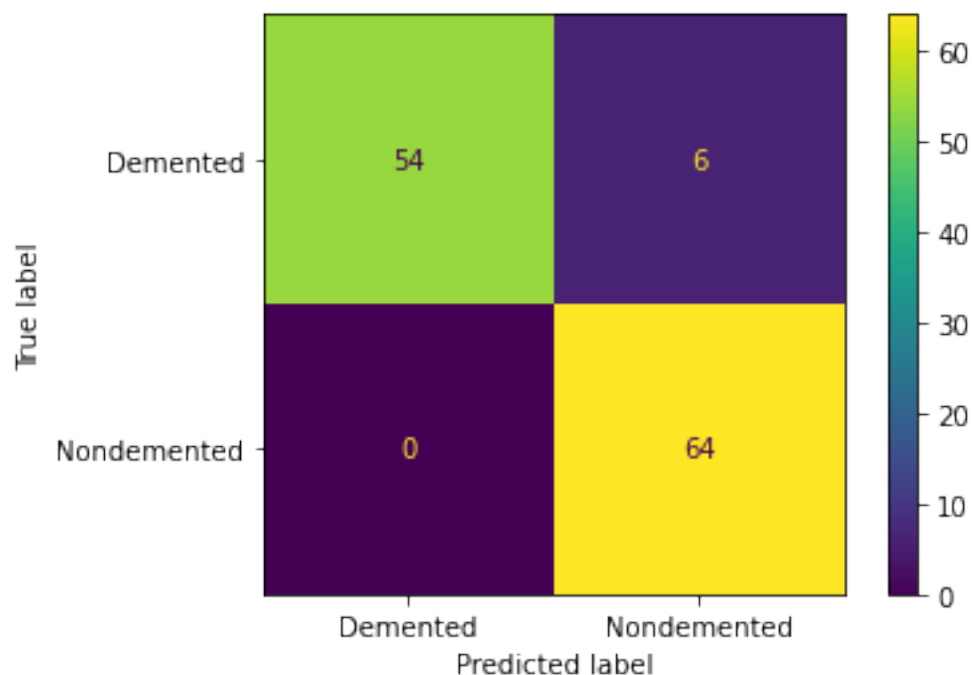
```
[47]: 0.9463509494394876
```

```
[48]: print('Accuracy: %.3f ' % (mean(svm_nested_score)))
```

Accuracy: 0.946

SVM Confusion matrix

```
[49]: #confusion_matrix(y_test,yhat_svm)
      from sklearn.metrics import confusion_matrix
      plot_confusion_matrix(svm_clf, X_test, y_test)
      plt.show()
```



```
[50]: # Confusion matrix using crosstab method of pandas.
svm_pd = pd.crosstab(y_test, yhat_svm, rownames=['True'],
    ↳ colnames=['Predicted'], margins=True)
svm_pd
```

```
[50]: Predicted    Demented  Nondemented  All
True
Demented          54           6    60
Nondemented        0          64    64
All                54          70   124
```

SVM Classification Report

```
[51]: #print(classification_report(y_test,yhat_svm))

print("confusion matrix\n",confusion_matrix(y_test,yhat_svm))
print("\nclassification report\n",classification_report(y_test,yhat_svm))
print("Accuracy: ",accuracy_score(y_test, yhat_svm)*100)
```

```
confusion matrix
[[54  6]
 [ 0 64]]
```

```
classification report
```

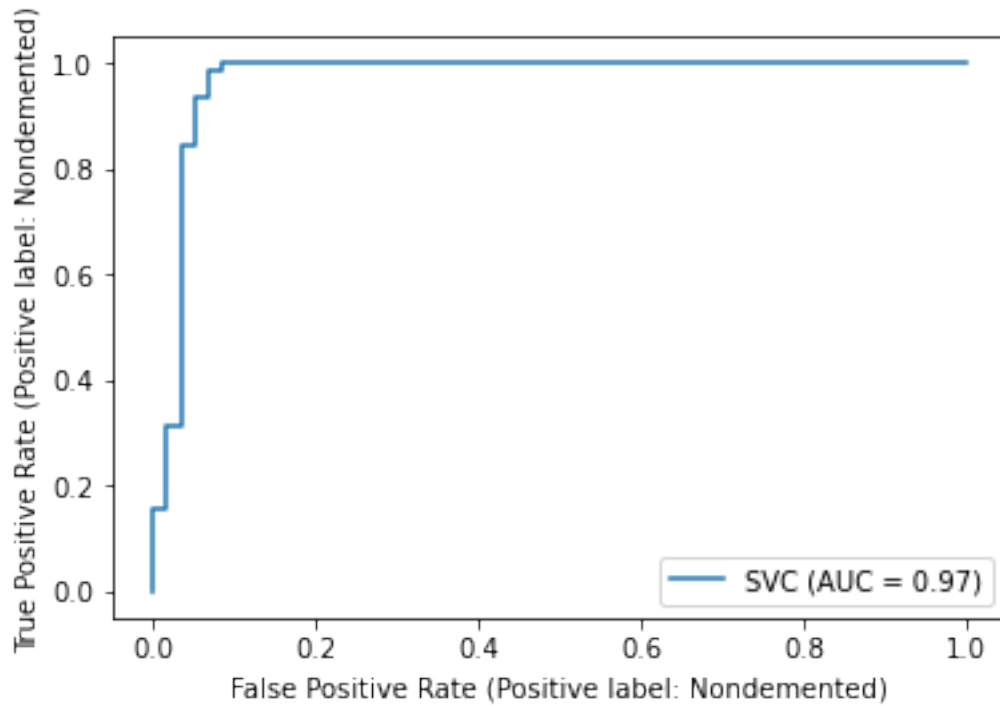
	precision	recall	f1-score	support
Demented	1.00	0.90	0.95	60
Nondemented	0.91	1.00	0.96	64
accuracy			0.95	124
macro avg	0.96	0.95	0.95	124
weighted avg	0.96	0.95	0.95	124

```
Accuracy: 95.16129032258065
```

```
[52]: SVM_roc_auc_score = roc_auc_score(y_test, svm_clf.decision_function(X_test),
    ↳ average=None)
print("SVM_roc_auc_score:", SVM_roc_auc_score)
```

```
SVM_roc_auc_score: 0.9705729166666667
```

```
[53]: metrics.plot_roc_curve(svm_clf, X_test, y_test)
plt.show()
```



0.0.3 Logistic Regression

```
[54]: #import Library LogisticRegression
      from sklearn.linear_model import LogisticRegression
```

```
[55]: #Train
      lr_clf = LogisticRegression(C=1, solver='liblinear').fit(X_train,y_train)
      lr_clf
```

```
[55]: LogisticRegression(C=1, solver='liblinear')
```

```
[56]: yhat_lr = lr_clf.predict(X_test)

      lr_yhat_prob = lr_clf.predict_proba(X_test)
```

```
[57]: LR_accuracy_score = accuracy_score(y_test,yhat_lr)*100
      print("LR_accuracy_score :",LR_accuracy_score )
```

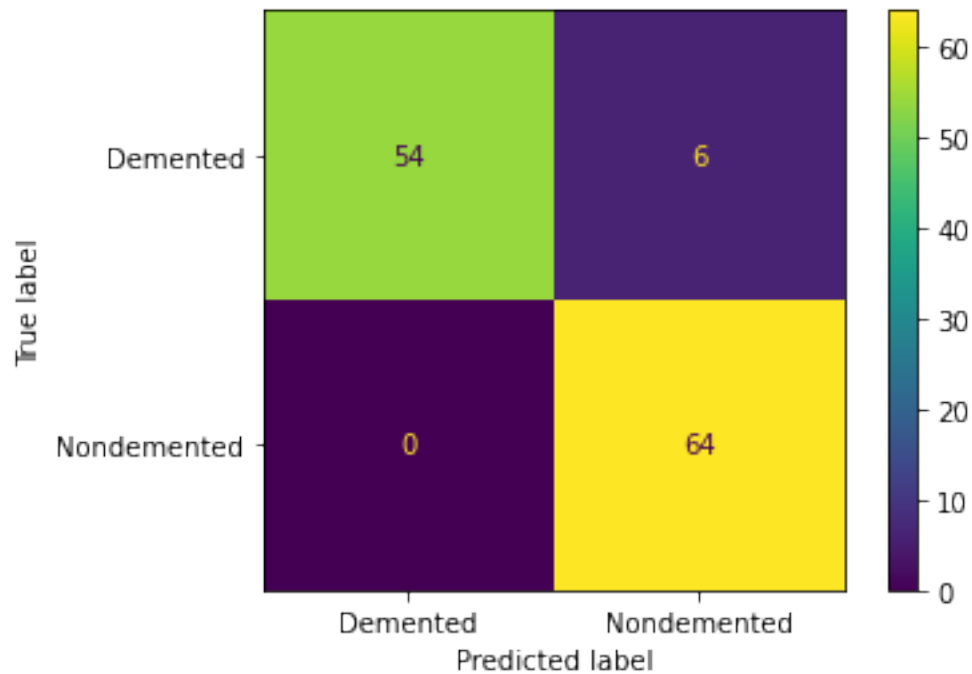
```
LR_accuracy_score : 95.16129032258065
```

LR Confusion matrix

```
[58]: #confusion_matrix(y_test,yhat_lr)

      plot_confusion_matrix(lr_clf, X_test, y_test)
```

```
plt.show()
```



```
[59]: # Confusion matrix using crosstab method of pandas.
LR_pd = pd.crosstab(y_test, yhat_lr, rownames=['True'], colnames=['Predicted'],
    ↪ margins=True)
LR_pd
```

```
[59]: Predicted    Demented  Nondemented  All
True
Demented         54           6      60
Nondemented        0          64      64
All              54          70     124
```

LR Classification Report

```
[60]: #print(classification_report(y_test,yhat_lr))

print("confusion matrix\n",confusion_matrix(y_test,yhat_lr))
print("\nclassification report\n",classification_report(y_test,yhat_lr))
print("Accuracy: ",accuracy_score(y_test, yhat_lr)*100)
```

```
confusion matrix
[[54  6]
 [ 0 64]]
```

```
classification report
```

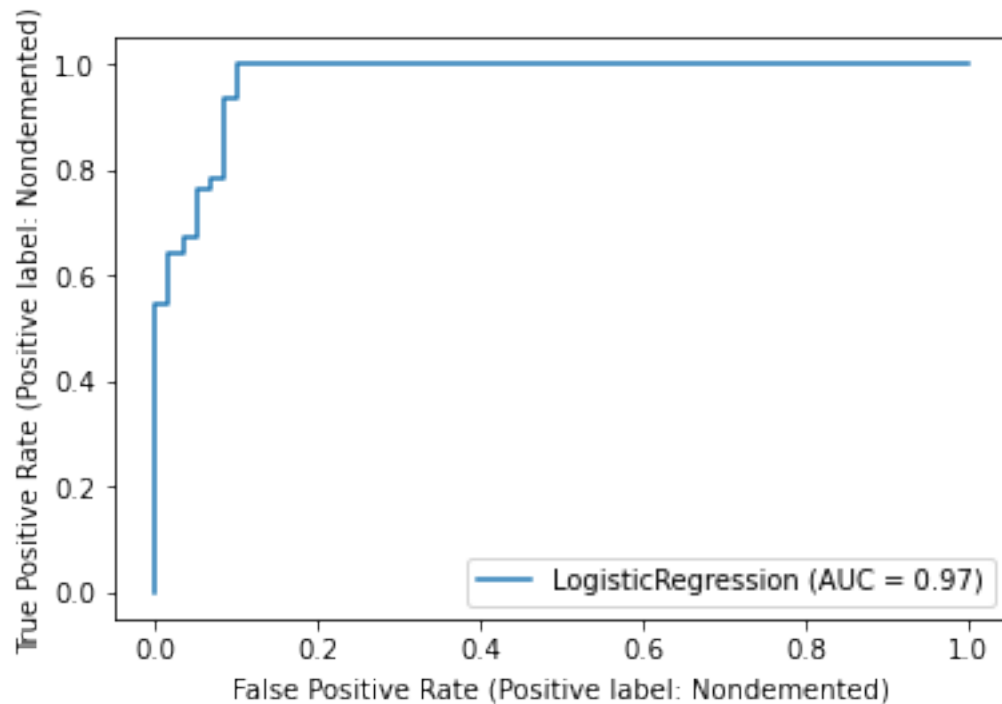
	precision	recall	f1-score	support
Demented	1.00	0.90	0.95	60
Nondemented	0.91	1.00	0.96	64
accuracy			0.95	124
macro avg	0.96	0.95	0.95	124
weighted avg	0.96	0.95	0.95	124

Accuracy: 95.16129032258065

```
[61]: LR_roc_auc_score= roc_auc_score(y_test, lr_clf.decision_function(X_test),
    ↪average=None)
    print("LR_roc_auc_score :",LR_roc_auc_score )
```

LR_roc_auc_score : 0.9723958333333333

```
[62]: metrics.plot_roc_curve(lr_clf, X_test, y_test)
    plt.show()
```



```
[ ]:
```

```
[63]: NUM_TRIALS = 10
```

```

lr_nested_scores = np.zeros(NUM_TRIALS)

p_grid = {"C": [1, 10, 100], "gamma": [0.01, 0.1]}

# Loop for each trial
for i in range(NUM_TRIALS):

    #cross-validation techniques for the inner and outer loops,
    # independently of the dataset.
    # E.g "GroupKFold", "LeaveOneOut", "LeaveOneGroupOut", etc.

    inner_cv = KFold(n_splits=4, shuffle=True, random_state=i)
    outer_cv = KFold(n_splits=4, shuffle=True, random_state=i)

    # Nested CV with parameter optimization
    Nes_lrclf = GridSearchCV(estimator=lr_clf, param_grid=p_grid, cv=inner_cv)

    lr_nested_score = cross_val_score(lr_clf, X=X_data, y=y_data, cv=outer_cv)

```

```

[64]: lr_nested_score = lr_nested_score.mean()
      lr_nested_score

```

```

[64]: 0.9463509494394876

```

```

[65]: print('Accuracy: %.3f ' % (mean(lr_nested_score)))

```

```

Accuracy: 0.946

```

0.0.4 Random Forest Classifier

```

[66]: from sklearn.ensemble import RandomForestClassifier

```

```

rf_clf = RandomForestClassifier().fit(X_train, y_train)
rf_pred = rf_clf.predict(X_test)
rf_yhat_prob = rf_clf.predict_proba(X_test)

```

```

[67]: from sklearn.metrics import classification_report, confusion_matrix, \
      ↪ accuracy_score

print("confusion matrix\n",confusion_matrix(y_test,rf_pred))
print("\nclassification report\n",classification_report(y_test,rf_pred))
print("Accuracy: ",accuracy_score(y_test, rf_pred)*100)

```

```
confusion matrix
[[54  6]
 [ 0 64]]
```

```
classification report
              precision    recall  f1-score   support

   Demented         1.00      0.90      0.95         60
  Nondemented        0.91      1.00      0.96         64

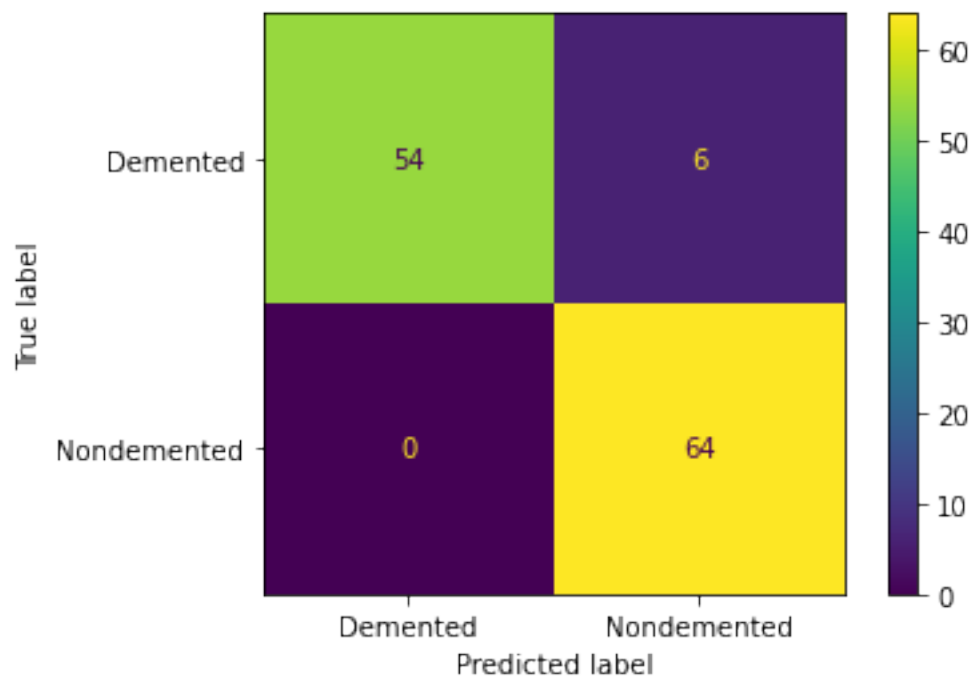
   accuracy              0.95         124
  macro avg              0.96      0.95      0.95         124
 weighted avg              0.96      0.95      0.95         124
```

Accuracy: 95.16129032258065

```
[68]: RF_accuracy_score = accuracy_score(y_test,rf_pred)*100
      print("RF_accuracy_score :",RF_accuracy_score )
```

RF_accuracy_score : 95.16129032258065

```
[69]: plot_confusion_matrix(rf_clf, X_test, y_test)
      plt.show()
```



```
[70]: # Confusion matrix using crosstab method of pandas.
RF_pd = pd.crosstab(y_test, rf_pred, rownames=['True'], colnames=['Predicted'],
    ↪margins=True)
RF_pd
```

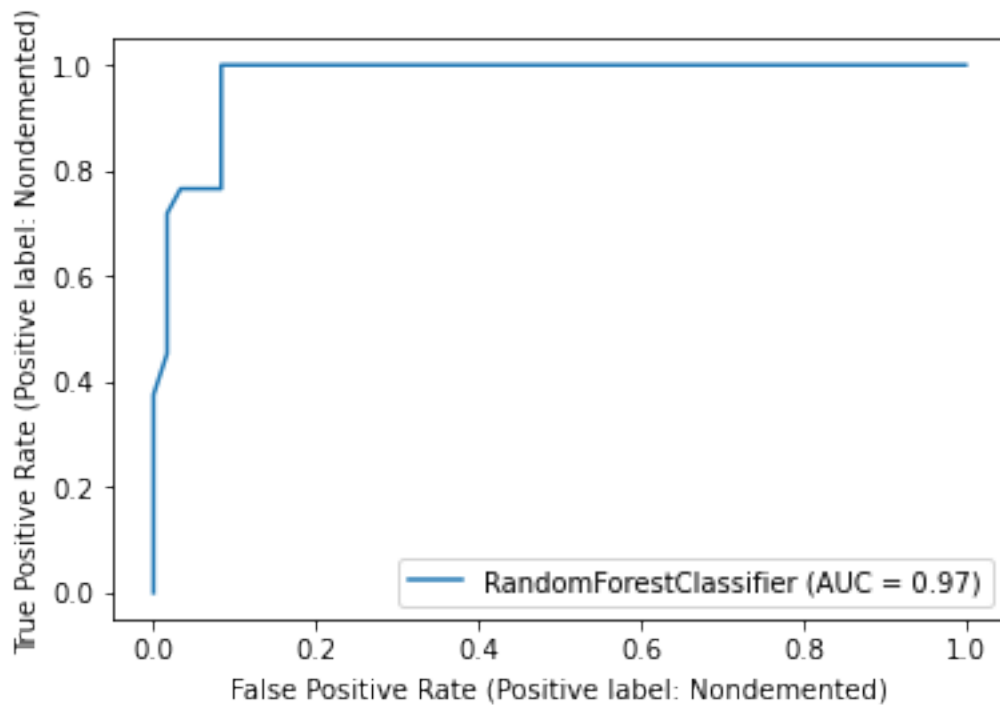
```
[70]: Predicted    Demented  Nondemented  All
True
Demented          54           6      60
Nondemented        0          64      64
All                54          70     124
```

```
[71]: RF_roc_auc_score= roc_auc_score(y_test, rf_clf.predict_proba(X_test)[: ,1],
    ↪average=None)
print("RF roc_auc score :",RF_roc_auc_score )
```

RF roc_auc score : 0.97421875

```
[72]: metrics.plot_roc_curve(rf_clf, X_test, y_test)

plt.show()
```



```
[73]: NUM_TRIALS = 10
```



```

rf_nested_scores = np.zeros(NUM_TRIALS)

p_grid = {"C": [1, 10, 100], "gamma": [0.01, 0.1]}

# Loop for each trial
for i in range(NUM_TRIALS):

    # Choose cross-validation techniques for the inner and outer loops,
    # independently of the dataset.
    # E.g "GroupKFold", "LeaveOneOut", "LeaveOneGroupOut", etc.

    inner_cv = KFold(n_splits=4, shuffle=True, random_state=i)
    outer_cv = KFold(n_splits=4, shuffle=True, random_state=i)

    # Nested CV with parameter optimization
    Nes_rfclf = GridSearchCV(estimator=rf_clf, param_grid=p_grid, cv=inner_cv)

    rf_nested_score = cross_val_score(rf_clf, X=X_data, y=y_data, cv=outer_cv)

```

```

[74]: rf_nested_score = rf_nested_score.mean()
      rf_nested_score

```

```

[74]: 0.9436627773964769

```

```

[75]: print('Accuracy: %.3f ' % (mean(rf_nested_score)))

```

```

Accuracy: 0.944

```

0.0.5 Naive Bayes

```

[76]: from sklearn.naive_bayes import GaussianNB

```

```

[77]: gnb_clf = GaussianNB()

```

```

[78]: y_pred_gnb = gnb_clf.fit(X_train, y_train).predict(X_test)

```

```

[79]: #yhat_gnb = gnb.predict(X_test)

      gnb_yhat_prob = gnb_clf.predict_proba(X_test)

```

```

[80]: gnb_Accuracy_score = metrics.accuracy_score(y_test, y_pred_gnb)*100
      gnb_Accuracy_score

```

```

[80]: 95.16129032258065

```

```
[81]: print("confusion matrix\n",confusion_matrix(y_test,y_pred_gnb))
print("\nclassification report\n",classification_report(y_test,y_pred_gnb))
print("Accuracy: ",accuracy_score(y_test, y_pred_gnb)*100)
```

confusion matrix

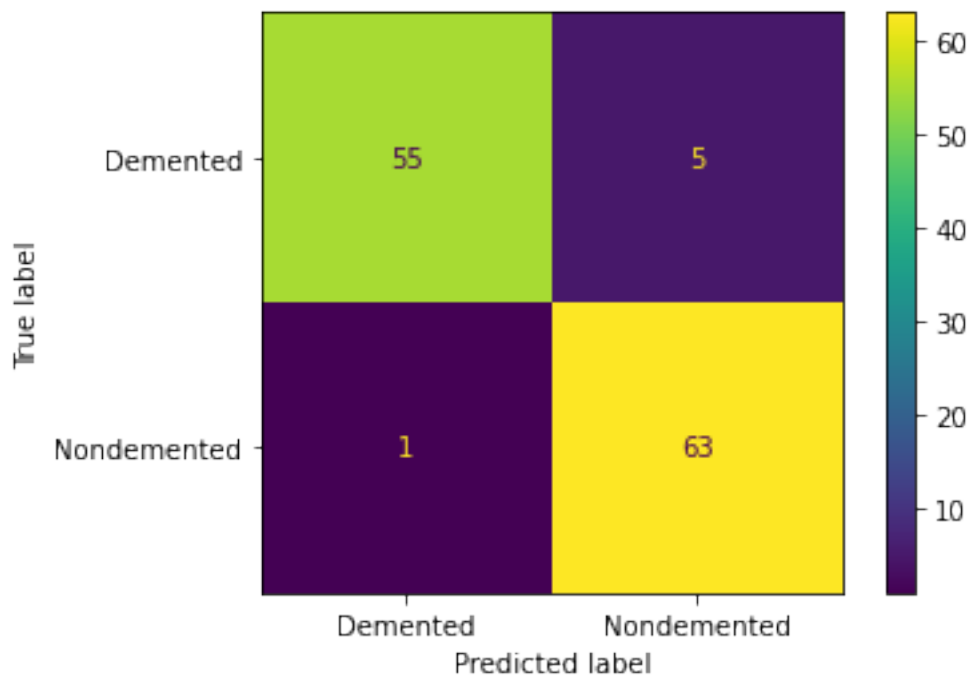
```
[[55  5]
 [ 1 63]]
```

classification report

	precision	recall	f1-score	support
Demented	0.98	0.92	0.95	60
Nondemented	0.93	0.98	0.95	64
accuracy			0.95	124
macro avg	0.95	0.95	0.95	124
weighted avg	0.95	0.95	0.95	124

Accuracy: 95.16129032258065

```
[82]: plot_confusion_matrix(gnb_clf, X_test, y_test)
plt.show()
```

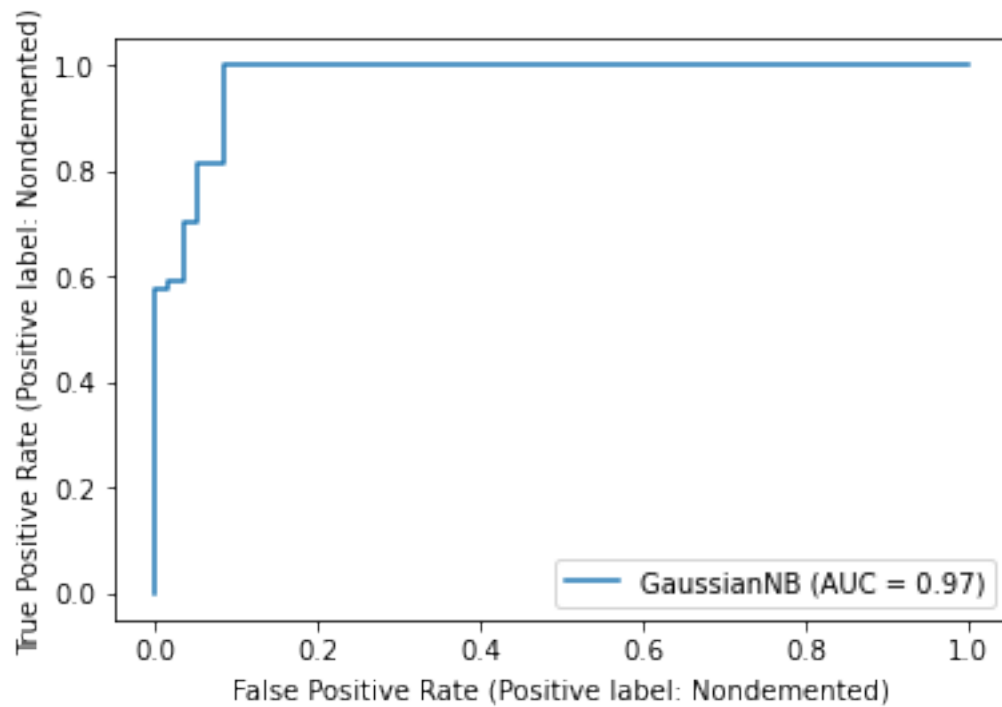


```
[83]: # Confusion matrix using crosstab method of pandas.
```

```
GNV_pd = pd.crosstab(y_test, y_pred_gnb, rownames=['True'],  
    ↳ colnames=['Predicted'], margins=True)  
GNV_pd
```

```
[83]: Predicted    Demented  Nondemented  All  
True  
Demented          55          5    60  
Nondemented        1         63    64  
All                56         68   124
```

```
[84]: metrics.plot_roc_curve(gnb_clf, X_test, y_test)  
  
plt.show()
```



```
[85]: gnb_roc_auc_score= roc_auc_score(y_test, gnb_clf.predict_proba(X_test)[:,-1],  
    ↳ average=None)  
print("GNB roc_auc score :", gnb_roc_auc_score )
```

GNB roc_auc score : 0.975

```
[86]: NUM_TRIALS = 10  
  
gnb_nested_scores = np.zeros(NUM_TRIALS)
```

```

p_grid = {"C": [1, 10, 100], "gamma": [0.01, 0.1]}

# Loop for each trial
for i in range(NUM_TRIALS):

    # Choose cross-validation techniques for the inner and outer loops,
    # independently of the dataset.
    # E.g "GroupKFold", "LeaveOneOut", "LeaveOneGroupOut", etc.

    inner_cv = KFold(n_splits=4, shuffle=True, random_state=i)
    outer_cv = KFold(n_splits=4, shuffle=True, random_state=i)

    # Nested CV with parameter optimization
    Nes_gnbclf = GridSearchCV(estimator=gnb_clf, param_grid=p_grid, cv=inner_cv)

    gnb_nested_score = cross_val_score(gnb_clf, X=X_data, y=y_data, cv=outer_cv)

```

```

[87]: gnb_nested_scores = gnb_nested_scores.mean()
      gnb_nested_scores

```

```

[87]: 0.0

```

```

[88]: print('Accuracy: %.3f ' % (mean(gnb_nested_scores)))

```

```

Accuracy: 0.000

```

0.0.6 Ada Boosting

```

[89]: from sklearn.ensemble import AdaBoostClassifier

      ada_clf = AdaBoostClassifier(n_estimators=100, random_state=42)
      ada_clf.fit(X_train, y_train)

```

```

[89]: AdaBoostClassifier(n_estimators=100, random_state=42)

```

```

[90]: yhat_ada = ada_clf.predict(X_test)

```

```

#yhat_ada

```

```

[91]: ada_yhat_prob = ada_clf.predict_proba(X_test)
      #ada_yhat_prob

```

```

[92]: ada_accuracy_score = accuracy_score(y_test, yhat_ada)*100
      print("ada_accuracy_score :", ada_accuracy_score )

```

ada_accuracy_score : 92.74193548387096

```
[93]: from sklearn.metrics import classification_report, confusion_matrix, \
      ↪ accuracy_score

print("confusion matrix\n",confusion_matrix(y_test,yhat_ada))
print("\nclassification report\n",classification_report(y_test,yhat_ada))
print("Accuracy: ",accuracy_score(y_test, yhat_ada)*100)
```

confusion matrix

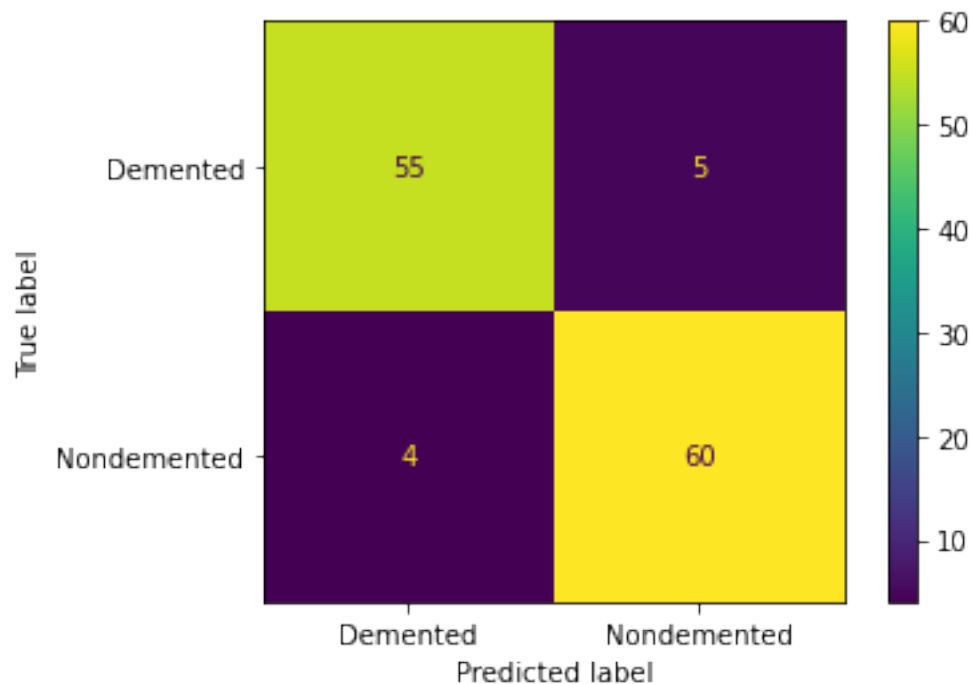
```
[[55  5]
 [ 4 60]]
```

classification report

	precision	recall	f1-score	support
Demented	0.93	0.92	0.92	60
Nondemented	0.92	0.94	0.93	64
accuracy			0.93	124
macro avg	0.93	0.93	0.93	124
weighted avg	0.93	0.93	0.93	124

Accuracy: 92.74193548387096

```
[94]: plot_confusion_matrix(ada_clf, X_test, y_test)
      plt.show()
```

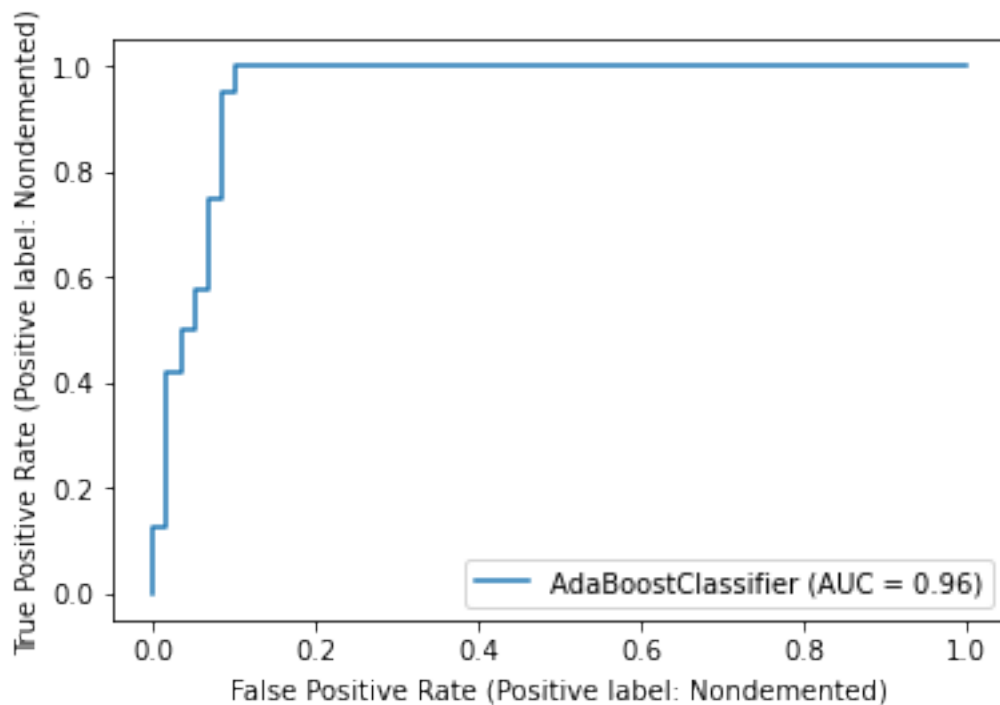


```
[95]: # Confusion matrix using crosstab method of pandas.
ada_pd = pd.crosstab(y_test, yhat_ada, rownames=['True'],
    ↳ colnames=['Predicted'], margins=True)
ada_pd
```

```
[95]: Predicted    Demented  Nondemented  All
True
Demented          55           5     60
Nondemented        4          60     64
All                59          65    124
```

```
[96]: metrics.plot_roc_curve(ada_clf, X_test, y_test)

plt.show()
```



```
[97]: ada_roc_auc_score= roc_auc_score(y_test, ada_clf.decision_function(X_test),
    ↳ average=None)
print("ada_roc_auc_score :",ada_roc_auc_score )
```

```
ada_roc_auc_score : 0.95546875
```

```
[98]: NUM_TRIALS = 10

ada_nested_scores = np.zeros(NUM_TRIALS)

p_grid = {"C": [1, 10, 100], "gamma": [0.01, 0.1]}

# Loop for each trial
for i in range(NUM_TRIALS):

    # cross-validation techniques for the inner and outer loops,
    # independently of the dataset.
    # E.g "GroupKFold", "LeaveOneOut", "LeaveOneGroupOut", etc.

    inner_cv = KFold(n_splits=4, shuffle=True, random_state=i)
    outer_cv = KFold(n_splits=4, shuffle=True, random_state=i)

    # Nested CV with parameter optimization
    Nes_adacclf = GridSearchCV(estimator=ada_clf, param_grid=p_grid, cv=inner_cv)

    ada_nested_score = cross_val_score(ada_clf, X=X_data, y=y_data, cv=outer_cv)
```

```
[99]: ada_nested_score = ada_nested_score.mean()
ada_nested_score
```

```
[99]: 0.9302505147563487
```

```
[100]: print('Accuracy: %.3f ' % (mean(ada_nested_score)))
```

```
Accuracy: 0.930
```

0.0.7 Gradient boosting

```
[101]: from sklearn.ensemble import GradientBoostingClassifier
```

```
[102]: Gra_clf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,
max_depth=1, random_state=0).fit(X_train,
↪y_train)
```

```
[103]: Gra_clf.score(X_test, y_test)
```

```
[103]: 0.9193548387096774
```

```
[104]: yhat_Gra_cl = Gra_clf.predict(X_test)
```

```
Gra_yhat_prob = Gra_clf.predict_proba(X_test)
```

```
[105]: gra_accuracy_score = Gra_clf.score(X_test, y_test)*100  
print("gra_accuracy_score :", gra_accuracy_score )
```

```
gra_accuracy_score : 91.93548387096774
```

```
[106]: print("confusion matrix\n",confusion_matrix(y_test,yhat_Gra_cl))  
print("\nclassification report\n",classification_report(y_test,yhat_Gra_cl))  
print("Accuracy: ",accuracy_score(y_test, yhat_Gra_cl)*100)
```

```
confusion matrix
```

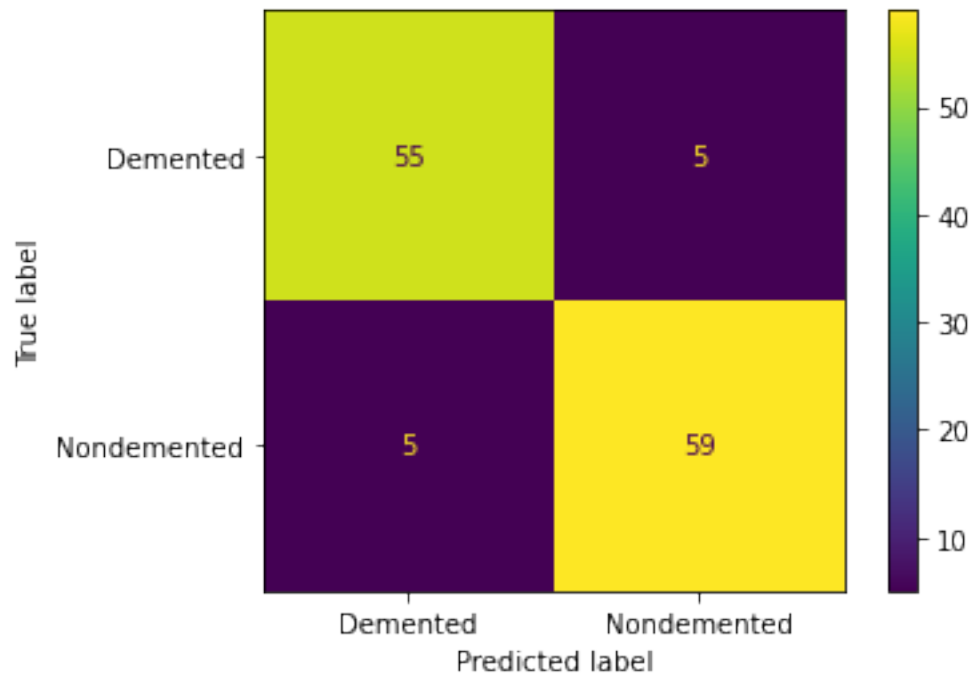
```
[[55  5]  
 [ 5 59]]
```

```
classification report
```

	precision	recall	f1-score	support
Demented	0.92	0.92	0.92	60
Nondemented	0.92	0.92	0.92	64
accuracy			0.92	124
macro avg	0.92	0.92	0.92	124
weighted avg	0.92	0.92	0.92	124

```
Accuracy: 91.93548387096774
```

```
[107]: plot_confusion_matrix(Gra_clf, X_test, y_test)  
plt.show()
```

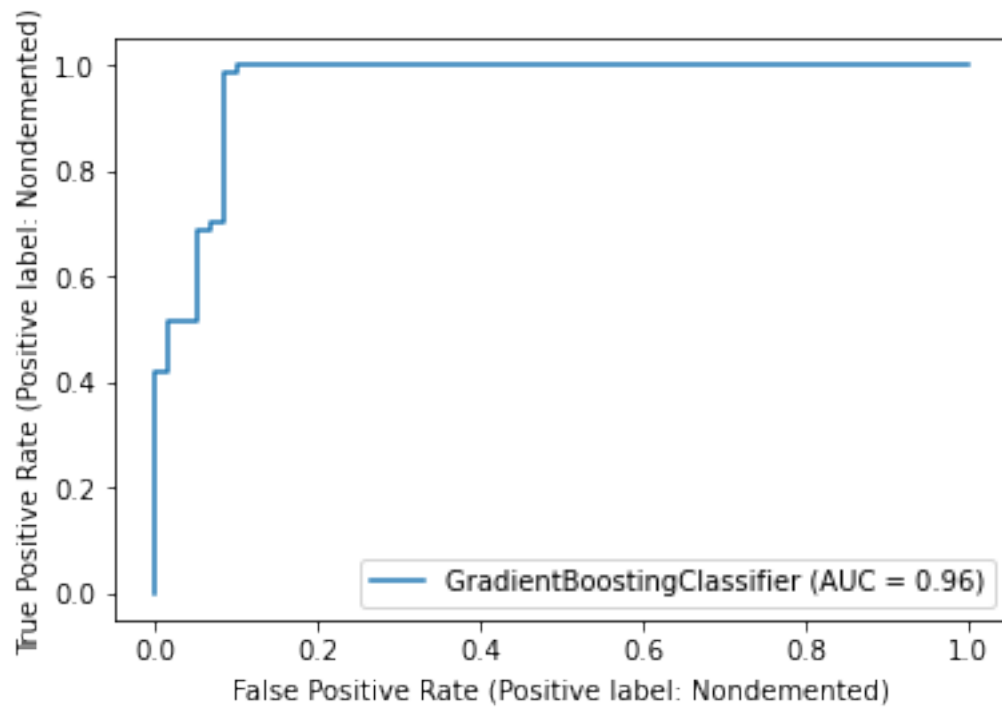



```
[108]: # Confusion matrix using crosstab method of pandas.
Gra_pd = pd.crosstab(y_test, yhat_Gra_cl, rownames=['True'],
                    ↳ colnames=['Predicted'], margins=True)
Gra_pd
```

```
[108]: Predicted    Demented  Nondemented  All
True
Demented          55           5    60
Nondemented        5          59    64
All                60          64   124
```

```
[109]: metrics.plot_roc_curve(Gra_clf, X_test, y_test)

plt.show()
```



```
[110]: gra_roc_auc_score= roc_auc_score(y_test, Gra_clf.decision_function(X_test),
    ↪average=None)
print("gra_roc_auc_score :",gra_roc_auc_score )
```

```
gra_roc_auc_score : 0.9638020833333334
```

```
[111]: NUM_TRIALS = 10

gra_nested_scores = np.zeros(NUM_TRIALS)

p_grid = {"C": [1, 10, 100], "gamma": [0.01, 0.1]}

for i in range(NUM_TRIALS):

    inner_cv = KFold(n_splits=4, shuffle=True, random_state=i)
    outer_cv = KFold(n_splits=4, shuffle=True, random_state=i)
```

```

# Nested CV with parameter optimization
Nes_graclf = GridSearchCV(estimator=Gra_clf, param_grid=p_grid, cv=inner_cv)

gra_nested_scores = cross_val_score(Gra_clf, X=X_data, y=y_data,
→cv=outer_cv)

```

```

[112]: gra_nested_scores = gra_nested_scores.mean()
       gra_nested_scores

```

```

[112]: 0.9355982612674445

```

```

[113]: print('Accuracy: %.3f ' % (mean(gra_nested_scores)))

```

```

Accuracy: 0.936

```

0.0.8 Results

Accuracy score report dataframe

```

[114]: # create a accuracy list
       accuracy_score = [
→SVM_accuracy_score,LR_accuracy_score,RF_accuracy_score,gnb_Accuracy_score,ada_accuracy_score

# create a AUC-score list
       auc_score = [
→[SVM_roc_auc_score,LR_roc_auc_score,RF_roc_auc_score,gnb_roc_auc_score,ada_roc_auc_score,gr

# create a nested-score list
       Nested_score = [
→[svm_nested_score,lr_nested_score,RF_roc_auc_score,gnb_nested_scores,ada_nested_score,gra_n

# fomulate the report format
       df_report = pd.DataFrame(accuracy_score, index=['SVM','Logistic',
→Regression','Random Forest','Naive Bayes','Ada Boosting','Gradient',
→boosting'])
       df_report.columns = ['Accuracy Score']
       df_report.insert(loc=1, column='ROC AUC Score', value= auc_score)
       df_report.insert(loc=2, column='Nested Score (mean)', value= Nested_score)
       df_report.columns.name = 'Algorithm'
       df_report

```

```

[114]: Algorithm          Accuracy Score  ROC AUC Score  Nested Score (mean)
       SVM                95.161290        0.970573        0.946351
       Logistic Regression 95.161290        0.972396        0.946351
       Random Forest      95.161290        0.974219        0.974219
       Naive Bayes         95.161290        0.975000        0.000000

```

Ada Boosting	92.741935	0.955469	0.930251
Gradient boosting	91.935484	0.963802	0.935598

```
[115]: fig = metrics.plot_roc_curve(rf_clf, X_test, y_test)
fig = metrics.plot_roc_curve(Gra_clf, X_test, y_test, ax = fig.ax_)
fig = metrics.plot_roc_curve(ada_clf, X_test, y_test, ax = fig.ax_)
fig = metrics.plot_roc_curve(gnb_clf, X_test, y_test, ax = fig.ax_)
#metrics.plot_roc_curve(rf_regressor, X_test, y_test)
fig = metrics.plot_roc_curve(lr_clf, X_test, y_test, ax = fig.ax_)
fig = metrics.plot_roc_curve(svm_clf, X_test, y_test, ax = fig.ax_)

fig.figure_.suptitle("ROC curve comparison")
fig.figure_.set_size_inches(14,10)
plt.show()
```

ROC curve comparison

