```
from google.colab import files
uploaded=files.upload()
```

```
Choose Files seeds.csv
```

- **seeds.csv**(text/csv) - 9498 bytes, last modified: 7/7/2025 - 100% done

```
Saving seeds.csv to seeds (1).csv
```

```
import pandas
import numpy
import matplotlib.pyplot as plt
```

```
from sklearn import model_selection
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
```

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

```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
```

```
dataset = pandas.read_csv("/content/seeds.csv")
print(dataset.head())
```

```
    15.26  14.84   0.871  5.763  3.312  2.221   5.22  1
0  14.88  14.57  0.8811  5.554  3.333  1.018  4.956  1
1  14.29  14.09  0.9050  5.291  3.337  2.699  4.825  1
2  13.84  13.94  0.8955  5.324  3.379  2.259  4.805  1
3  16.14  14.99  0.9034  5.658  3.562  1.355  5.175  1
4  14.38  14.21  0.8951  5.386  3.312  2.462  4.956  1
```

```
# load the data assign names
url="seeds.csv"
names=['area','parameter','compact','length','width','asymmetry','groove','class']
dataset=pandas.read_csv(url,names=names)
#Data
print(dataset)
```

```
       area  parameter  compact  length  width  asymmetry  groove  class
0     15.26      14.84   0.8710   5.763  3.312      2.221   5.220      1
1     14.88      14.57   0.8811   5.554  3.333      1.018   4.956      1
2     14.29      14.09   0.9050   5.291  3.337      2.699   4.825      1
3     13.84      13.94   0.8955   5.324  3.379      2.259   4.805      1
4     16.14      14.99   0.9034   5.658  3.562      1.355   5.175      1
..      ...        ...      ...     ...    ...        ...     ...    ...
205  12.19      13.20   0.8783   5.137  2.981      3.631   4.870      3
206  11.23      12.88   0.8511   5.140  2.795      4.325   5.003      3
207  13.20      13.66   0.8883   5.236  3.232      8.315   5.056      3
208  11.84      13.21   0.8521   5.175  2.836      3.598   5.044      3
209  12.30      13.34   0.8684   5.243  2.974      5.637   5.063      3

[210 rows x 8 columns]
```

```
# drescriptive statsfor data
print(dataset.dtypes)
```

```
area         float64
parameter    float64
compact      float64
```

```
length       float64
width        float64
asymmetry    float64
groove       float64
class          int64
dtype: object
```

```
print("data shepe",dataset.shape)
print("head of dataset",dataset.head(3))
print("tail of dataset",dataset.tail(3))
```

```
data shepe (210, 8)
head of dataset     area  parameter  compact  length  width  asymmetry  groove  class
0  15.26      14.84   0.8710   5.763  3.312      2.221   5.220      1
1  14.88      14.57   0.8811   5.554  3.333      1.018   4.956      1
2  14.29      14.09   0.9050   5.291  3.337      2.699   4.825      1
tail of dataset       area  parameter  compact  length  width  asymmetry  groove  class
207  13.20      13.66   0.8883   5.236  3.232      8.315   5.056      3
208  11.84      13.21   0.8521   5.175  2.836      3.598   5.044      3
209  12.30      13.34   0.8684   5.243  2.974      5.637   5.063      3
```

```
dataset.apply(lambda x:sum(x.isnull()),axis=0)
```

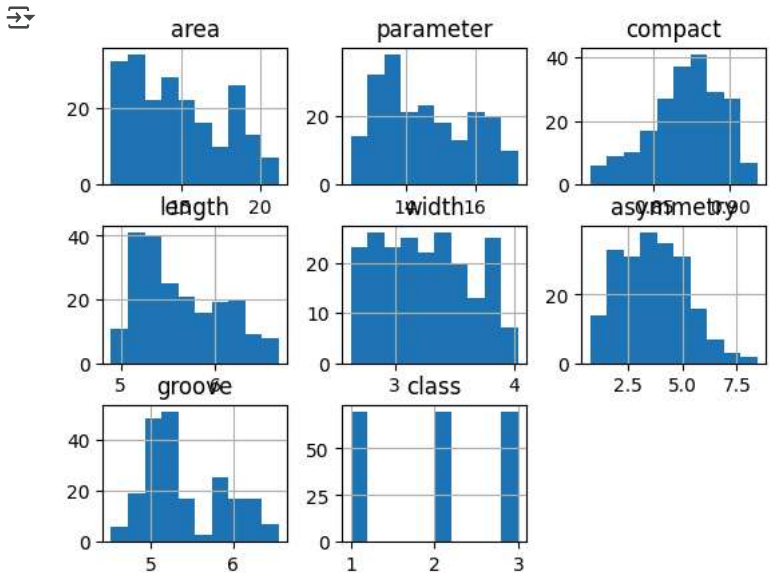
	0
area	0
parameter	0
compact	0
length	0
width	0
asymmetry	0
groove	0
class	0

dtype: int64

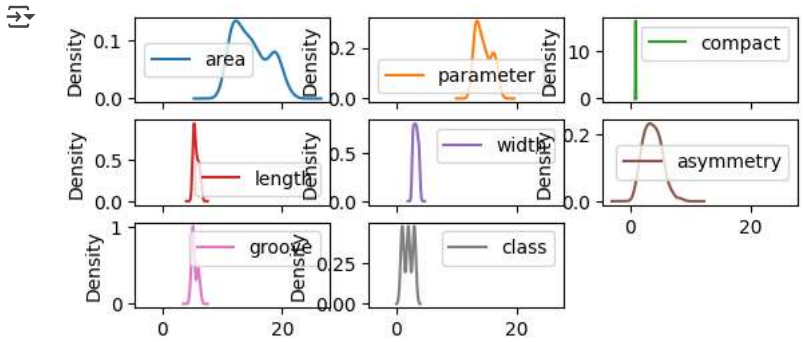
```
pandas.set_option('display.precision',2)
print(dataset.describe)
```

```
<bound method NDFrame.describe of       area  parameter  compact  length  width  asymmetry  groove  class
0    15.26      14.84     0.87    5.76   3.31       2.22    5.22      1
1    14.88      14.57     0.88    5.55   3.33       1.02    4.96      1
2    14.29      14.09     0.91    5.29   3.34       2.70    4.83      1
3    13.84      13.94     0.90    5.32   3.38       2.26    4.80      1
4    16.14      14.99     0.90    5.66   3.56       1.35    5.17      1
..     ...        ...      ...     ...    ...        ...     ...    ...
205  12.19      13.20     0.88    5.14   2.98       3.63    4.87      3
206  11.23      12.88     0.85    5.14   2.79       4.33    5.00      3
207  13.20      13.66     0.89    5.24   3.23       8.31    5.06      3
208  11.84      13.21     0.85    5.17   2.84       3.60    5.04      3
209  12.30      13.34     0.87    5.24   2.97       5.64    5.06      3

[210 rows x 8 columns]>
```

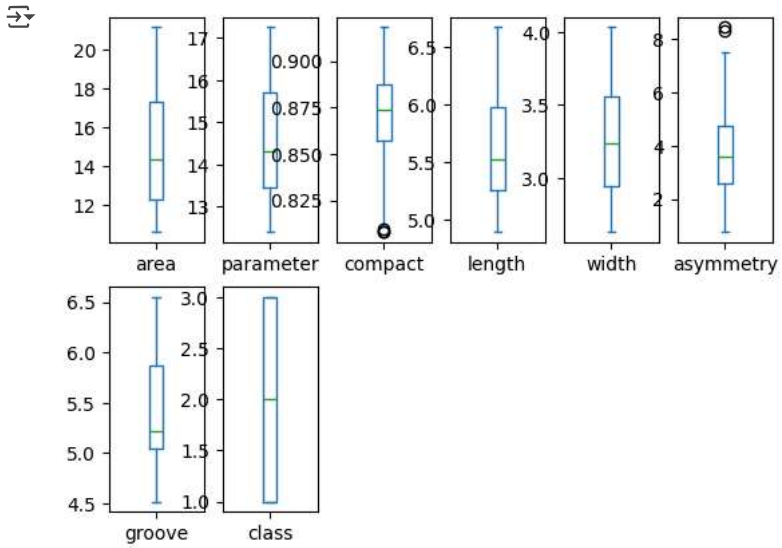
```
dataset.hist( )
plt.show()
```



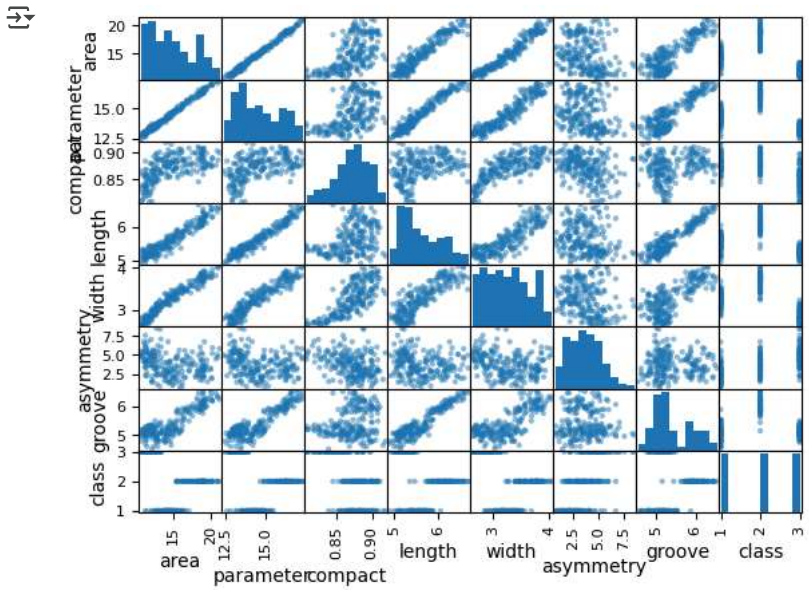

```
dataset.plot(kind='density',subplots=True,layout=(5,3))
plt.show()
```



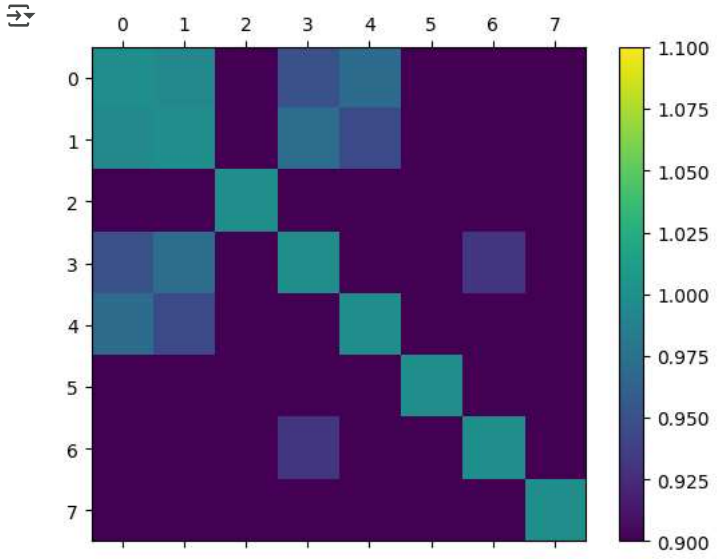

```
#to find outliers in dataset's attributes
dataset.plot(kind='box',subplots=True,layout=(2,6))
plt.show()
```




```
scatter_matrix(dataset)
plt.show()
```




```
fig=plt.figure()
ax=fig.add_subplot(111)
cax=ax.matshow(dataset.corr(),vmin=1,vmax=1)
fig.colorbar(cax)
plt.show()
```



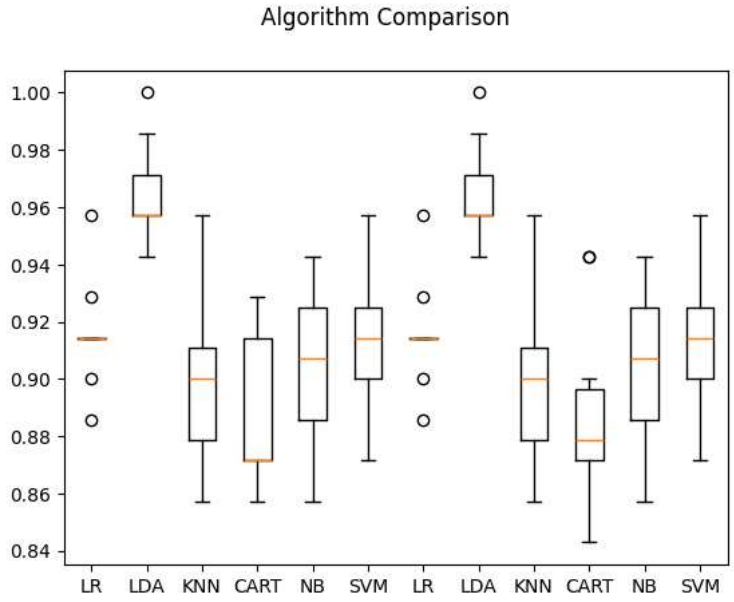
```
array=dataset.values
X=array[:,0:7]
Y=array[:,7]
seed=8
#diving data into x and y variables as independent and dependent variables
print(X,Y)
```

```
[[15.26   14.84    0.871  ...  3.312   2.221   5.22  ]
 [14.88   14.57    0.8811 ...  3.333   1.018   4.956 ]
 [14.29   14.09    0.905  ...  3.337   2.699   4.825 ]
 ...
 [13.2    13.66    0.8883 ...  3.232   8.315   5.056 ]
 [11.84   13.21    0.8521 ...  2.836   3.598   5.044 ]
 [12.3    13.34    0.8684 ...  2.974   5.637   5.063 ]] [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 2. 2.
2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 3. 3. 3. 3.
3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.]
```

```
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))


results = []
names = []
scoring = 'accuracy'
test_size = 0.33
seed = 8


for name, model in models:
    kfold = model_selection.ShuffleSplit(n_splits=10, test_size=test_size, random_state=seed)
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y, test_size=test_size, random_state=seed)
    cv_results = model_selection.cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean()*100, cv_results.std())
    print(msg)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=1)
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=1)
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=1)
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=1)
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=1)
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=1)
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=1)
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.
```

```
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```




```
pipelines = []
pipelines.append(('ScaledLR', Pipeline([('Scaler', StandardScaler()),('LR', LogisticRegression())])))
pipelines.append(('ScaledLDA', Pipeline([('Scaler', StandardScaler()),('LDA', LinearDiscriminantAnalysis())])))
pipelines.append(('ScaledKNN', Pipeline([('Scaler', StandardScaler()),('KNN', KNeighborsClassifier())])))
pipelines.append(('ScaledCART', Pipeline([('Scaler', StandardScaler()),('CART', DecisionTreeClassifier())])))
pipelines.append(('ScaledNB', Pipeline([('Scaler', StandardScaler()),('NB', GaussianNB())])))
pipelines.append(('ScaledSVM', Pipeline([('Scaler', StandardScaler()),('SVM', SVC())])))
results = []
names = []
for name, model in pipelines:
    kfold = model_selection.ShuffleSplit(n_splits=10, test_size=test_size, random_state=seed)
    #X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y, test_size=test_size, random_state=seed)
    cv_results = model_selection.cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean()*100, cv_results.std())
    print(msg)
```

```
ScaledLR: 93.285714 (0.015714)
ScaledLDA: 96.428571 (0.017202)
ScaledKNN: 92.571429 (0.029137)
ScaledCART: 89.428571 (0.037362)
ScaledNB: 90.285714 (0.026186)
ScaledSVM: 93.000000 (0.016225)
```

```
scaler = StandardScaler().fit(X)   # KNN
rescaledX = scaler.transform(X)
neighbors = [1,3,5,7,9,11,13,15,17,19,21]
param_grid = dict(n_neighbors=neighbors)
model = KNeighborsClassifier()
kfold = model_selection.ShuffleSplit(n_splits=10, test_size=test_size, random_state=seed)
#X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y, test_size=test_size, random_state=seed)
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring=scoring, cv=kfold)
grid_result = grid.fit(rescaledX, Y)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
Best: 0.931429 using {'n_neighbors': 1}
```

```
scaler = StandardScaler().fit(X)                        #SVM
rescaledX = scaler.transform(X)
c_values = [0.1, 0.3, 0.5, 0.7, 0.9, 1.0, 1.3, 1.5, 1.7, 2.0]
kernel_values = ['linear', 'poly', 'rbf', 'sigmoid']
```

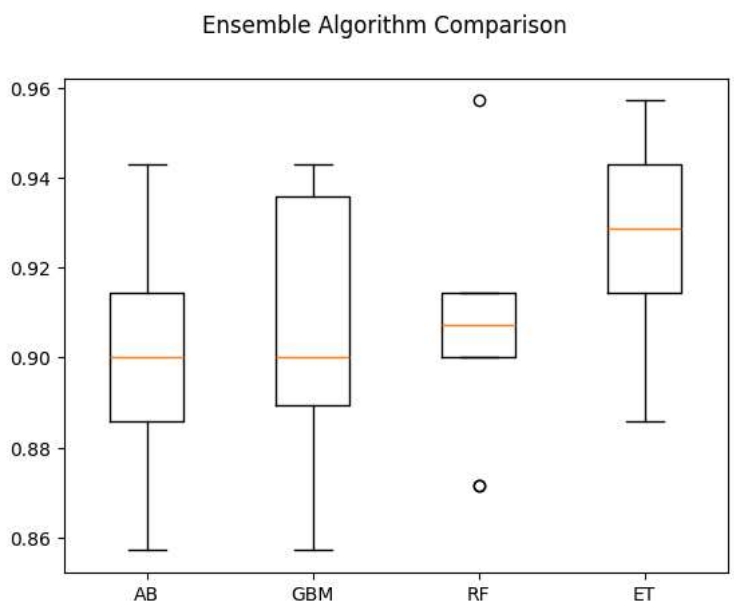
```
param_grid = dict(C=c_values, kernel=kernel_values)
model = SVC()
kfold = model_selection.ShuffleSplit(n_splits=10, test_size=test_size, random_state=seed)
#X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y, test_size=test_size, random_state=seed)
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring=scoring, cv=kfold)
grid_result = grid.fit(rescaledX, Y)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
Best: 0.941429 using {'C': 1.7, 'kernel': 'linear'}
```

```
ensembles = []
ensembles.append(('AB', AdaBoostClassifier()))
ensembles.append(('GBM', GradientBoostingClassifier()))
ensembles.append(('RF', RandomForestClassifier()))
ensembles.append(('ET', ExtraTreesClassifier()))
results = []
names = []
for name, model in ensembles:
    kfold = model_selection.ShuffleSplit(n_splits=10, test_size=test_size, random_state=seed)
    #X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y, test_size=test_size, random_state=seed)
    cv_results = model_selection.cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = " mean %s: %f ( standard deviation %f)" % (name, cv_results.mean()*100, cv_results.std())
    print(msg)
```

```
 mean AB: 89.857143 ( standard deviation 0.026688)
 mean GBM: 90.714286 ( standard deviation 0.027293)
 mean RF: 91.000000 ( standard deviation 0.028607)
 mean ET: 92.285714 ( standard deviation 0.027255)
```

```
fig = plt.figure()
fig.suptitle('Ensemble Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```




```
scaler = StandardScaler().fit(X);
rescaledX = scaler.transform(X)
model = LinearDiscriminantAnalysis()
model.fit(rescaledX, Y)
```

```
▾ LinearDiscriminantAnalysis
LinearDiscriminantAnalysis()
```

```
rescaledValidationX = scaler.transform(X)
predictions = model.predict(rescaledValidationX)
```

```
print(accuracy_score(Y, predictions))
print("  ")
print(confusion_matrix(Y, predictions))
print("  ")
print(classification_report(Y,predictions))
```

```
0.9666666666666667

[[66  1  3]
 [ 0 70  0]
 [ 3  0 67]]

              precision    recall  f1-score   support

         1.0       0.96      0.94      0.95        70
         2.0       0.99      1.00      0.99        70
         3.0       0.96      0.96      0.96        70

    accuracy                           0.97       210
   macro avg       0.97      0.97      0.97       210
weighted avg       0.97      0.97      0.97       210
```