Name : Pankaj Chawla

Roll : 001811001052

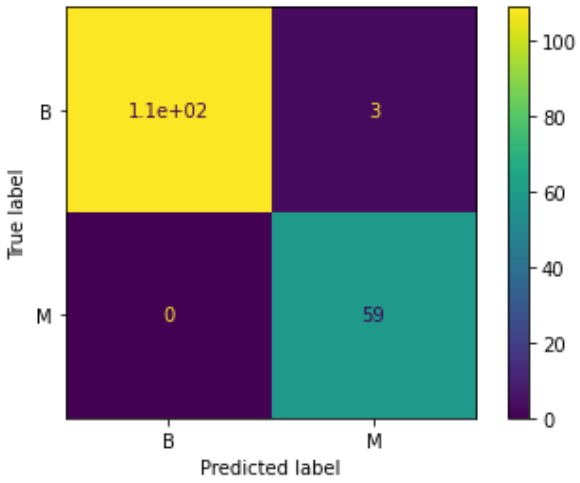
Subject : Machine Learning Lab
Assignment – 2

Department : I.T (4th year, 1st semester)

```
# BREAST CANCER DATASET
# Random Forest Classifier(Without Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']

df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------")
print("-------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------")
print("-------------------------------------------------")
```

```
print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[109   3]
 [  0  59]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       1.00      0.97      0.99       112
           M       0.95      1.00      0.98        59

    accuracy                           0.98       171
   macro avg       0.98      0.99      0.98       171
weighted avg       0.98      0.98      0.98       171

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9824561403508771
```




```
# BREAST CANCER DATASET
# Random Forest Classifier(Without Tuning)[60-40 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']

df.columns = col_name
```

```
df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------------")
print("-------------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------------")
print("-------------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[145   4]
 [  1  78]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       0.99      0.97      0.98       149
           M       0.95      0.99      0.97        79

    accuracy                           0.98       228
   macro avg       0.97      0.98      0.98       228
weighted avg       0.98      0.98      0.98       228

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9780701754385965
```


```
# BREAST CANCER DATASET
# Random Forest Classifier(Without Tuning)[50-50 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']

df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,rando



# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
```

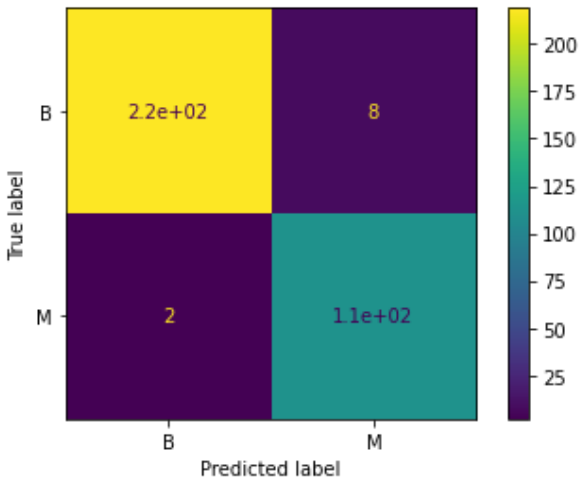
```
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[179   4]
 [  0 102]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       1.00      0.98      0.99       183
           M       0.96      1.00      0.98       102

    accuracy                           0.99       285
```

```
# BREAST CANCER DATASET
# Random Forest Classifier(Without Tuning)[40-60 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']

df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
print("-----------------------------------------------------")
print("-----------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-----------------------------------------------------")
print("-----------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[219   8]
 [  2 113]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       0.99      0.96      0.98       227
           M       0.93      0.98      0.96       115

    accuracy                           0.97       342
   macro avg       0.96      0.97      0.97       342
weighted avg       0.97      0.97      0.97       342

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9707602339181286
```

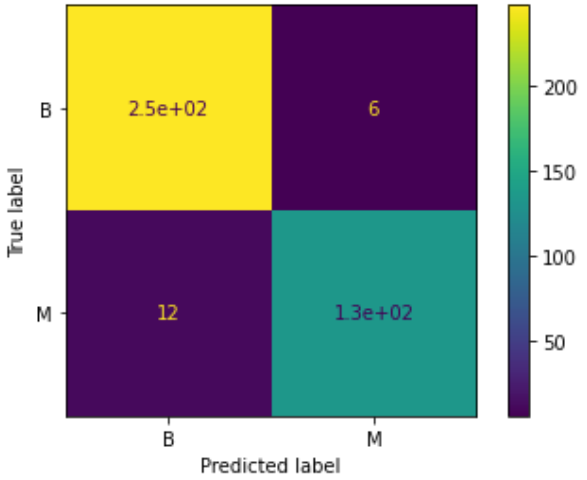



```
# BREAST CANCER DATASET
# Random Forest Classifier(Without Tuning)[30-70 split]
```

```
import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']

df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[247   6]
 [ 12 134]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       0.95      0.98      0.96       253
           M       0.96      0.92      0.94       146

    accuracy                           0.95       399
   macro avg       0.96      0.95      0.95       399
weighted avg       0.95      0.95      0.95       399

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9548872180451128
```




```
# BREAST CANCER DATASET
# Random Forest Classifier(With Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']


df.columns = col_name


X = df.drop(['Class'], axis=1)
```

```
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.30,rand


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

###############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

###############################################################################
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)

###############################################################################
```

```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
      Parameters currently in use:

      {'bootstrap': True,
       'ccp_alpha': 0.0,
       'class_weight': None,
       'criterion': 'gini',
       'max_depth': None,
       'max_features': 'auto',
       'max_leaf_nodes': None,
       'max_samples': None,
       'min_impurity_decrease': 0.0,
       'min_impurity_split': None,
       'min_samples_leaf': 1,
       'min_samples_split': 2,
       'min_weight_fraction_leaf': 0.0,
       'n_estimators': 100,
       'n_jobs': None,
       'oob_score': False,
       'random_state': None,
       'verbose': 0,
       'warm_start': False}
      {'bootstrap': [True, False],
       'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
       'max_features': ['auto', 'sqrt'],
       'min_samples_leaf': [1, 2, 4],
       'min_samples_split': [2, 5, 10],
       'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
      Fitting 3 folds for each of 100 candidates, totalling 300 fits
      [Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
      [Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:   57.6s
      [Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed:  3.9min
      [Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:  7.3min finished
      Confusion Matrix:
      [[109   3]
       [  0  59]]
      --------------------------------------------------
```

```
# BREAST CANCER DATASET
# Random Forest Classifier(With Tuning)[60-40 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

###############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

###############################################################################
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)

###############################################################################

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
```

```
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
Fitting 3 folds for each of 100 candidates, totalling 300 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:   53.9s
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed:  3.7min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:  7.0min finished
```

```
# BREAST CANCER DATASET
# Random Forest Classifier(With Tuning)[50-50 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,rando


# Feature Scaling
```

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

###############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

###############################################################################
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)

###############################################################################

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)
```

```
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

```
# BREAST CANCER DATASET
# Random Forest Classifier(With Tuning)[40-60 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,rando



# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
```

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

################################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

################################################################################
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)

################################################################################

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

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

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

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("----------------------------------------------------")
print("----------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("----------------------------------------------------")
print("----------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
      Parameters currently in use:

      {'bootstrap': True,
       'ccp_alpha': 0.0,
       'class_weight': None,
       'criterion': 'gini',
       'max_depth': None,
       'max_features': 'auto',
       'max_leaf_nodes': None,
       'max_samples': None,
       'min_impurity_decrease': 0.0,
       'min_impurity_split': None,
       'min_samples_leaf': 1,
       'min_samples_split': 2,
       'min_weight_fraction_leaf': 0.0,
       'n_estimators': 100,
       'n_jobs': None,
       'oob_score': False,
       'random_state': None,
       'verbose': 0,
       'warm_start': False}
      {'bootstrap': [True, False],
       'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
       'max_features': ['auto', 'sqrt']
```

```
# BREAST CANCER DATASET
# Random Forest Classifier(With Tuning)[30-70 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
```

```
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

###############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

###############################################################################
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)

###############################################################################

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
print("-------------------------------------------------")
print("-------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------")
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
     Parameters currently in use:

     {'bootstrap': True,
      'ccp_alpha': 0.0,
      'class_weight': None,
      'criterion': 'gini',
      'max_depth': None,
      'max_features': 'auto',
      'max_leaf_nodes': None,
      'max_samples': None,
      'min_impurity_decrease': 0.0,
      'min_impurity_split': None,
      'min_samples_leaf': 1,
      'min_samples_split': 2,
      'min_weight_fraction_leaf': 0.0,
      'n_estimators': 100,
      'n_jobs': None,
      'oob_score': False,
      'random_state': None,
```

```
##############################################################################################
```

```
     {'bootstrap': [True, False],
```

```
# BREAST CANCER DATASET
# Multi Layer Perceptron(Without Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']


df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.30,rand


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification using MLP
from sklearn.neural_network import MLPClassifier
```

```
classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[110   2]
 [  2  57]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       0.98      0.98      0.98       112
           M       0.97      0.97      0.97        59

    accuracy                           0.98       171
```

```
# BREAST CANCER DATASET
# Multi Layer Perceptron(Without Tuning)[60-40 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
          ,'20','21','22','23','24','25','26','27','28','29','30','31','32']


df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

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

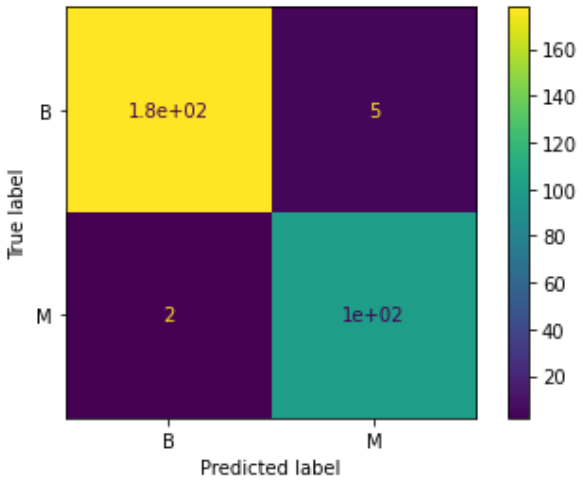
```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------------")
print("-------------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------------")
print("-------------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[147   2]
 [  2  77]]
--------------------------------------------------
```

```
# BREAST CANCER DATASET
# Multi Layer Perceptron(Without Tuning)[50-50 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']


df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")
```

```
print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("----------------------------------------------------")
print("----------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

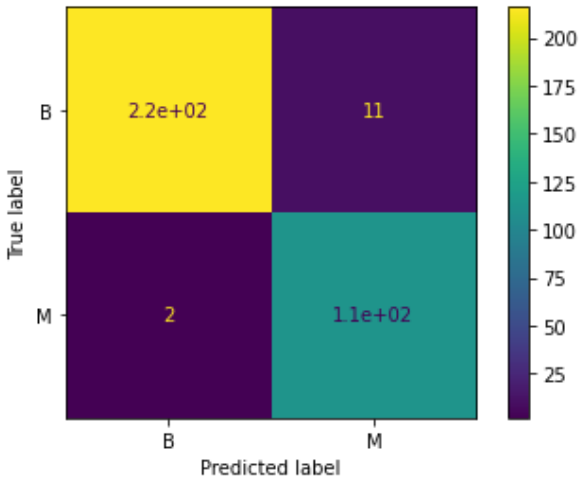
```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[178   5]
 [  2 100]]
----------------------------------------------------
----------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       0.99      0.97      0.98       183
           M       0.95      0.98      0.97       102

    accuracy                           0.98       285
   macro avg       0.97      0.98      0.97       285
weighted avg       0.98      0.98      0.98       285

----------------------------------------------------
----------------------------------------------------
Accuracy:
0.9754385964912281
```



```
# BREAST CANCER DATASET
# Multi Layer Perceptron(Without Tuning)[40-60 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']


df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------")
print("-------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------")
print("-------------------------------------------------")
```

```
print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[216  11]
 [  2 113]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       0.99      0.95      0.97       227
           M       0.91      0.98      0.95       115

    accuracy                           0.96       342
   macro avg       0.95      0.97      0.96       342
weighted avg       0.96      0.96      0.96       342

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9619883040935673
```

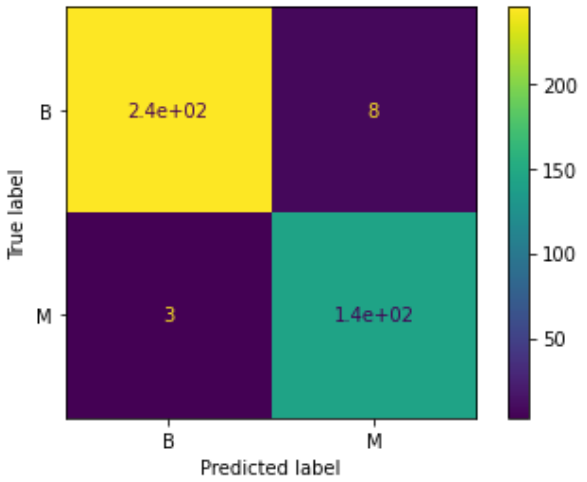



```
# BREAST CANCER DATASET
# Multi Layer Perceptron(Without Tuning)[30-70 split]


import pandas as pd
import numpy as np

# Dataset Preparation
```

```
# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']


df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------")
print("-------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------")
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[245   8]
 [  3 143]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       0.99      0.97      0.98       253
           M       0.95      0.98      0.96       146

    accuracy                           0.97       399
   macro avg       0.97      0.97      0.97       399
weighted avg       0.97      0.97      0.97       399

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9724310776942355
```




```
# BREAST CANCER DATASET
# Multi Layer Perceptron(With Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']


df.columns = col_name
```

```
df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

###############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

###############################################################################
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

###############################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)
```

```
y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
{'activation': ['tanh', 'relu'],
 'alpha': [0.0001, 0.05],
```

```
# BREAST CANCER DATASET
# Multi Layer Perceptron(With Tuning)[60-40 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']


df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)

# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

###############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

###############################################################################
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

###############################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----------------------------------------------------")
print("-----------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-----------------------------------------------------")
```

```
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
      Parameters currently in use:

      {'activation': 'relu',
       'alpha': 0.0001,
       'batch_size': 'auto',
       'beta_1': 0.9,
       'beta_2': 0.999,
       'early_stopping': False,
       'epsilon': 1e-08,
```

```
# BREAST CANCER DATASET
# Multi Layer Perceptron(With Tuning)[50-50 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']


df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

##############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

##############################################################################
# Creating a set of important sample features
```

```
parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

################################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------")
print("-------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------")
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
{'activation': ['tanh', 'relu'],
 'alpha': [0.0001, 0.05],
 'hidden_layer_sizes': [(50, 50, 50), (50, 100, 50), (100,)],
 'learning_rate': ['constant', 'adaptive'],
 'solver': ['sgd', 'adam']}
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[177   6]
 [  0 102]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       1.00      0.97      0.98       183
           M       0.94      1.00      0.97       102

    accuracy                           0.98       285
   macro avg       0.97      0.98      0.98       285
weighted avg       0.98      0.98      0.98       285

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9789473684210527
```


```
# BREAST CANCER DATASET
# Multi Layer Perceptron(With Tuning)[40-60 split]


import pandas as pd
import numpy as np
```

```
# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']


df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

###############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

###############################################################################
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

###############################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
```

```
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
{'activation': ['tanh', 'relu'],
 'alpha': [0.0001, 0.05],
 'hidden_layer_sizes': [(50, 50, 50), (50, 100, 50), (100,)],
 'learning_rate': ['constant', 'adaptive'],
 'solver': ['sgd', 'adam']}
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[220   7]
 [  2 113]]
```

```
# BREAST CANCER DATASET
# Multi Layer Perceptron(With Tuning)[30-70 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']


df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.3, test_size=0.7, rando
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

###############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

###############################################################################
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

###############################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------")
print("-------------------------------------------------")
```

```
print(                                                    )


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------")
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
      Parameters currently in use:

      {'activation': 'relu',
       'alpha': 0.0001,
       'batch_size': 'auto',
       'beta_1': 0.9,
       'beta_2': 0.999,
       'early_stopping': False,
       'epsilon': 1e-08,
       'hidden_layer_sizes': (100,),
       'learning_rate': 'constant',
       'learning_rate_init': 0.001,
       'max_fun': 15000,
       'max_iter': 100,
       'momentum': 0.9,
       'n_iter_no_change': 10,
       'nesterovs_momentum': True,
       'power_t': 0.5
```

```
##########################################################################################
```

```
       'solver': 'adam'
```

```
# BREAST CANCER DATASET
# SVM(Without Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']


df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.svm import SVC

classifier = SVC()
```

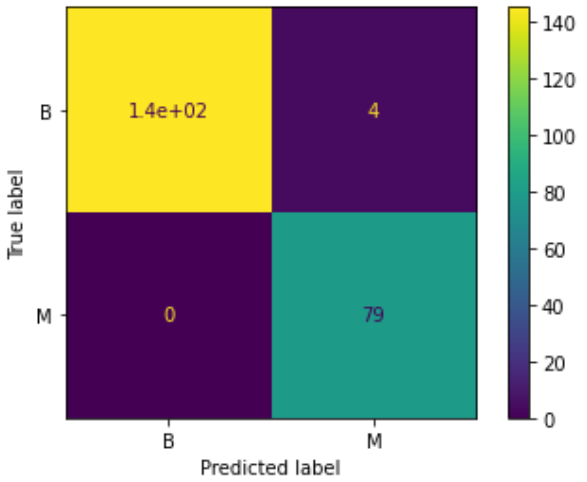
```
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[109   3]
 [  0  59]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       1.00      0.97      0.99       112
```

```
# BREAST CANCER DATASET
# SVM(Without Tuning)[60-40 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']


df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
```

```
print("-------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------")
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[145   4]
 [  0  79]]
-------------------------------------------------
-------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       1.00      0.97      0.99       149
           M       0.95      1.00      0.98        79

    accuracy                           0.98       228
   macro avg       0.98      0.99      0.98       228
weighted avg       0.98      0.98      0.98       228

-------------------------------------------------
-------------------------------------------------
Accuracy:
0.9824561403508771
```

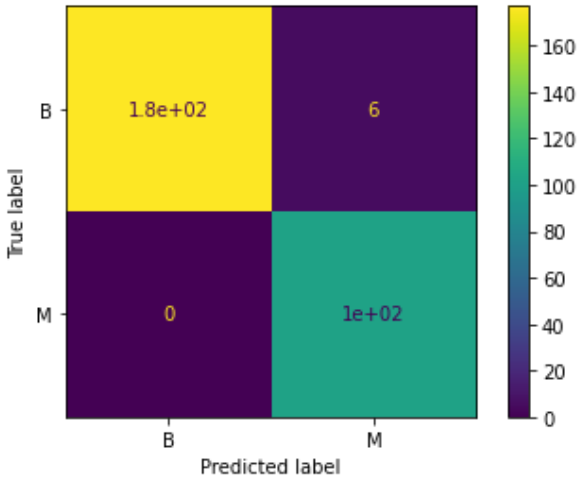



```
# BREAST CANCER DATASET
```

```
# SVM(Without Tuning)[50-50 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']


df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")
```

```
print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

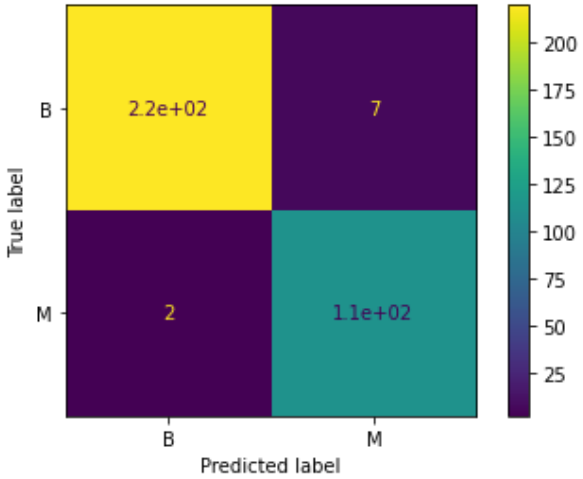
```
Confusion Matrix:
[[177   6]
 [  0 102]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       1.00      0.97      0.98       183
           M       0.94      1.00      0.97       102

    accuracy                           0.98       285
   macro avg       0.97      0.98      0.98       285
weighted avg       0.98      0.98      0.98       285

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9789473684210527
```




```
# BREAST CANCER DATASET
# SVM(Without Tuning)[40-60 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
```

```
            ,'20','21','22','23','24','25','26','27','28','29','30','31','32']


df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------")
print("-------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------")
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

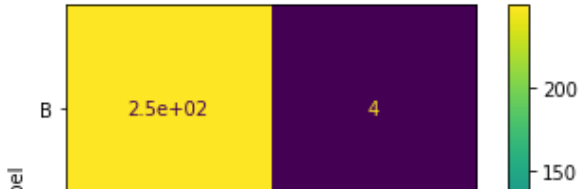
```
Confusion Matrix:
[[220   7]
 [  2 113]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       0.99      0.97      0.98       227
           M       0.94      0.98      0.96       115

    accuracy                           0.97       342
   macro avg       0.97      0.98      0.97       342
weighted avg       0.97      0.97      0.97       342

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9736842105263158
```




```
# BREAST CANCER DATASET
# SVM(Without Tuning)[30-70 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']


df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[249   4]
 [ 10 136]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       0.96      0.98      0.97       253
           M       0.97      0.93      0.95       146

    accuracy                           0.96       399
   macro avg       0.97      0.96      0.96       399
weighted avg       0.97      0.96      0.96       399

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9649122807017544
```




```
# BREAST CANCER DATASET
# SVM(With Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']


df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

```
# Classification
from sklearn.svm import SVC

classifier = SVC()

################################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

################################################################################
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly',

pprint(param_grid)

################################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------")
print("-------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------")
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
     Parameters currently in use:

     {'C': 1.0,
      'break_ties': False,
```

```
# BREAST CANCER DATASET
# SVM(With Tuning)[60-40 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']


df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Classification
from sklearn.svm import SVC

classifier = SVC()

#################################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#################################################################################
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly',
```

```
pprint(param_grid)

################################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
{'C': [0.1, 1, 10, 100],
 'gamma': [1, 0.1, 0.01, 0.001],
 'kernel': ['rbf', 'poly', 'sigmoid']}
Fitting 5 folds for each of 48 candidates, totalling 240 fits
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .................................
[CV] .................. C=0.1, gamma=1, kernel=sigmoid, total=   0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .................................
[CV] .................. C=0.1, gamma=1, kernel=sigmoid, total=   0.0s
```

```
# BREAST CANCER DATASET
# SVM(With Tuning)[50-50 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
          ,'20','21','22','23','24','25','26','27','28','29','30','31','32']
```

```
df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Classification
from sklearn.svm import SVC

classifier = SVC()

########################################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

########################################################################################
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly',

pprint(param_grid)

########################################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

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

```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
{'C': [0.1, 1, 10, 100],
 'gamma': [1, 0.1, 0.01, 0.001],
 'kernel': ['rbf', 'poly', 'sigmoid']}
Fitting 5 folds for each of 48 candidates, totalling 240 fits
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ....................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf
```

```
# BREAST CANCER DATASET
# SVM(With Tuning)[40-60 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']


df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

```
# Classification
from sklearn.svm import SVC

classifier = SVC()

################################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

################################################################################
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly',

pprint(param_grid)

################################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
     Parameters currently in use:

     {'C': 1.0
```

```
# BREAST CANCER DATASET
# SVM(With Tuning)[30-70 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']


df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Classification
from sklearn.svm import SVC

classifier = SVC()

#############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#############################################################################
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly',

pprint(param_grid)
```

```
################################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

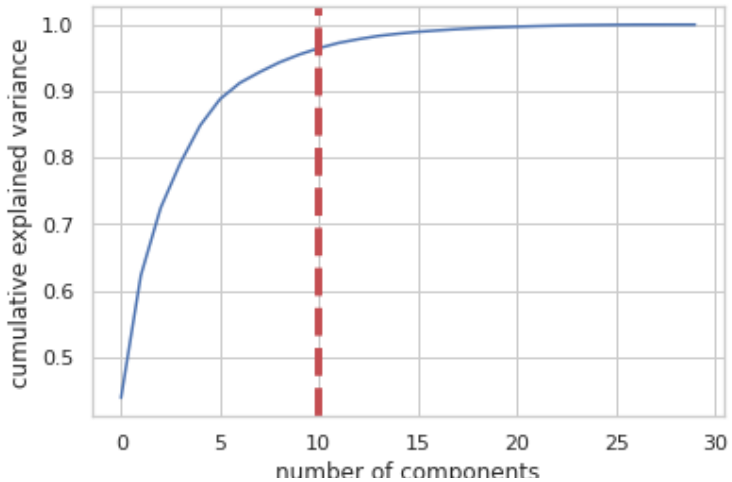
```
Parameters currently in use:

{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
{'C': [0.1, 1, 10, 100],
 'gamma': [1, 0.1, 0.01, 0.001],
 'kernel': ['rbf', 'poly', 'sigmoid']}
Fitting 5 folds for each of 48 candidates, totalling 240 fits
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .................................
[CV] .................. C=0.1, gamma=1, kernel=sigmoid, total=   0.0s
```

```
##########################################################################################
[CV] C=0.1, gamma=1, kernel=sigmoid .................................
```

```
# BREAST CANCER DATASET
# Random Forest Classifier(Without Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
          ,'20','21','22','23','24','25','26','27','28','29','30','31','32']
```

```
df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=30)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters

pca = PCA(n_components=10)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
```

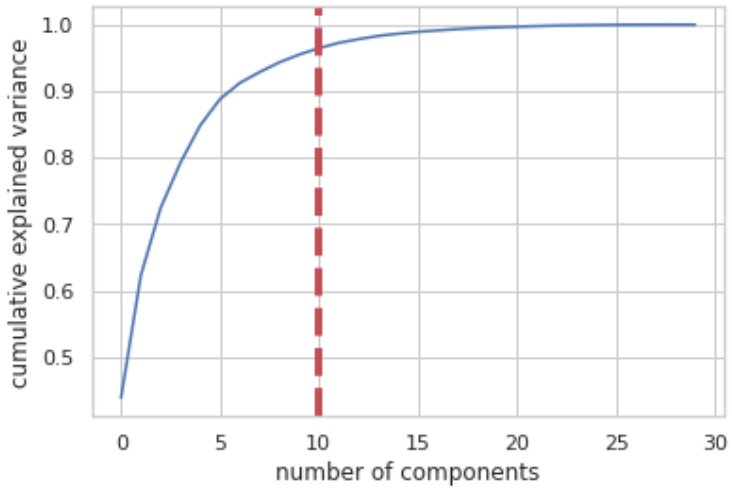
```
print("-----------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-----------------------------------------------------")
print("-----------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```




```
# BREAST CANCER DATASET
# Random Forest Classifier(With Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']


df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.30,rand


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=30)
pca_test.fit(X_train)
```

```
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters

pca = PCA(n_components=10)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)


# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

###############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

###############################################################################
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)

###############################################################################
```

```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

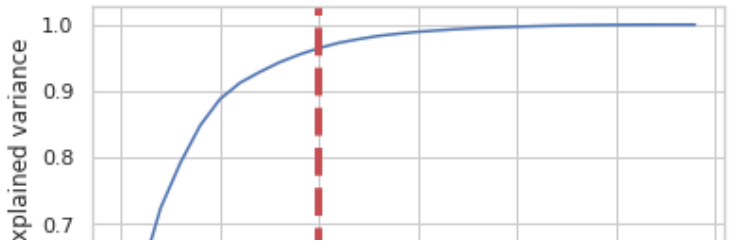



```
None
Parameters currently in use:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

```
# BREAST CANCER DATASET
# Multi Layer Perceptron(Without Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']


df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.30,rand


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=30)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters

pca = PCA(n_components=10)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)


# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))
```

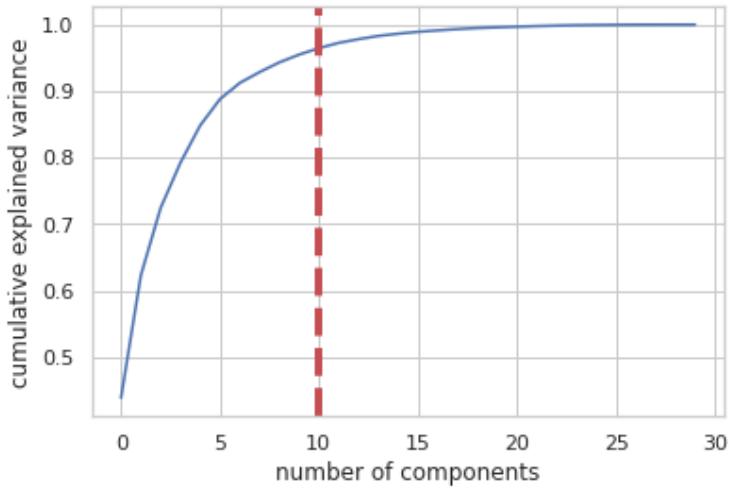
```
print("-------------------------------------------------")
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```




```
# BREAST CANCER DATASET
# Multi Layer Perceptron(With Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']


df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=30)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())
```

```
# So we can see that we have 10 important parameters

pca = PCA(n_components=10)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)


# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

##############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

##############################################################################
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

##############################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------")
print("-------------------------------------------------")
```

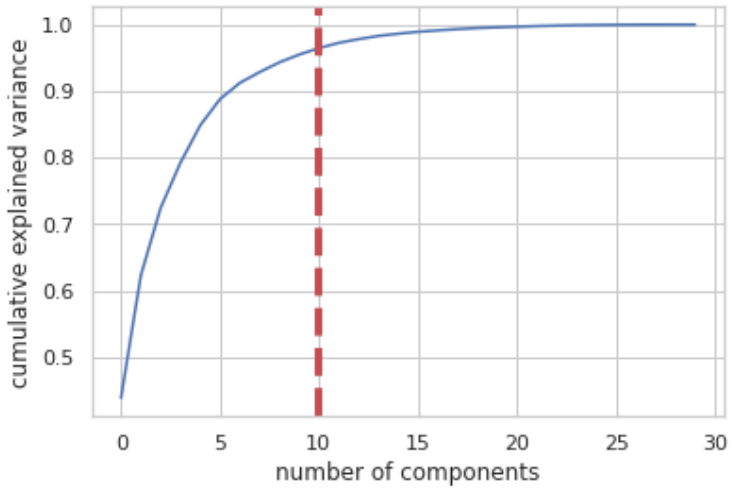
```
print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-----------------------------------------------------")
print("-----------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```



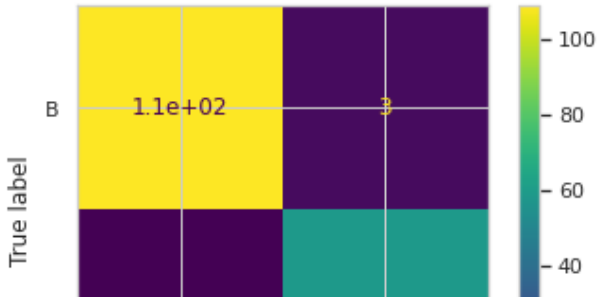

```
None
```

```
# BREAST CANCER DATASET
# SVM(Without Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']


df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
```

```
pca_test = PCA(n_components=30)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters

pca = PCA(n_components=10)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)


# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```



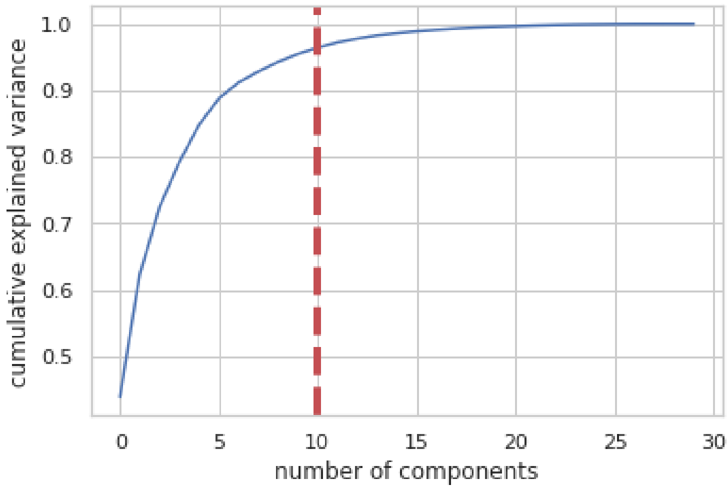

```
None
Confusion Matrix:
[[109   3]
 [  0  59]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       1.00      0.97      0.99       112
           M       0.95      1.00      0.98        59

    accuracy                           0.98       171
   macro avg       0.98      0.99      0.98       171
weighted avg       0.98      0.98      0.98       171

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9824561403508771
```




```
# BREAST CANCER DATASET
# SVM(With Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']
```

```
df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=30)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters

pca = PCA(n_components=10)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)


# Classification
from sklearn.svm import SVC

classifier = SVC()

########################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())
```

```
################################################################################
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly',

pprint(param_grid)

################################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```



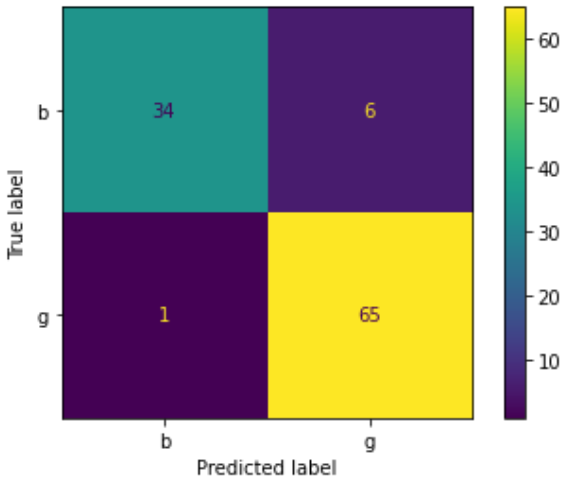

```
None
Parameters currently in use:

{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
{'C': [0.1, 1, 10, 100],
 'gamma': [1, 0.1, 0.01, 0.001],
 'kernel': ['rbf', 'poly', 'sigmoid']}
Fitting 5 folds for each of 48 candidates, totalling 240 fits
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s

[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .................................
[CV] .................. C=0.1, gamma=1, kernel=sigmoid, total=   0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .................................
```

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

```
# IONOSPHERE DATASET
# Random Forest Classifier(Without Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.30,rand


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
```

```
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

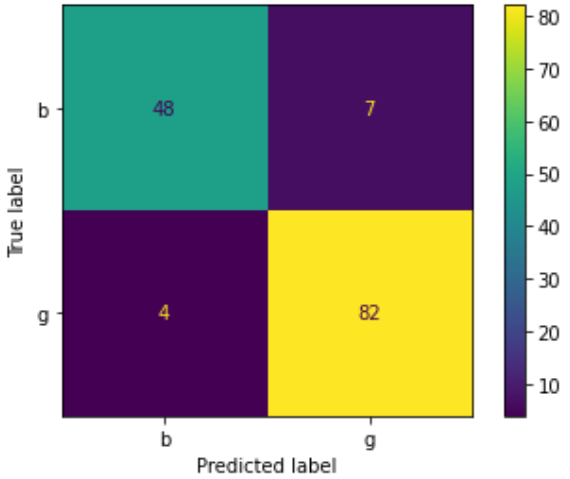
```
Confusion Matrix:
[[34  6]
 [ 1 65]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.97      0.85      0.91        40
           g       0.92      0.98      0.95        66

    accuracy                           0.93       106
   macro avg       0.94      0.92      0.93       106
weighted avg       0.94      0.93      0.93       106

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9339622641509434
```




```
# IONOSPHERE DATASET
# Random Forest Classifier(Without Tuning)[60-40 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)
```

```
col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------")
print("-------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------")
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

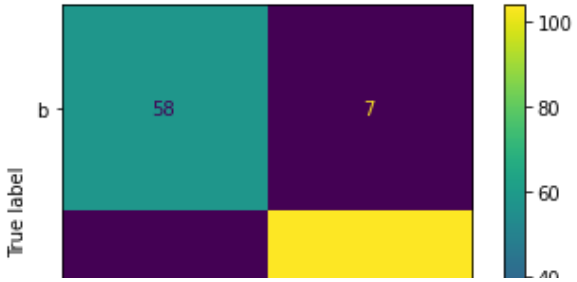
```
Confusion Matrix:
[[48  7]
 [ 4 82]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.92      0.87      0.90        55
           g       0.92      0.95      0.94        86

    accuracy                           0.92       141
   macro avg       0.92      0.91      0.92       141
weighted avg       0.92      0.92      0.92       141

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9219858156028369
```




```
# IONOSPHERE DATASET
# Random Forest Classifier(Without Tuning)[50-50 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[ 58   7]
 [  7 104]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.89      0.89      0.89        65
           g       0.94      0.94      0.94       111

    accuracy                           0.92       176
   macro avg       0.91      0.91      0.91       176
weighted avg       0.92      0.92      0.92       176

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9204545454545454
```




```
# IONOSPHERE DATASET
# Random Forest Classifier(Without Tuning)[40-60 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
```

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[ 66  10]
 [  7 128]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.90      0.87      0.89        76
           g       0.93      0.95      0.94       135

    accuracy                           0.92       211
   macro avg       0.92      0.91      0.91       211
weighted avg       0.92      0.92      0.92       211

--------------------------------------------------
```

```
# IONOSPHERE DATASET
# Random Forest Classifier(Without Tuning)[30-70 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,rando



# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)
```

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

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
    Confusion Matrix:
    [[ 76   8]
     [ 13 149]]
    -------------------------------------------------
    -------------------------------------------------
```

```
##########################################################################################
```

```
# IONOSPHERE DATASET
# Random Forest Classifier(With Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

#########################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#########################################################################
# Creating a set of important sample features
```

```
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)

################################################################################

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
     Parameters currently in use:

     {'bootstrap': True,
      'ccp_alpha': 0.0,
```

```
# IONOSPHERE DATASET
# Random Forest Classifier(With Tuning)[60-40 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

##############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

##############################################################################
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
```

```
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)

################################################################################

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
```

```
    plt.show()
```

```
# IONOSPHERE DATASET
# Random Forest Classifier(With Tuning)[50-50 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

##############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

##############################################################################
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
```

```
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)

################################################################################

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
Fitting 3 folds for each of 100 candidates, totalling 300 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:   49.3s
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed:  3.4min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:  6.4min finished
Confusion Matrix:
[[ 57   8]
 [  5 106]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.92      0.88      0.90        65
           g       0.93      0.95      0.94       111

    accuracy                           0.93       176
   macro avg       0.92      0.92      0.92       176
weighted avg       0.93      0.93      0.93       176

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9261363636363636
```


```
# IONOSPHERE DATASET
# Random Forest Classifier(With Tuning)[40-60 split]
```

```
import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

##############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

##############################################################################
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
```

```
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)

################################################################################

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
Fitting 3 folds for each of 100 candidates, totalling 300 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:   47.1s
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed:  3.5min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:  6.9min finished
Confusion Matrix:
[[ 66  10]
 [  6 129]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.92      0.87      0.89        76
           g       0.93      0.96      0.94       135

    accuracy                           0.92       211
   macro avg       0.92      0.91      0.92       211
weighted avg       0.92      0.92      0.92       211

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9241706161137441
```


```
# IONOSPHERE DATASET
# Random Forest Classifier(With Tuning)[30-70 split]


import pandas as pd
import numpy as np
```

```
# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

##############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

##############################################################################
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
```

```
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)

################################################################################

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

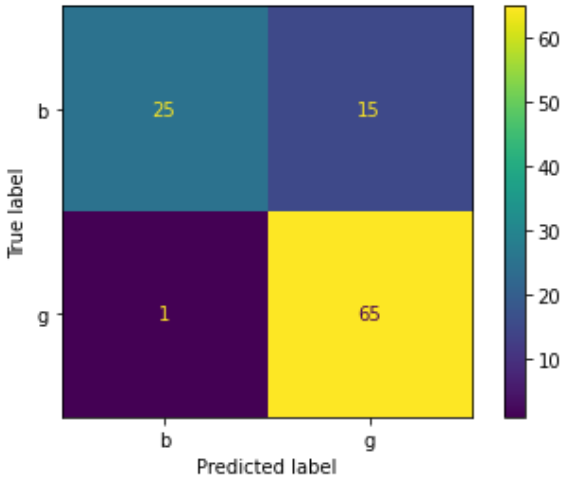
```
Parameters currently in use:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
Fitting 3 folds for each of 100 candidates, totalling 300 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:   48.1s
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed:  3.4min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:  6.4min finished
Confusion Matrix:
[[ 75   9]
 [  9 153]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.89      0.89      0.89        84
           g       0.94      0.94      0.94       162

    accuracy                           0.93       246
   macro avg       0.92      0.92      0.92       246
weighted avg       0.93      0.93      0.93       246

--------------------------------------------------
```

```
##########################################################################################################
```

```
# IONOSPHERE DATASET
# Multi Layer Perceptron(Without Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
```

```
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

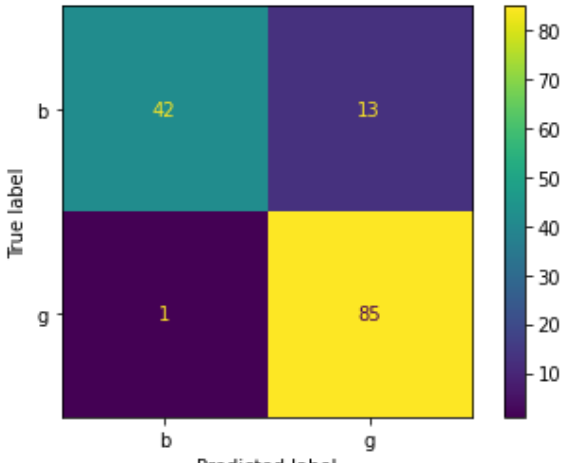
```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[25 15]
 [ 1 65]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.96      0.62      0.76        40
           g       0.81      0.98      0.89        66

    accuracy                           0.85       106
   macro avg       0.89      0.80      0.82       106
weighted avg       0.87      0.85      0.84       106

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.8490566037735849
```




```
# IONOSPHERE DATASET
# Multi Layer Perceptron(Without Tuning)[60-40 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------")
print("-------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------")
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

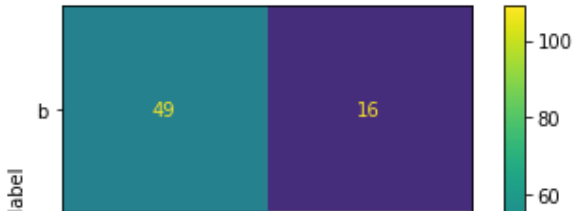
```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[42 13]
 [ 1 85]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.98      0.76      0.86        55
           g       0.87      0.99      0.92        86

    accuracy                           0.90       141
   macro avg       0.92      0.88      0.89       141
weighted avg       0.91      0.90      0.90       141

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.900709219858156
```




```
# IONOSPHERE DATASET
# Multi Layer Perceptron(Without Tuning)[50-50 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[ 49  16]
 [  2 109]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.96      0.75      0.84        65
           g       0.87      0.98      0.92       111

    accuracy                           0.90       176
   macro avg       0.92      0.87      0.88       176
weighted avg       0.90      0.90      0.89       176

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.8977272727272727
```




```
# IONOSPHERE DATASET
# Multi Layer Perceptron(Without Tuning)[40-60 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
```

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

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[ 55  21]
 [  0 135]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       1.00      0.72      0.84        76
           g       0.87      1.00      0.93       135

    accuracy                           0.90       211
   macro avg       0.93      0.86      0.88       211
weighted avg       0.91      0.90      0.90       211

--------------------------------------------------
--------------------------------------------------
Accuracy:
```

```
# IONOSPHERE DATASET
# Multi Layer Perceptron(Without Tuning)[30-70 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,rando



# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
```

```
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------")
print("-------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------")
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[ 63  21]
 [  3 159]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.95      0.75      0.84        84
           g       0.88      0.98      0.93       162

########################################################################################
    macro avg       0.92      0.87      0.88       246
```

```
# IONOSPHERE DATASET
# Multi Layer Perceptron(With Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando



# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

##############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
```

```
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#########################################################################
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

#########################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----------------------------------------------------")
print("-----------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-----------------------------------------------------")
print("-----------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
{'activation': ['tanh', 'relu'],
 'alpha': [0.0001, 0.05],
 'hidden_layer_sizes': [(50, 50, 50), (50, 100, 50), (100,)],
 'learning_rate': ['constant', 'adaptive'],
 'solver': ['sgd', 'adam']}
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[35  5]
 [ 1 65]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.97      0.88      0.92        40
           g       0.93      0.98      0.96        66

    accuracy                           0.94       106
   macro avg       0.95      0.93      0.94       106
weighted avg       0.95      0.94      0.94       106

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9433962264150944
```


```
# IONOSPHERE DATASET
# Multi Layer Perceptron(With Tuning)[60 40 split]
```

```
# Multi Layer Perceptron(with Tuning)[60-40 Split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

##############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

##############################################################################
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

##############################################################################
```

```
##############################################################################################
```

```
from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----------------------------------------------------")
print("-----------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-----------------------------------------------------")
print("-----------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
{'activation': ['tanh', 'relu'],
 'alpha': [0.0001, 0.05],
 'hidden_layer_sizes': [(50, 50, 50), (50, 100, 50), (100,)],
 'learning_rate': ['constant', 'adaptive'],
 'solver': ['sgd', 'adam']}
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[48  7]
 [ 0 86]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       1.00      0.87      0.93        55
```

```
# IONOSPHERE DATASET
# Multi Layer Perceptron(With Tuning)[50-50 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

##############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

##############################################################################
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

##############################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

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

```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
      Parameters currently in use:

      {'activation': 'relu',
       'alpha': 0.0001,
       'batch_size': 'auto',
       'beta_1': 0.9,
       'beta_2': 0.999,
       'early_stopping': False,
       'epsilon': 1e-08,
       'hidden_layer_sizes': (100,),
       'learning_rate': 'constant',
       'learning_rate_init': 0.001,
       'max_fun': 15000,
       'max_iter': 100,
       'momentum': 0.9,
       'n_iter_no_change': 10,
       'nesterovs_momentum': True,
       'power_t': 0.5,
       'random_state': None,
       'shuffle': True,
       'solver': 'adam',
       'tol': 0.0001,
       'validation_fraction': 0.1,
       'verbose': False
```

```
# IONOSPHERE DATASET
# Multi Layer Perceptron(With Tuning)[40-60 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
```

```
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

###############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

###############################################################################
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

###############################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------------")
print("-------------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------------")
print("-------------------------------------------------------")

print("Accuracy:")
```

```
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
      Parameters currently in use:

      {'activation': 'relu',
       'alpha': 0.0001,
       'batch_size': 'auto',
       'beta_1': 0.9,
```

```
# IONOSPHERE DATASET
# Multi Layer Perceptron(With Tuning)[30-70 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

##############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

##############################################################################
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
```

```
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

##############################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----------------------------------------------------")
print("-----------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-----------------------------------------------------")
print("-----------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
{'activation': ['tanh', 'relu'],
 'alpha': [0.0001, 0.05],
 'hidden_layer_sizes': [(50, 50, 50), (50, 100, 50), (100,)],
 'learning_rate': ['constant', 'adaptive'],
 'solver': ['sgd', 'adam']}
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[ 69  15]
 [  0 162]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       1.00      0.82      0.90        84
           g       0.92      1.00      0.96       162

    accuracy                           0.94       246
   macro avg       0.96      0.91      0.93       246
weighted avg       0.94      0.94      0.94       246

--------------------------------------------------
--------------------------------------------------
Accuracy:
```

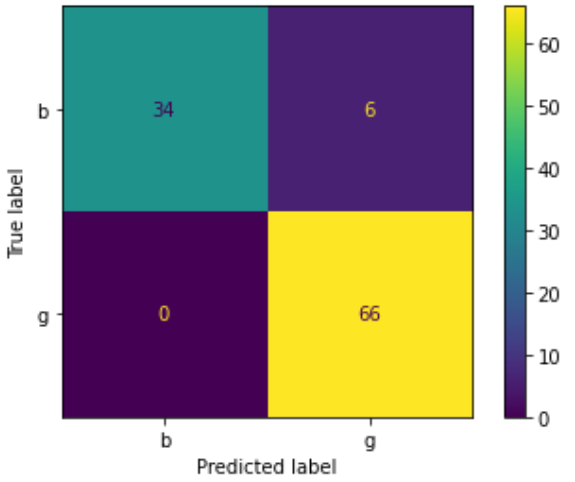
```
##############################################################################################
```

```
# IONOSPHERE DATASET
# SVM(Without Tuning)[70-30 split]


import pandas as pd
import numpy as np
```

```
# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
```

```
plt.show()
```

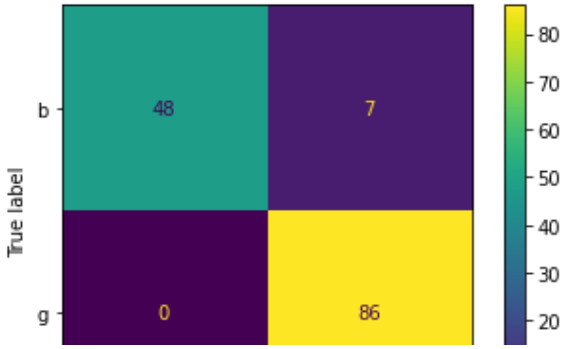
```
Confusion Matrix:
[[34  6]
 [ 0 66]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       1.00      0.85      0.92        40
           g       0.92      1.00      0.96        66

    accuracy                           0.94       106
   macro avg       0.96      0.93      0.94       106
weighted avg       0.95      0.94      0.94       106

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9433962264150944
```




```
# IONOSPHERE DATASET
# SVM(Without Tuning)[60-40 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------------")
print("-------------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------------")
print("-------------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[48  7]
 [ 0 86]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       1.00      0.87      0.93        55
           g       0.92      1.00      0.96        86

    accuracy                           0.95       141
   macro avg       0.96      0.94      0.95       141
weighted avg       0.95      0.95      0.95       141

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.950354609929078
```




```
# IONOSPHERE DATASET
# SVM(Without Tuning)[50-50 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler
```

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

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[ 55  10]
 [  2 109]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.96      0.85      0.90        65
           g       0.92      0.98      0.95       111

    accuracy                           0.93       176
   macro avg       0.94      0.91      0.92       176
weighted avg       0.93      0.93      0.93       176

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9318181818181818
```

```
# IONOSPHERE DATASET
# SVM(Without Tuning)[40-60 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,rando



# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)
```

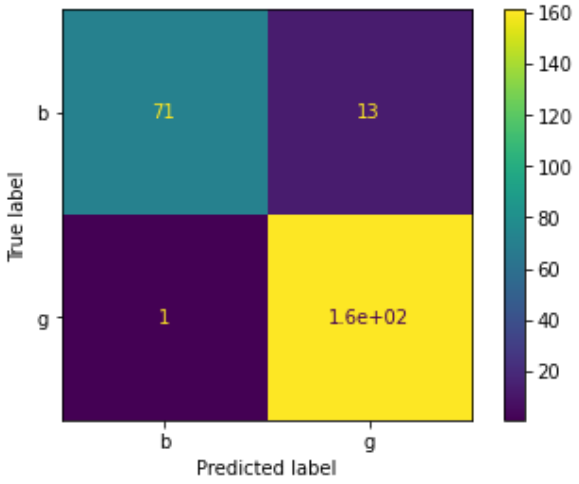
```
y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[ 64  12]
 [  1 134]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support
```

```
# IONOSPHERE DATASET
# SVM(Without Tuning)[30-70 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")
```

```
print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("----------------------------------------------------")
print("----------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[ 71  13]
 [  1 161]]
----------------------------------------------------
----------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.99      0.85      0.91        84
           g       0.93      0.99      0.96       162

    accuracy                           0.94       246
   macro avg       0.96      0.92      0.93       246
weighted avg       0.95      0.94      0.94       246

----------------------------------------------------
----------------------------------------------------
Accuracy:
0.943089430894309
```




```
##########################################################################################
```

```
# IONOSPHERE DATASET
# SVM(With Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Classification
from sklearn.svm import SVC

classifier = SVC()

##############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

##############################################################################
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly',

pprint(param_grid)

##############################################################################
```

```
from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
{'C': [0.1, 1, 10, 100],
 'gamma': [1, 0.1, 0.01, 0.001],
 'kernel': ['rbf', 'poly', 'sigmoid']}
Fitting 5 folds for each of 48 candidates, totalling 240 fits
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
```

```
# IONOSPHERE DATASET
# SVM(With Tuning)[60-40 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
          ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name



X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Classification
from sklearn.svm import SVC

classifier = SVC()

########################################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

########################################################################################
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly',

pprint(param_grid)

########################################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
```

```
print("-------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------")
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
      Parameters currently in use:

      {'C': 1.0,
       'break_ties': False,
       'cache_size': 200,
       'class_weight': None,
       'coef0': 0.0,
       'decision_function_shape': 'ovr',
       'degree': 3,
       'gamma': 'scale',
       'kernel': 'rbf',
       'max_iter': -1,
       'probability': False,
       'random_state': None,
       'shrinking': True,
       'tol': 0.001,
       'verbose': False}
      {'C': [0.1, 1, 10, 100],
```

```
# IONOSPHERE DATASET
# SVM(With Tuning)[50-50 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Classification
from sklearn.svm import SVC

classifier = SVC()
```

```
################################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

################################################################################
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly',

pprint(param_grid)

################################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
{'C': [0.1, 1, 10, 100],
 'gamma': [1, 0.1, 0.01, 0.001],
 'kernel': ['rbf', 'poly', 'sigmoid']}
Fitting 5 folds for each of 48 candidates, totalling 240 fits
[CV] C=0.1, gamma=1, kernel=rbf ......................................
[CV] ....................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf ......................................
[CV] ....................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf ......................................
[CV] ....................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf ......................................
[CV] ....................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf ......................................
[CV] ....................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly .....................................
[CV] ...................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly .....................................
[CV] ...................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly .....................................
[CV] ...................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly .....................................
[CV] ...................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly .....................................
[CV] ...................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid ..................................
[CV] ................... C=0.1, gamma=1, kernel=sigmoid, total=   0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid ..................................
[CV] ................... C=0.1, gamma=1, kernel=sigmoid, total=   0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid ..................................
[CV] ................... C=0.1, gamma=1, kernel=sigmoid, total=   0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid ..................................
[CV] ................... C=0.1, gamma=1, kernel=sigmoid, total=   0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid ..................................
[CV] ................... C=0.1, gamma=1, kernel=sigmoid, total=   0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf ....................................
[CV] ..................... C=0.1, gamma=0.1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf ....................................
[CV] ..................... C=0.1, gamma=0.1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf ....................................
[CV] ..................... C=0.1, gamma=0.1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf ....................................
[CV] ..................... C=0.1, gamma=0.1, kernel=rbf, total=   0.0s
```

```
# IONOSPHERE DATASET
```

```
# SVM(With Tuning)[40-60 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Classification
from sklearn.svm import SVC

classifier = SVC()

########################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

########################################################################
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly',

pprint(param_grid)

########################################################################

from sklearn.model_selection import GridSearchCV
```

```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
{'C': [0.1, 1, 10, 100],
 'gamma': [1, 0.1, 0.01, 0.001],
 'kernel': ['rbf', 'poly', 'sigmoid']}
Fitting 5 folds for each of 48 candidates, totalling 240 fits
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
```

```
# IONOSPHERE DATASET
# SVM(With Tuning)[30-70 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Classification
from sklearn.svm import SVC

classifier = SVC()

########################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

########################################################################
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly',

pprint(param_grid)

########################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------")
print("-------------------------------------------------")
```

```
print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------")
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

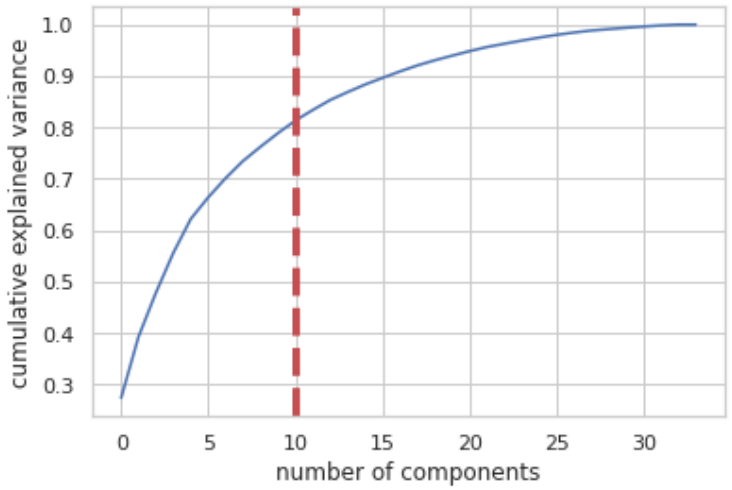
```
      Parameters currently in use:

      {'C': 1.0,
       'break_ties': False,
       'cache_size': 200,
       'class_weight': None,
       'coef0': 0.0,
       'decision_function_shape': 'ovr',
       'degree': 3,
       'gamma': 'scale',
       'kernel': 'rbf',
       'max_iter': -1,
       'probability': False,
       'random_state': None,
       'shrinking': True,
       'tol': 0.001,
       'verbose': False}
```

```
##############################################################################################
```

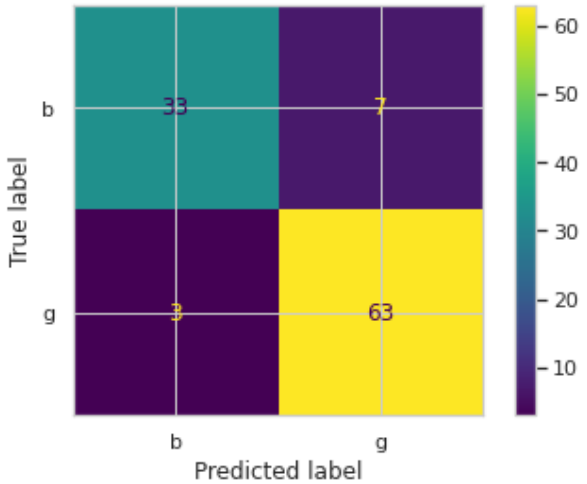
```
       'kernel': ['rbf', 'poly', 'sigmoid']}
```

```
# IONOSPHERE DATASET
# Random Forest Classifier(Without Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.30,rand


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
```

```
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=34)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 15 important parameters

pca = PCA(n_components=15)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)


# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```




```
None
Confusion Matrix:
[[33  7]
 [ 3 63]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.92      0.82      0.87        40
           g       0.90      0.95      0.93        66

    accuracy                           0.91       106
   macro avg       0.91      0.89      0.90       106
weighted avg       0.91      0.91      0.90       106

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9056603773584906
```

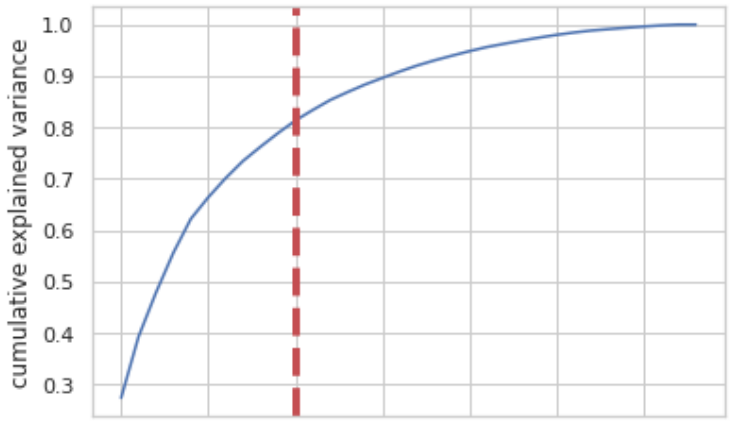



```
# IONOSPHERE DATASET
# Random Forest Classifier(With Tuning)[70-30 split]


import pandas as pd
import numpy as np
```

```
# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=34)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 15 important parameters

pca = PCA(n_components=15)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)


# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

#########################################################################
```

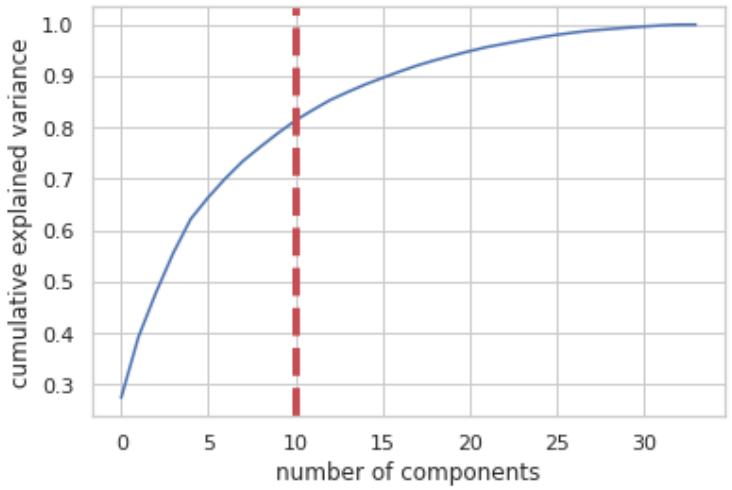
```
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

################################################################################
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)

################################################################################

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------")
print("-------------------------------------------------")
```

```
print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------")
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

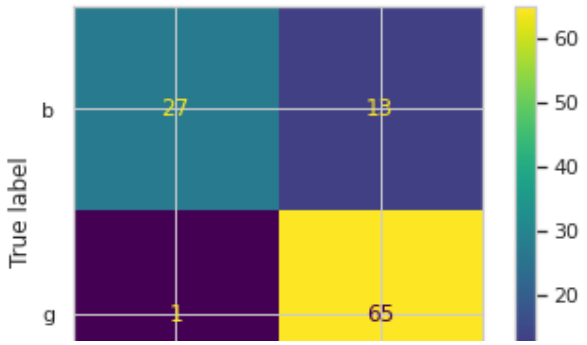



```
# IONOSPHERE DATASET
# Multi Layer Perceptron(Without Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=34)
pca_test.fit(X_train)
sns.set(style='whitegrid')
```

```
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters

pca = PCA(n_components=15)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)


# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```



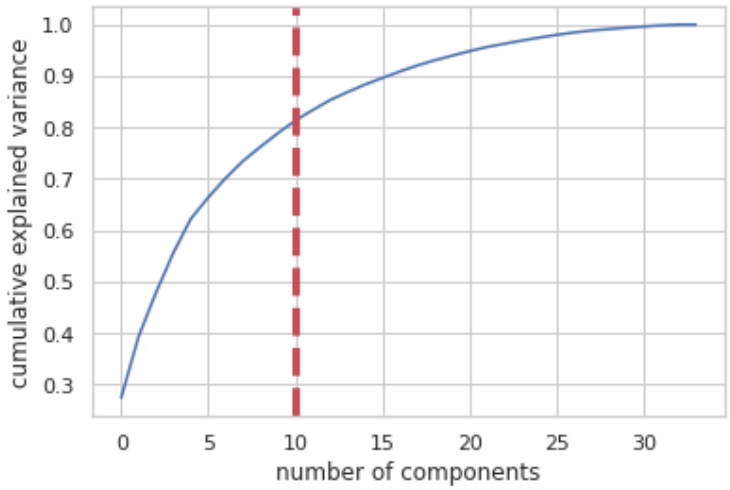

```
None
Confusion Matrix:
[[27 13]
 [ 1 65]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.96      0.68      0.79        40
           g       0.83      0.98      0.90        66

    accuracy                           0.87       106
   macro avg       0.90      0.83      0.85       106
weighted avg       0.88      0.87      0.86       106

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.8679245283018868
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
```




```
# IONOSPHERE DATASET
# Multi Layer Perceptron(With Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
```

```
col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
            ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=34)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters

pca = PCA(n_components=15)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)


# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

##############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
```

```
print('Parameters currently in use:\n')
pprint(classifier.get_params())

################################################################################
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

################################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----------------------------------------------------")
print("-----------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-----------------------------------------------------")
print("-----------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

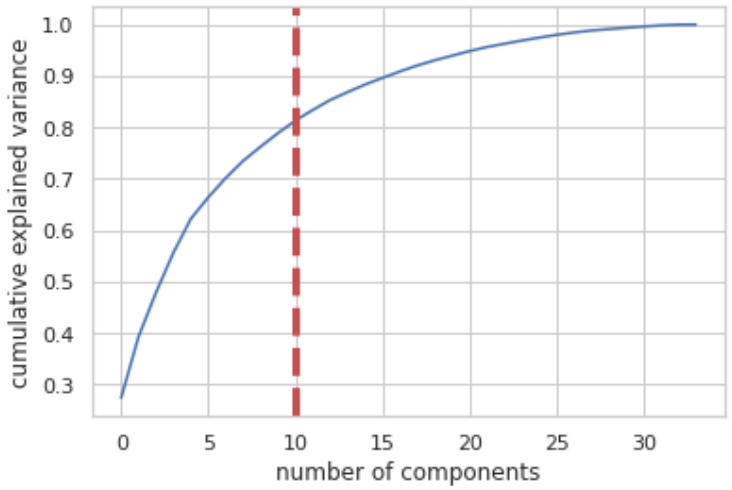



```
None
Parameters currently in use:

{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
{'activation': ['tanh', 'relu'],
 'alpha': [0.0001, 0.05],
 'hidden_layer_sizes': [(50, 50, 50), (50, 100, 50), (100,)],
 'learning_rate': ['constant', 'adaptive'],
 'solver': ['sgd', 'adam']}
Confusion Matrix:
[[30 10]
 [ 1 65]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.97      0.75      0.85        40
           g       0.87      0.98      0.92        66
```

```
# IONOSPHERE DATASET
# SVM(Without Tuning)[70-30 split]
```

```
import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=34)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters

pca = PCA(n_components=15)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)


# Classification
from sklearn.svm import SVC
```

```
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

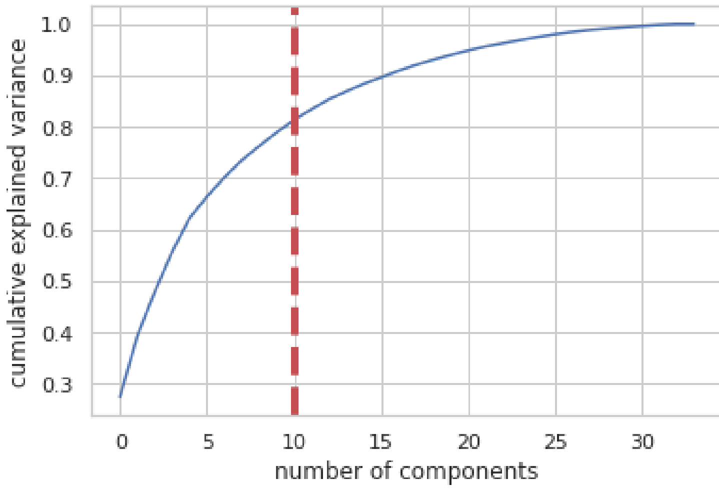



```
None
Confusion Matrix:
[[33  7]
 [ 0 66]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       1.00      0.82      0.90        40
           g       0.90      1.00      0.95        66

    accuracy                           0.93       106
```

```
# IONOSPHERE DATASET
# SVM(With Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Cl


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler
```

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


# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=34)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters

pca = PCA(n_components=15)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)


# Classification
from sklearn.svm import SVC

classifier = SVC()

#########################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#########################################################################
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly',

pprint(param_grid)

#########################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
```

```
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

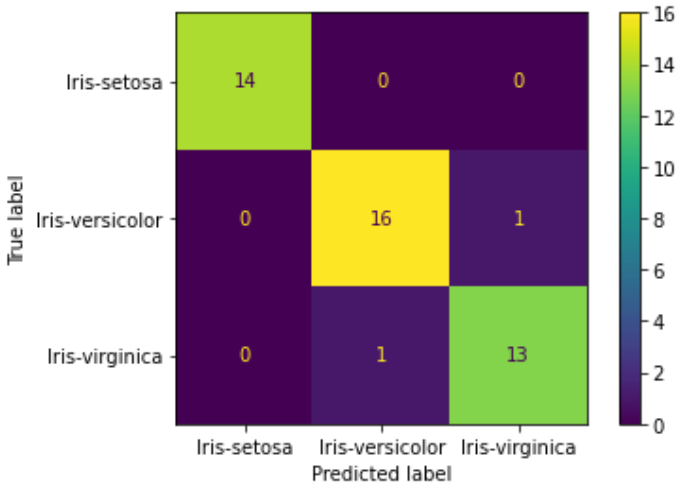



```
None
Parameters currently in use:

{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}


 'kernel': ['rbf', 'poly', 'sigmoid']}
Fitting 5 folds for each of 48 candidates, totalling 240 fits
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .................................
[CV] .................. C=0.1, gamma=1, kernel=sigmoid, total=   0.0s
```

```
# IRIS PLANT DATASET
# Random Forest Classifier(Without Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']

df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------")
print("-------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------")
print("-------------------------------------------------")
```

```
print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

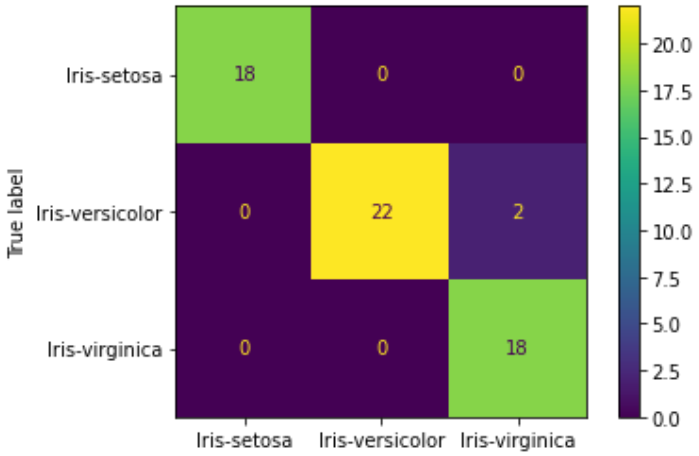
```
Confusion Matrix:
[[14  0  0]
 [ 0 16  1]
 [ 0  1 13]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        14
Iris-versicolor       0.94      0.94      0.94        17
 Iris-virginica       0.93      0.93      0.93        14

       accuracy                           0.96        45
      macro avg       0.96      0.96      0.96        45
   weighted avg       0.96      0.96      0.96        45

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9555555555555556
```




```
# IRIS PLANT DATASET
# Random Forest Classifier(Without Tuning)[60-40 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']

df.columns = col_name
```

```
df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------")
print("-------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------")
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[18  0  0]
 [ 0 22  2]
 [ 0  0 18]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        18
Iris-versicolor       1.00      0.92      0.96        24
 Iris-virginica       0.90      1.00      0.95        18

       accuracy                           0.97        60
      macro avg       0.97      0.97      0.97        60
   weighted avg       0.97      0.97      0.97        60

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9666666666666667
```




```
# IRIS PLANT DATASET
# Random Forest Classifier(Without Tuning)[50-50 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']

df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,rando
```

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[23  0  0]
 [ 0 23  4]
 [ 0  1 24]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        23
Iris-versicolor       0.96      0.85      0.90        27
 Iris-virginica       0.86      0.96      0.91        25

       accuracy                           0.93        75
      macro avg       0.94      0.94      0.94        75
   weighted avg       0.94      0.93      0.93        75

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9333333333333333
```


```
# IRIS PLANT DATASET
# Random Forest Classifier(Without Tuning)[40-60 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']

df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier
```

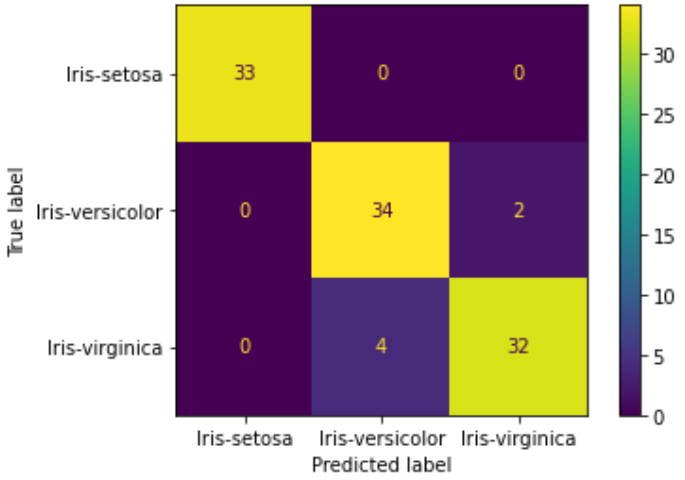
```
classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[28  0  0]
 [ 0 28  4]
 [ 0  1 29]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        28
Iris-versicolor       0.97      0.88      0.92        32
 Iris-virginica       0.88      0.97      0.92        30
```

```
# IRIS PLANT DATASET
# Random Forest Classifier(Without Tuning)[30-70 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']

df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
print("-------------------------------------------------")
print("-------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------")
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[33  0  0]
 [ 0 34  2]
 [ 0  4 32]]
-------------------------------------------------
-------------------------------------------------
Performance Evaluation
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        33
Iris-versicolor       0.89      0.94      0.92        36
 Iris-virginica       0.94      0.89      0.91        36

       accuracy                           0.94       105
      macro avg       0.95      0.94      0.94       105
   weighted avg       0.94      0.94      0.94       105

-------------------------------------------------
-------------------------------------------------
Accuracy:
0.9428571428571428
```



```
##################################################################################################


# IRIS PLANT DATASET
# Random Forest Classifier(With Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.30,rand


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

#########################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#########################################################################
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
```

```
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)

################################################################################

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
```

```
    plt.show()
```

```
# IRIS PLANT DATASET
# Random Forest Classifier(With Tuning)[60-40 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

##############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

##############################################################################
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
```

```
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)

################################################################################

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
Fitting 3 folds for each of 100 candidates, totalling 300 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:   43.7s
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed:  3.1min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:  5.9min finished
Confusion Matrix:
[[18  0  0]
 [ 0 22  2]
 [ 0  0 18]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        18
Iris-versicolor       1.00      0.92      0.96        24
 Iris-virginica       0.90      1.00      0.95        18

       accuracy                           0.97        60
      macro avg       0.97      0.97      0.97        60
   weighted avg       0.97      0.97      0.97        60

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9666666666666667
```


```
# IRIS PLANT DATASET
# Random Forest Classifier(With Tuning)[50-50 split]


import pandas as pd
```

```
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

##############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

##############################################################################
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
```

```
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)

################################################################################

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
Fitting 3 folds for each of 100 candidates, totalling 300 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:   44.2s
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed:  3.1min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:  5.9min finished
Confusion Matrix:
[[23  0  0]
 [ 0 23  4]
 [ 0  1 24]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        23
Iris-versicolor       0.96      0.85      0.90        27
 Iris-virginica       0.86      0.96      0.91        25

       accuracy                           0.93        75
      macro avg       0.94      0.94      0.94        75
   weighted avg       0.94      0.93      0.93        75

--------------------------------------------------
```

```
# IRIS PLANT DATASET
# Random Forest Classifier(With Tuning)[40-60 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)
```

```
col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

##############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

##############################################################################
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth
```

```
                'max_depth': max_depth,
                'min_samples_split': min_samples_split,
                'min_samples_leaf': min_samples_leaf,
                'bootstrap': bootstrap}
pprint(random_grid)

################################################################################

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
Fitting 3 folds for each of 100 candidates, totalling 300 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:   44.2s
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed:  3.0min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:  5.9min finished
Confusion Matrix:
[[28  0  0]
 [ 0 28  4]
 [ 0  1 29]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        28
Iris-versicolor       0.97      0.88      0.92        32
 Iris-virginica       0.88      0.97      0.92        30
```

```
# IRIS PLANT DATASET
# Random Forest Classifier(With Tuning)[30-70 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']


df.columns = col_name
```

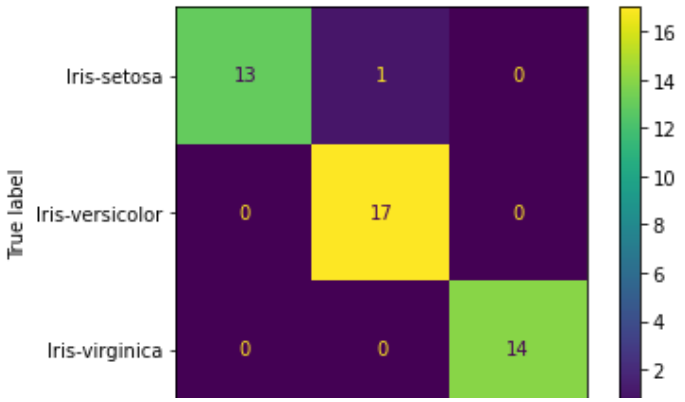
```
X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

##############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

##############################################################################
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)
```

```
##############################################################################

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------")
print("-------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------")
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
      Parameters currently in use:

      {'bootstrap': True,
       'ccp_alpha': 0.0,
       'class_weight': None,
       'criterion': 'gini',
       'max_depth': None,
       'max_features': 'auto',
       'max_leaf_nodes': None,
       'max_samples': None,
       'min_impurity_decrease': 0.0,
       'min_impurity_split': None,
       'min_samples_leaf': 1,
       'min_samples_split': 2,
       'min_weight_fraction_leaf': 0.0,
       'n_estimators': 100,
       'n_jobs': None,
       'oob_score': False,
       'random_state': None,
       'verbose': 0,
       'warm_start': False}
      {'bootstrap': [True, False],
       'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
       'max_features': ['auto', 'sqrt'],
       'min_samples_leaf': [1, 2, 4],
       'min_samples_split': [2, 5, 10],
       'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
      Fitting 3 folds for each of 100 candidates, totalling 300 fits
      [Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
      [Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:   43.5s
      [Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed:  3.0min
      [Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:  5.9min finished
      Confusion Matrix:
      [[33  0  0]
       [ 0 33  3]
       [ 0  3 33]]
      --------------------------------------------------
      --------------------------------------------------
      Performance Evaluation

##################################################################################################
         Iris-setosa       1.00      1.00      1.00        33

# IRIS PLANT DATASET
# Multi Layer Perceptron(Without Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']


df.columns = col_name


X = df.drop(['Class'], axis=1)
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.30,rand


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[13  1  0]
 [ 0 17  0]
 [ 0  0 14]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
                precision    recall  f1-score   support

    Iris-setosa       1.00      0.93      0.96        14
Iris-versicolor       0.94      1.00      0.97        17
 Iris-virginica       1.00      1.00      1.00        14

       accuracy                           0.98        45
      macro avg       0.98      0.98      0.98        45
   weighted avg       0.98      0.98      0.98        45

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9777777777777777
```




```
# IRIS PLANT DATASET
# Multi Layer Perceptron(Without Tuning)[60-40 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,rando
```

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[17  1  0]
 [ 0 20  4]
 [ 0  0 18]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
                 precision    recall  f1-score   support

    Iris-setosa       1.00      0.94      0.97        18
Iris-versicolor       0.95      0.83      0.89        24
 Iris-virginica       0.82      1.00      0.90        18

       accuracy                           0.92        60
      macro avg       0.92      0.93      0.92        60
   weighted avg       0.93      0.92      0.92        60

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9166666666666666
```


```
# IRIS PLANT DATASET
# Multi Layer Perceptron(Without Tuning)[50-50 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
```

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

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[22  1  0]
 [ 0 23  4]
 [ 0  0 25]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
                 precision    recall  f1-score   support

    Iris-setosa       1.00      0.96      0.98        23
Iris-versicolor       0.96      0.85      0.90        27
 Iris-virginica       0.86      1.00      0.93        25

       accuracy                           0.93        75
      macro avg       0.94      0.94      0.94        75
   weighted avg       0.94      0.93      0.93        75

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9333333333333333
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
```

```
# IRIS PLANT DATASET
# Multi Layer Perceptron(Without Tuning)[40-60 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,rando



# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification using MLP
from sklearn.neural_network import MLPClassifier
```

```
classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[28  0  0]
 [ 0 28  4]
 [ 0  0 30]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        28
Iris-versicolor       1.00      0.88      0.93        32
 Iris-virginica       0.88      1.00      0.94        30

       accuracy                           0.96        90
      macro avg       0.96      0.96      0.96        90
   weighted avg       0.96      0.96      0.96        90
```

```
# IRIS PLANT DATASET
# Multi Layer Perceptron(Without Tuning)[30-70 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)
```

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

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[33  0  0]
 [ 0 31  5]
 [ 0  1 35]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        33
```

```
# IRIS PLANT DATASET
# Multi Layer Perceptron(With Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

##############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

##############################################################################
# Creating a set of important sample features
```

```
parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

################################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------")
print("-------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------")
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
{'activation': ['tanh', 'relu'],
 'alpha': [0.0001, 0.05],
 'hidden_layer_sizes': [(50, 50, 50), (50, 100, 50), (100,)],
 'learning_rate': ['constant', 'adaptive'],
 'solver': ['sgd', 'adam']}
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[14  0  0]
 [ 0 17  0]
 [ 0  0 14]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        14
Iris-versicolor       1.00      1.00      1.00        17
 Iris-virginica       1.00      1.00      1.00        14

       accuracy                           1.00        45
      macro avg       1.00      1.00      1.00        45
   weighted avg       1.00      1.00      1.00        45

--------------------------------------------------
--------------------------------------------------
Accuracy:
1.0
```

```
# IRIS PLANT DATASET
# Multi Layer Perceptron(With Tuning)[60-40 split]


import pandas as pd
import numpy as np
```

```
# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

###############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

###############################################################################
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

###############################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
```

```
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
{'activation': ['tanh', 'relu'],
 'alpha': [0.0001, 0.05],
 'hidden_layer_sizes': [(50, 50, 50), (50, 100, 50), (100,)],
 'learning_rate': ['constant', 'adaptive'],
 'solver': ['sgd', 'adam']}
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[18  0  0]
```

```
# IRIS PLANT DATASET
# Multi Layer Perceptron(With Tuning)[50-50 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,rando
```

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

###############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

###############################################################################
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

###############################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------")
print("-------------------------------------------------")
```

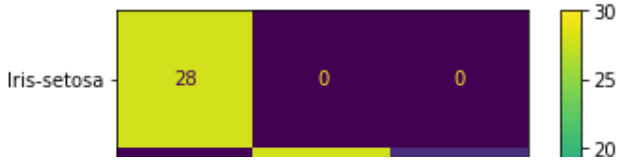
```
print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------")
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
```

```
# IRIS PLANT DATASET
# Multi Layer Perceptron(With Tuning)[40-60 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

##############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
```

```
print('Parameters currently in use:\n')
pprint(classifier.get_params())

################################################################################
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

################################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------")
print("-------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------")
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
{'activation': ['tanh', 'relu'],
 'alpha': [0.0001, 0.05],
 'hidden_layer_sizes': [(50, 50, 50), (50, 100, 50), (100,)],
 'learning_rate': ['constant', 'adaptive'],
 'solver': ['sgd', 'adam']}
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[28  0  0]
 [ 0 28  4]
 [ 0  0 30]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        28
Iris-versicolor       1.00      0.88      0.93        32
 Iris-virginica       0.88      1.00      0.94        30

       accuracy                           0.96        90
      macro avg       0.96      0.96      0.96        90
   weighted avg       0.96      0.96      0.96        90

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9555555555555556
```




```
# IRIS PLANT DATASET
# Multi Layer Perceptron(With Tuning)[30-70 split]
```

```
import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

##############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

##############################################################################
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

##############################################################################
```

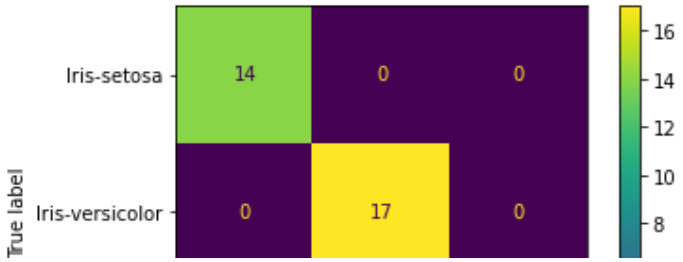
```
from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
{'activation': ['tanh', 'relu'],
 'alpha': [0.0001, 0.05],
 'hidden_layer_sizes': [(50, 50, 50), (50, 100, 50), (100,)],
 'learning_rate': ['constant', 'adaptive'],
 'solver': ['sgd', 'adam']}
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[33  0  0]
 [ 0 32  4]
 [ 0  1 35]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
                  precision    recall  f1-score   support
```

```
# IRIS PLANT DATASET
# SVM(Without Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[14  0  0]
 [ 0 17  0]
 [ 0  0 14]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        14
Iris-versicolor       1.00      1.00      1.00        17
 Iris-virginica       1.00      1.00      1.00        14

       accuracy                           1.00        45
      macro avg       1.00      1.00      1.00        45
   weighted avg       1.00      1.00      1.00        45

--------------------------------------------------
--------------------------------------------------
Accuracy:
1.0
```




```
# IRIS PLANT DATASET
# SVM(Without Tuning)[60-40 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
```

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

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[18  0  0]
 [ 0 23  1]
 [ 0  0 18]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        18
Iris-versicolor       1.00      0.96      0.98        24
 Iris-virginica       0.95      1.00      0.97        18

       accuracy                           0.98        60
      macro avg       0.98      0.99      0.98        60
   weighted avg       0.98      0.98      0.98        60

--------------------------------------------------
--------------------------------------------------
```

```
# IRIS PLANT DATASET
# SVM(Without Tuning)[50-50 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)
```

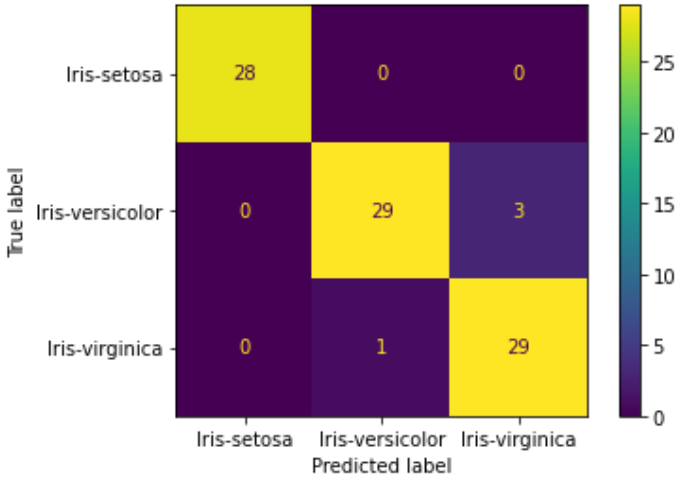
```
y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[23  0  0]
 [ 0 23  4]
 [ 0  1 24]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
                precision    recall  f1-score   support
```

```
# IRIS PLANT DATASET
# SVM(Without Tuning)[40-60 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")
```

```
print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------")
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

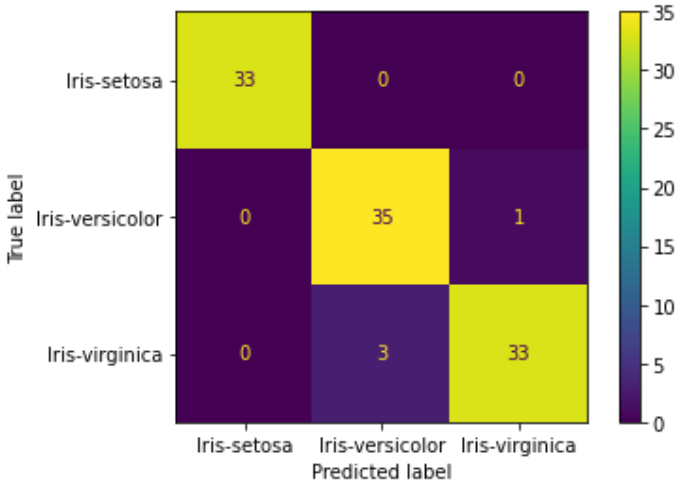
```
Confusion Matrix:
[[28  0  0]
 [ 0 29  3]
 [ 0  1 29]]
-------------------------------------------------
-------------------------------------------------
Performance Evaluation
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        28
Iris-versicolor       0.97      0.91      0.94        32
 Iris-virginica       0.91      0.97      0.94        30

       accuracy                           0.96        90
      macro avg       0.96      0.96      0.96        90
   weighted avg       0.96      0.96      0.96        90

-------------------------------------------------
-------------------------------------------------
Accuracy:
0.9555555555555556
```



```
# IRIS PLANT DATASET
# SVM(Without Tuning)[30-70 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")
```

```
print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[33  0  0]
 [ 0 35  1]
 [ 0  3 33]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        33
Iris-versicolor       0.92      0.97      0.95        36
 Iris-virginica       0.97      0.92      0.94        36

       accuracy                           0.96       105
      macro avg       0.96      0.96      0.96       105
   weighted avg       0.96      0.96      0.96       105

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9619047619047619
```




```
# IRIS PLANT DATASET
# SVM(With Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)
```

```
col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Classification
from sklearn.svm import SVC

classifier = SVC()

########################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

########################################################################
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly',

pprint(param_grid)

########################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)
```

```
y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
{'C': [0.1, 1, 10, 100],
 'gamma': [1, 0.1, 0.01, 0.001],
 'kernel': ['rbf', 'poly', 'sigmoid']}
Fitting 5 folds for each of 48 candidates, totalling 240 fits
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ....................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ....................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ....................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
```

```
# IRIS PLANT DATASET
# SVM(With Tuning)[60-40 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,rando



# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)


# Classification
from sklearn.svm import SVC

classifier = SVC()

###############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

###############################################################################
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly',

pprint(param_grid)

###############################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
```

```
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
```

```
# IRIS PLANT DATASET
# SVM(With Tuning)[50-50 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Classification
from sklearn.svm import SVC

classifier = SVC()

###############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

###############################################################################
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly',
```

```
param_grid = { 'C': [0.1, 1, 10, 100], 'gamma': [1,0.1,0.01,0.001], 'kernel': ['rbf', 'poly']}

pprint(param_grid)

##############################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----------------------------------------------------")
print("-----------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-----------------------------------------------------")
print("-----------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
{'C': [0.1, 1, 10, 100],
 'gamma': [1, 0.1, 0.01, 0.001],
 'kernel': ['rbf', 'poly', 'sigmoid']}
Fitting 5 folds for each of 48 candidates, totalling 240 fits
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .................................
[CV] .................. C=0.1, gamma=1, kernel=sigmoid, total=   0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .................................
[CV] .................. C=0.1, gamma=1, kernel=sigmoid, total=   0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid
```

```
# IRIS PLANT DATASET
# SVM(With Tuning)[40-60 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']


df.columns = col_name
```

```
df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Classification
from sklearn.svm import SVC

classifier = SVC()

##############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

##############################################################################
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly',

pprint(param_grid)

##############################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

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

```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
{'C': [0.1, 1, 10, 100],
 'gamma': [1, 0.1, 0.01, 0.001],
 'kernel': ['rbf', 'poly', 'sigmoid']}
Fitting 5 folds for each of 48 candidates, totalling 240 fits
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ..... C=0.1, gamma=1, kernel=rbf, total=   0.0s
```

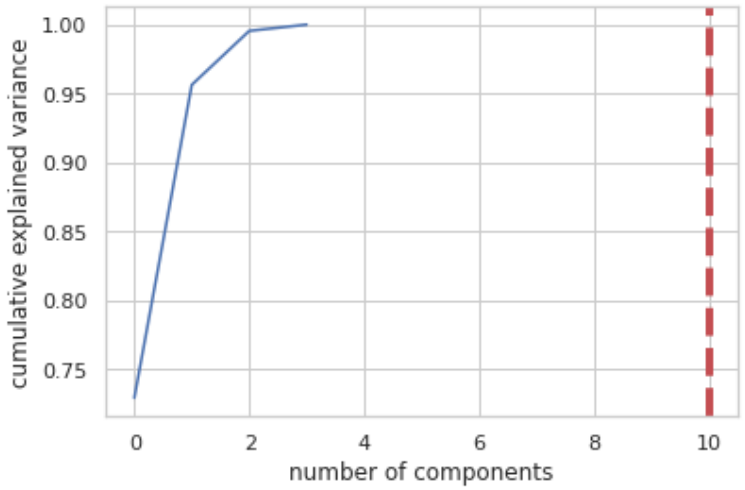
```
# IRIS PLANT DATASET
# SVM(With Tuning)[30-70 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,rando



# Feature Scaling
from sklearn.preprocessing import StandardScaler

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



# Classification
```

```
from sklearn.svm import SVC

classifier = SVC()

################################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

################################################################################
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly',

pprint(param_grid)

################################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
```

```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
     Parameters currently in use:

##################################################################################################
       break_ties : false,

# IRIS PLANT DATASET
# Random Forest Classifier(Without Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']

df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=4)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters

pca = PCA(n_components=2)
pca.fit(X_train)
```

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


# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

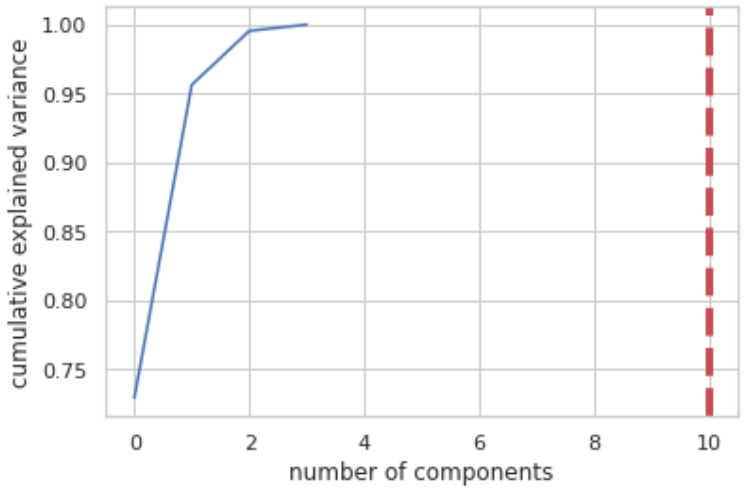



```
None
Confusion Matrix:
[[14  0  0]
 [ 0 14  3]
 [ 0  0 14]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        14
Iris-versicolor       1.00      0.82      0.90        17
 Iris-virginica       0.82      1.00      0.90        14

       accuracy                           0.93        45
      macro avg       0.94      0.94      0.94        45
   weighted avg       0.95      0.93      0.93        45

--------------------------------------------------
--------------------------------------------------
Accuracy:
```

```
# IRIS PLANT DATASET
# Random Forest Classifier(With Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.30,rand


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=4)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters

pca = PCA(n_components=2)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)


# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

##############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

##############################################################################
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
```

```
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)

################################################################################

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

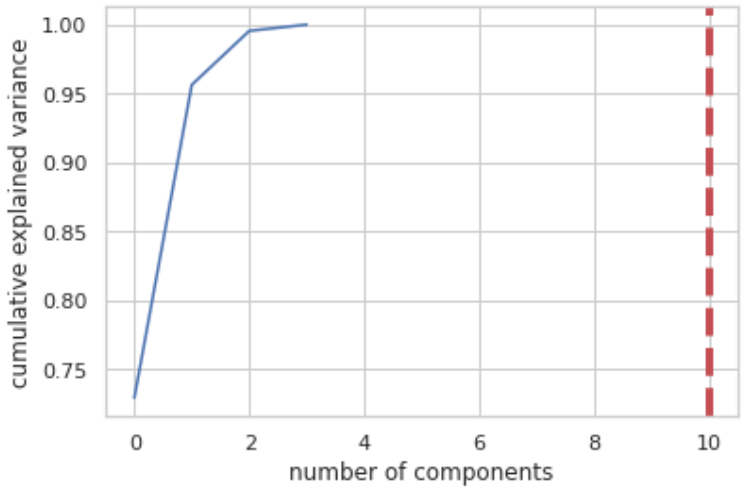



```
None
Parameters currently in use:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
Fitting 3 folds for each of 100 candidates, totalling 300 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:   44.7s
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed:  3.0min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:  5.8min finished
Confusion Matrix:
[[14  0  0]
 [ 0 14  3]
 [ 0  0 14]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        14
```

```
# IRIS PLANT DATASET
# Multi Layer Perceptron(Without Tuning)[70-30 split]
```

```
import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.30,rand


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=4)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters

pca = PCA(n_components=2)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)


# Classification using MLP
from sklearn.neural_network import MLPClassifier
```

```
classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

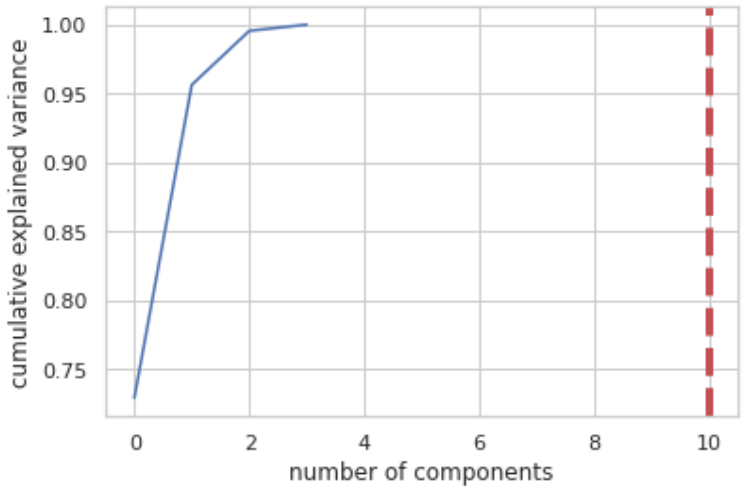



```
None
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[13  1  0]
 [ 0 14  3]
 [ 0  0 14]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
                precision    recall  f1-score   support

    Iris-setosa       1.00      0.93      0.96        14
Iris-versicolor       0.93      0.82      0.87        17
```

```
# IRIS PLANT DATASET
# Multi Layer Perceptron(With Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
```

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


# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=4)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters

pca = PCA(n_components=2)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)


# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

###############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

###############################################################################
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

###############################################################################

from sklearn.model_selection import GridSearchCV
```

```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------------")
print("-------------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------------")
print("-------------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```




```
None
Parameters currently in use:

{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'solver': 'adam',
```

```
# IRIS PLANT DATASET
# SVM(Without Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split
```

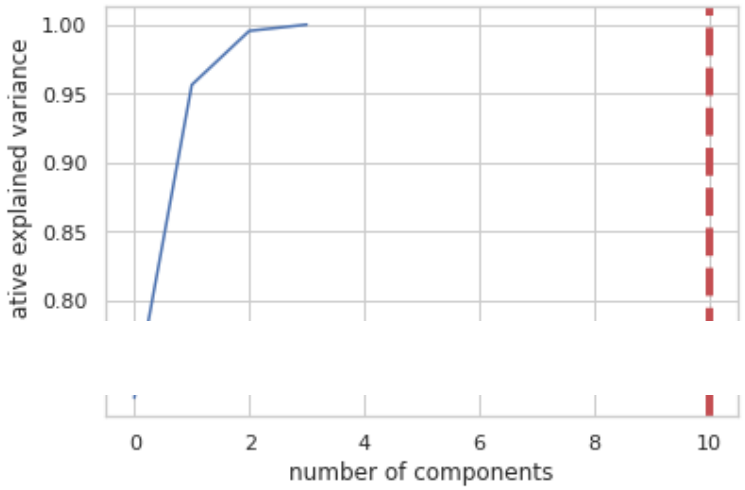
```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=4)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters

pca = PCA(n_components=2)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)


# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))
```

```
print("-------------------------------------------------")
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```


```
# IRIS PLANT DATASET
# SVM(With Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=4)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters
```

```
pca = PCA(n_components=2)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)



# Classification
from sklearn.svm import SVC

classifier = SVC()

#########################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#########################################################################
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly',

pprint(param_grid)

#########################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------")
print("-------------------------------------------------")



print("Performance Evaluation")
print(classification_report(y_test, y_pred))
```

```
    print("-------------------------------------------------")
    print("-------------------------------------------------")

    print("Accuracy:")
    print(accuracy_score(y_test, y_pred))


    import matplotlib.pyplot as plt
    from sklearn.metrics import plot_confusion_matrix
    plot_confusion_matrix(rf_random, X_test, y_test)
    plt.show()
```

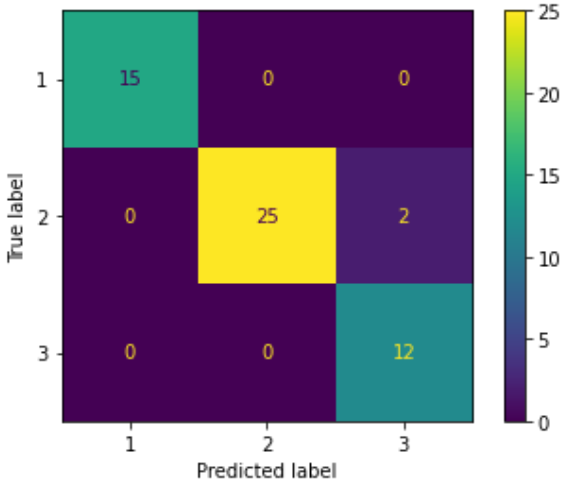



```
None
Parameters currently in use:

{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
{'C': [0.1, 1, 10, 100],
 'gamma': [1, 0.1, 0.01, 0.001],
 'kernel': ['rbf', 'poly', 'sigmoid']}
```

```
# WINE DATASET
# Random Forest Classifier(Without Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
```

```
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

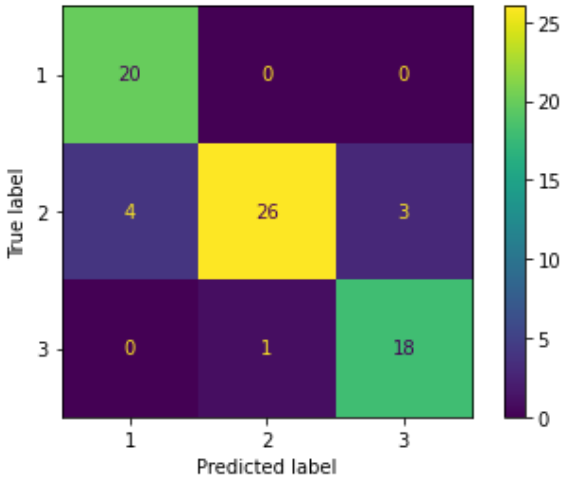
```
Confusion Matrix:
[[15  0  0]
 [ 0 25  2]
 [ 0  0 12]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        15
           2       1.00      0.93      0.96        27
           3       0.86      1.00      0.92        12

    accuracy                           0.96        54
   macro avg       0.95      0.98      0.96        54
weighted avg       0.97      0.96      0.96        54

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9629629629629629
```




```
# WINE DATASET
# Random Forest Classifier(Without Tuning)[60-40 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class' 'Alcohol' 'Malic acid' 'Ash' 'Alcalinity of ash' 'Magnesium' 'Total ph
```

```
col_name = ['Class', 'Alcohol', 'Malic acid', 'Ash', 'Alcalinity of ash', 'Magnesium', 'Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------")
print("-------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------")
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

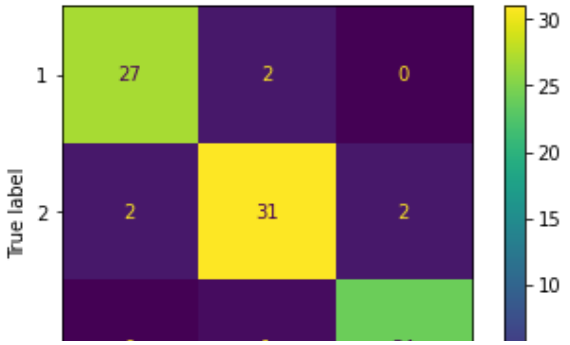
```
Confusion Matrix:
[[20  0  0]
 [ 4 26  3]
 [ 0  1 18]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.83      1.00      0.91        20
           2       0.96      0.79      0.87        33
           3       0.86      0.95      0.90        19

    accuracy                           0.89        72
   macro avg       0.88      0.91      0.89        72
weighted avg       0.90      0.89      0.89        72

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.8888888888888888
```




```
# WINE DATASET
# Random Forest Classifier(Without Tuning)[50-50 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[27  2  0]
 [ 2 31  2]
 [ 0  1 24]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.93      0.93      0.93        29
           2       0.91      0.89      0.90        35
           3       0.92      0.96      0.94        25

    accuracy                           0.92        89
   macro avg       0.92      0.93      0.92        89
weighted avg       0.92      0.92      0.92        89

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9213483146067416
```




```
# WINE DATASET
# Random Forest Classifier(Without Tuning)[40-60 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,rando


# Feature Scaling
```

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[30  4  0]
 [ 6 34  2]
 [ 0  1 30]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.83      0.88      0.86        34
           2       0.87      0.81      0.84        42
           3       0.94      0.97      0.95        31

    accuracy                           0.88       107
   macro avg       0.88      0.89      0.88       107
weighted avg       0.88      0.88      0.88       107

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.8785046728971962
```


```
# WINE DATASET
# Random Forest Classifier(Without Tuning)[30-70 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
```

```
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[35  6  0]
 [ 1 47  1]
 [ 0  0 35]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.97      0.85      0.91        41
           2       0.89      0.96      0.92        49
           3       0.97      1.00      0.99        35

    accuracy                           0.94       125
   macro avg       0.94      0.94      0.94       125
```

```
# WINE DATASET
# Random Forest Classifier(With Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.30,rand


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

################################################################################
# Showing all the parameters

from pprint import pprint
```

```
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

################################################################################
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)

################################################################################

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------")
print("-------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))
```

```
print("-------------------------------------------------")
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0
```

```
# WINE DATASET
# Random Forest Classifier(With Tuning)[60-40 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

################################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())
```

```
##############################################################################
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)

##############################################################################

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")
```

```
print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
```

```
# WINE DATASET
# Random Forest Classifier(With Tuning)[50-50 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

################################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

################################################################################
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
```

```
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)

################################################################################

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
     Parameters currently in use:

     {'bootstrap': True,
```

```
# WINE DATASET
# Random Forest Classifier(With Tuning)[40-60 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

##############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

##############################################################################
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
```

```
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)

################################################################################

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
```

```
    plot_confusion_matrix(rf_random, X_test, y_test)
    plt.show()
```

```
# WINE DATASET
# Random Forest Classifier(With Tuning)[30-70 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.,random


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

################################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

################################################################################
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
```

```
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)

################################################################################

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

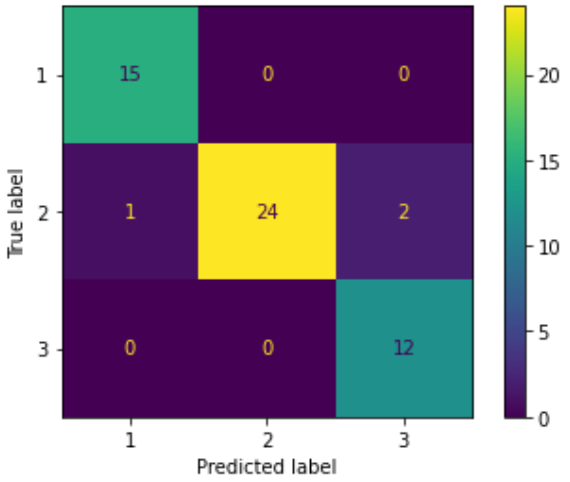
```
Parameters currently in use:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
Fitting 3 folds for each of 100 candidates, totalling 300 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:   51.3s
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed:  3.5min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:  6.8min finished
Confusion Matrix:
[[35  6  0]
 [ 1 46  2]
 [ 0  1 34]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.97      0.85      0.91        41
           2       0.87      0.94      0.90        49
           3       0.94      0.97      0.96        35

    accuracy                           0.92       125
   macro avg       0.93      0.92      0.92       125
weighted avg       0.92      0.92      0.92       125

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.92
```


```
##################################################################################################
```

```
# WINE DATASET
# Multi Layer Perceptron(Without Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.30,rand


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------")
print("-------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------")
```

```
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[15  0  0]
 [ 1 24  2]
 [ 0  0 12]]
-------------------------------------------------
-------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.94      1.00      0.97        15
           2       1.00      0.89      0.94        27
           3       0.86      1.00      0.92        12

    accuracy                           0.94        54
   macro avg       0.93      0.96      0.94        54
weighted avg       0.95      0.94      0.94        54

-------------------------------------------------
-------------------------------------------------
Accuracy:
0.9444444444444444
```

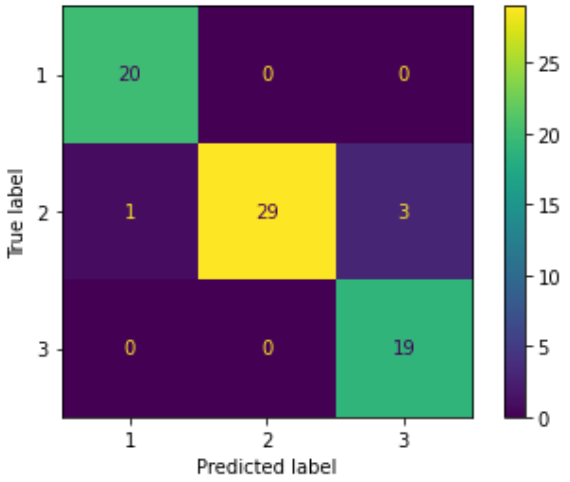



```
# WINE DATASET
# Multi Layer Perceptron(Without Tuning)[60-40 split]
```

```
import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

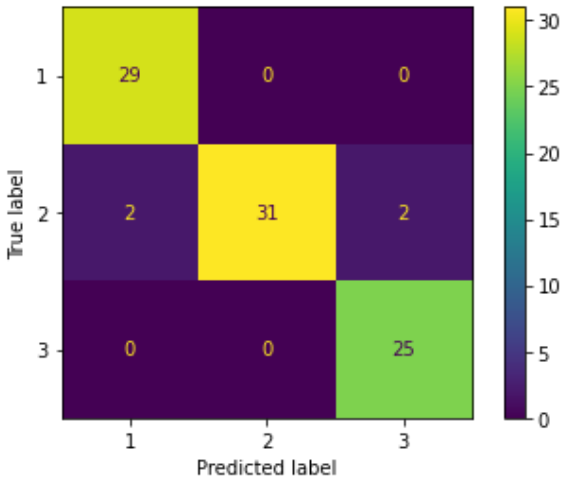
```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[20  0  0]
 [ 1 29  3]
 [ 0  0 19]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.95      1.00      0.98        20
           2       1.00      0.88      0.94        33
           3       0.86      1.00      0.93        19

    accuracy                           0.94        72
   macro avg       0.94      0.96      0.95        72
weighted avg       0.95      0.94      0.94        72

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9444444444444444
```




```
# WINE DATASET
# Multi Layer Perceptron(Without Tuning)[50-50 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)
```

```
col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

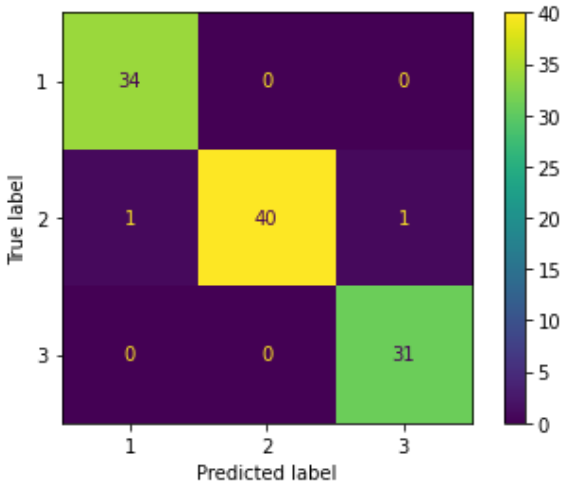
```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[29  0  0]
 [ 2 31  2]
 [ 0  0 25]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.94      1.00      0.97        29
           2       1.00      0.89      0.94        35
           3       0.93      1.00      0.96        25

    accuracy                           0.96        89
   macro avg       0.95      0.96      0.96        89
weighted avg       0.96      0.96      0.95        89

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9550561797752809
```




```
# WINE DATASET
# Multi Layer Perceptron(Without Tuning)[40-60 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315
```

```
df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------")
print("-------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------")
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

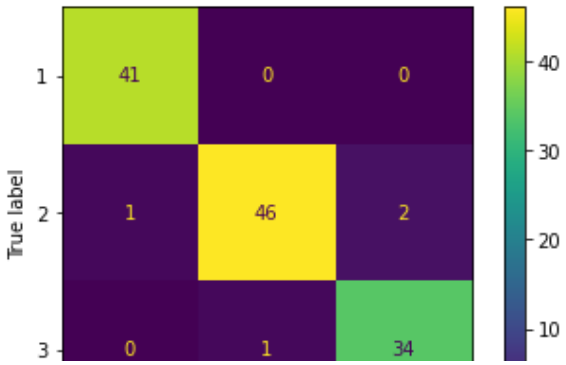
```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[34  0  0]
 [ 1 40  1]
 [ 0  0 31]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.97      1.00      0.99        34
           2       1.00      0.95      0.98        42
           3       0.97      1.00      0.98        31

    accuracy                           0.98       107
   macro avg       0.98      0.98      0.98       107
weighted avg       0.98      0.98      0.98       107

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9813084112149533
```




```
# WINE DATASET
# Multi Layer Perceptron(Without Tuning)[30-70 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name


X = df.drop(['Class'], axis=1)
```

```
y = df[ 'Class' ]

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[41  0  0]
 [ 1 46  2]
 [ 0  1 34]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.98      1.00      0.99        41
           2       0.98      0.94      0.96        49
           3       0.94      0.97      0.96        35

    accuracy                           0.97       125
   macro avg       0.97      0.97      0.97       125
weighted avg       0.97      0.97      0.97       125

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.968
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
```




```
# WINE DATASET
# Multi Layer Perceptron(With Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando
```

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

###############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

###############################################################################
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

###############################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------")
print("-------------------------------------------------")
```

```
print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
```

```
# WINE DATASET
# Multi Layer Perceptron(With Tuning)[60-40 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

##############################################################################
# Showing all the parameters
```

```
from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

################################################################################
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

################################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----------------------------------------------------")
print("-----------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-----------------------------------------------------")
print("-----------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
```

```
    plt.show()
```

```
# WINE DATASET
# Multi Layer Perceptron(With Tuning)[50-50 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

##############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

##############################################################################
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)
```

```
################################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------------")
print("-------------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------------")
print("-------------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
{'activation': ['tanh', 'relu'],
 'alpha': [0.0001, 0.05],
 'hidden_layer_sizes': [(50, 50, 50), (50, 100, 50), (100,)],
 'learning_rate': ['constant', 'adaptive'],
 'solver': ['sgd', 'adam']}
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[27  2  0]
 [ 3 29  3]
 [ 0  0 25]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.90      0.93      0.92        29
           2       0.94      0.83      0.88        35
```

```
# WINE DATASET
# Multi Layer Perceptron(With Tuning)[40-60 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

###############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

###############################################################################
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

###############################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)
```

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

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
```

```
# WINE DATASET
# Multi Layer Perceptron(With Tuning)[30-70 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

```
# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

###############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

###############################################################################
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

###############################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----------------------------------------------------")
print("-----------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-----------------------------------------------------")
print("-----------------------------------------------------")
```

```
print("Accuracy:")
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
      Parameters currently in use:

      {'activation': 'relu',
       'alpha': 0.0001,
       'batch_size': 'auto',
       'beta_1': 0.9,
       'beta_2': 0.999,
       'early_stopping': False,
```

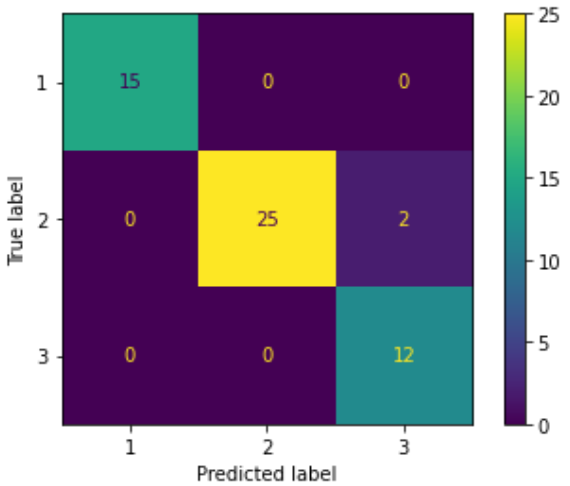
```
#############################################################################################
```

```
       'learning_rate': 'constant',
```

```
# WINE DATASET
# SVM(Without Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
print("-------------------------------------------------")
print("-------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------")
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

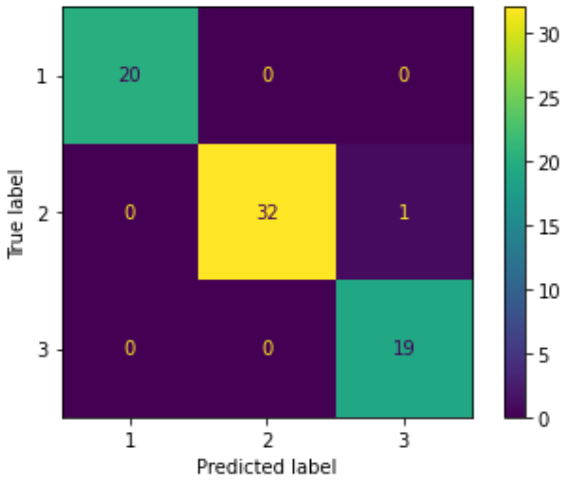
```
Confusion Matrix:
[[15  0  0]
 [ 0 25  2]
 [ 0  0 12]]
-------------------------------------------------
-------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        15
           2       1.00      0.93      0.96        27
           3       0.86      1.00      0.92        12

    accuracy                           0.96        54
   macro avg       0.95      0.98      0.96        54
weighted avg       0.97      0.96      0.96        54

-------------------------------------------------
-------------------------------------------------
Accuracy:
0.9629629629629629
```



```
# WINE DATASET
# SVM(Without Tuning)[60-40 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------")
print("-------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))
```

```
print("-------------------------------------------------")
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[20  0  0]
 [ 0 32  1]
 [ 0  0 19]]
-------------------------------------------------
-------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        20
           2       1.00      0.97      0.98        33
           3       0.95      1.00      0.97        19

    accuracy                           0.99        72
   macro avg       0.98      0.99      0.99        72
weighted avg       0.99      0.99      0.99        72

-------------------------------------------------
-------------------------------------------------
Accuracy:
0.9861111111111112
```

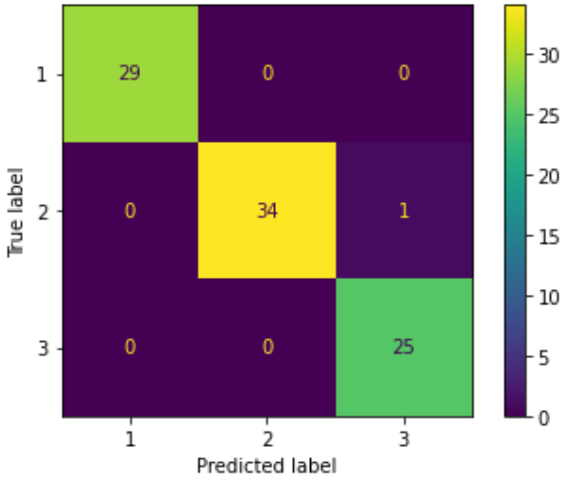



```
# WINE DATASET
# SVM(Without Tuning)[50-50 split]


import pandas as pd
import numpy as np
```

```
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
```

```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

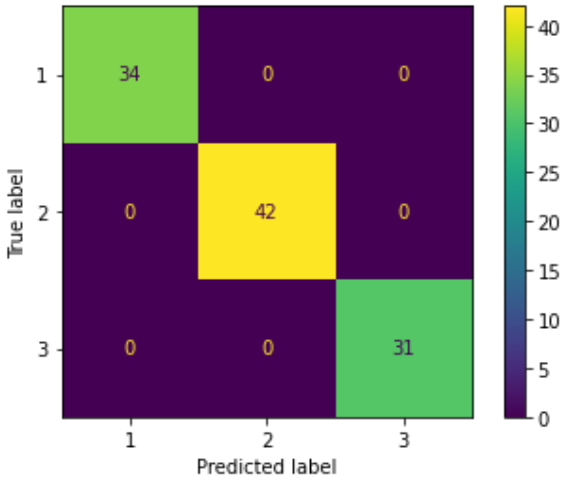
```
Confusion Matrix:
[[29  0  0]
 [ 0 34  1]
 [ 0  0 25]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        29
           2       1.00      0.97      0.99        35
           3       0.96      1.00      0.98        25

    accuracy                           0.99        89
   macro avg       0.99      0.99      0.99        89
weighted avg       0.99      0.99      0.99        89

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9887640449438202
```




```
# WINE DATASET
# SVM(Without Tuning)[40-60 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315
```

```
df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------")
print("-------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------")
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[34  0  0]
 [ 0 42  0]
 [ 0  0 31]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        34
           2       1.00      1.00      1.00        42
           3       1.00      1.00      1.00        31

    accuracy                           1.00       107
   macro avg       1.00      1.00      1.00       107
weighted avg       1.00      1.00      1.00       107

--------------------------------------------------
--------------------------------------------------
Accuracy:
1.0
```




```
# WINE DATASET
# SVM(Without Tuning)[30-70 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
Confusion Matrix:
[[38  3  0]
 [ 0 49  0]
 [ 0  2 33]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       1.00      0.93      0.96        41
           2       0.91      1.00      0.95        49
           3       1.00      0.94      0.97        35

    accuracy                           0.96       125
   macro avg       0.97      0.96      0.96       125
weighted avg       0.96      0.96      0.96       125

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.96
```


```
# WINE DATASET
# SVM(With Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)


# Classification
from sklearn.svm import SVC

classifier = SVC()

################################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

################################################################################
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly',

pprint(param_grid)

################################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
```

```
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
     Parameters currently in use:

     {'C': 1.0,
      'break_ties': False,
      'cache_size': 200,
      'class_weight': None,
```

```
# WINE DATASET
# SVM(With Tuning)[60-40 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Classification
from sklearn.svm import SVC

classifier = SVC()

#############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#############################################################################
# Creating a set of important sample features
```

```
param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly',

pprint(param_grid)

##############################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
{'C': [0.1, 1, 10, 100],
 'gamma': [1, 0.1, 0.01, 0.001],
 'kernel': ['rbf', 'poly', 'sigmoid']}
Fitting 5 folds for each of 48 candidates, totalling 240 fits
[CV] C=0.1, gamma=1, kernel=rbf ......................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf ......................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf ......................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf ......................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf ......................................
[CV] ...................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly .....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly .....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly .....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly .....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly .....................................
[CV] ..................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid ..................................
[CV] .................. C=0.1, gamma=1, kernel=sigmoid, total=   0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid ..................................
[CV] .................. C=0.1, gamma=1, kernel=sigmoid, total=   0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid ..................................
[CV] .................. C=0.1, gamma=1, kernel=sigmoid, total=   0.0s
```

```
# WINE DATASET
# SVM(With Tuning)[50-50 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315
```

```
df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Classification
from sklearn.svm import SVC

classifier = SVC()

###############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

###############################################################################
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly',

pprint(param_grid)

###############################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)
```

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

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
Parameters currently in use:

{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
{'C': [0.1, 1, 10, 100],
 'gamma': [1, 0.1, 0.01, 0.001],
 'kernel': ['rbf', 'poly', 'sigmoid']}
Fitting 5 folds for each of 48 candidates, totalling 240 fits
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ....................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ....................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
```

```
# WINE DATASET
# SVM(With Tuning)[40-60 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

```
# Classification
from sklearn.svm import SVC

classifier = SVC()

################################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

################################################################################
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly',

pprint(param_grid)

################################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

```
     Parameters currently in use:

     {'C': 1.0,
      'break_ties': False,
      'cache_size': 200,
```

```
# WINE DATASET
# SVM(With Tuning)[30-70 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Classification
from sklearn.svm import SVC

classifier = SVC()

#############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#############################################################################
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly',
```

```
pprint(param_grid)

################################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

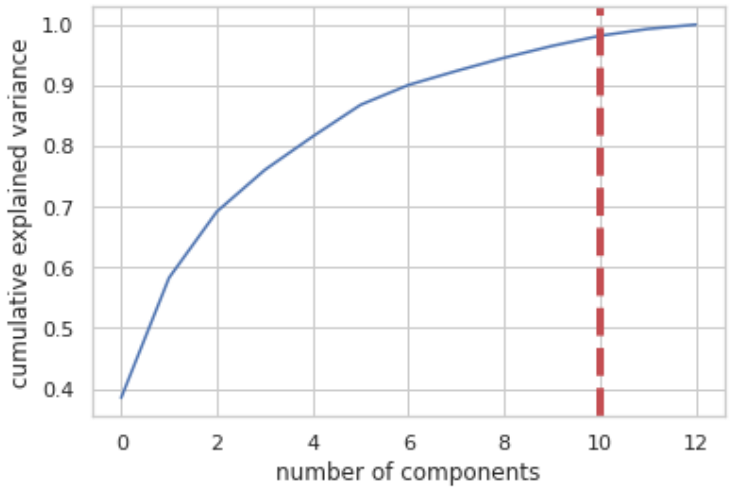
```
Parameters currently in use:

{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
{'C': [0.1, 1, 10, 100],
 'gamma': [1, 0.1, 0.01, 0.001],
 'kernel': ['rbf', 'poly', 'sigmoid']}
Fitting 5 folds for each of 48 candidates, totalling 240 fits
[CV] C=0.1, gamma=1, kernel=rbf ....................................
[CV] ..................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf ....................................
[CV] ..................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf ....................................
[CV] ..................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf ....................................
[CV] ..................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf ....................................
[CV] ..................... C=0.1, gamma=1, kernel=rbf, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ...................................
[CV] .................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ...................................
[CV] .................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ...................................
[CV] .................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ...................................
[CV] .................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=poly ...................................
[CV] .................... C=0.1, gamma=1, kernel=poly, total=   0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid ................................
[CV] ................. C=0.1, gamma=1, kernel=sigmoid, total=   0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid ................................
[CV] ................. C=0.1, gamma=1, kernel=sigmoid, total=   0.0s
```

```
##########################################################################################
```

```
# WINE DATASET
# Random Forest Classifier(Without Tuning)[70-30 split]



import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
```

```
              'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315

    df.columns = col_name


    X = df.drop(['Class'], axis=1)
    y = df['Class']

    from sklearn.model_selection import train_test_split

    X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


    # Feature Scaling
    from sklearn.preprocessing import StandardScaler

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



    # Finding the important parameters that contribute to most of the variance in the data.

    import matplotlib.pyplot as plt
    import seaborn as sns
    from sklearn.decomposition import PCA

    pca_test = PCA(n_components=13)
    pca_test.fit(X_train)
    sns.set(style='whitegrid')
    plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
    plt.xlabel('number of components')
    plt.ylabel('cumulative explained variance')
    plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
    display(plt.show())

    # So we can see that we have 10 important parameters

    pca = PCA(n_components=10)
    pca.fit(X_train)
    X_train = pca.transform(X_train)
    X_test = pca.transform(X_test)



    # Classification
    from sklearn.ensemble import RandomForestClassifier

    classifier = RandomForestClassifier()
    classifier.fit(X_train, y_train)

    y_pred = classifier.predict(X_test)
```

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

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

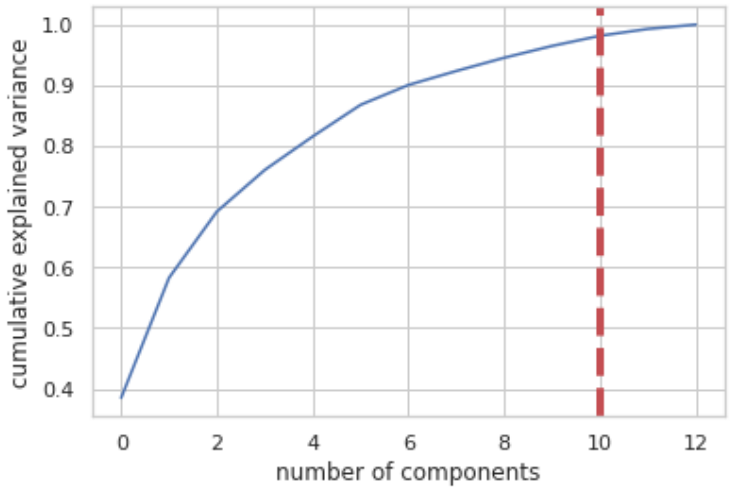



```
None
Confusion Matrix:
[[15  0  0]
 [ 2 23  2]
 [ 0  0 12]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
```

```
# WINE DATASET
# Random Forest Classifier(With Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.30,rand


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

```
# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=13)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters

pca = PCA(n_components=10)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

##############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

##############################################################################
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
```

```
                'max_depth': max_depth,
                'min_samples_split': min_samples_split,
                'min_samples_leaf': min_samples_leaf,
                'bootstrap': bootstrap}
pprint(random_grid)

################################################################################

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------")
print("-------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("-------------------------------------------------")
print("-------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

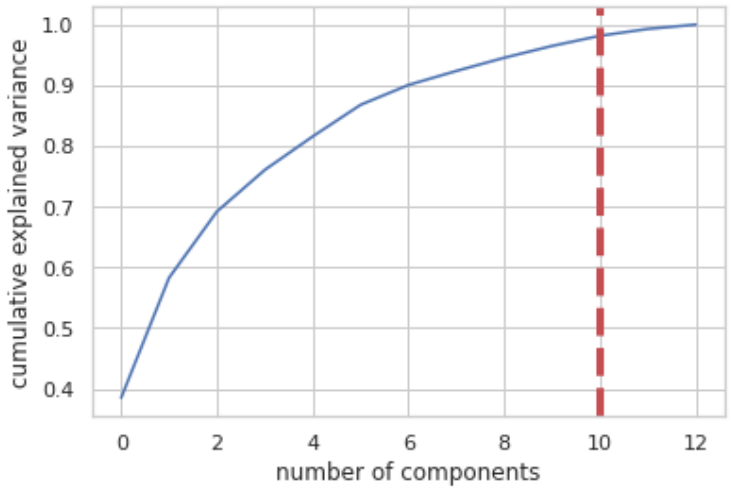



```
None
Parameters currently in use:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
Fitting 3 folds for each of 100 candidates, totalling 300 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers
```

```
# WINE DATASET
# Multi Layer Perceptron(Without Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315
```

```
df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.30,rand


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=13)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters

pca = PCA(n_components=10)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
```

```
print("                                                    ")
print("----------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("----------------------------------------------------")
print("----------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

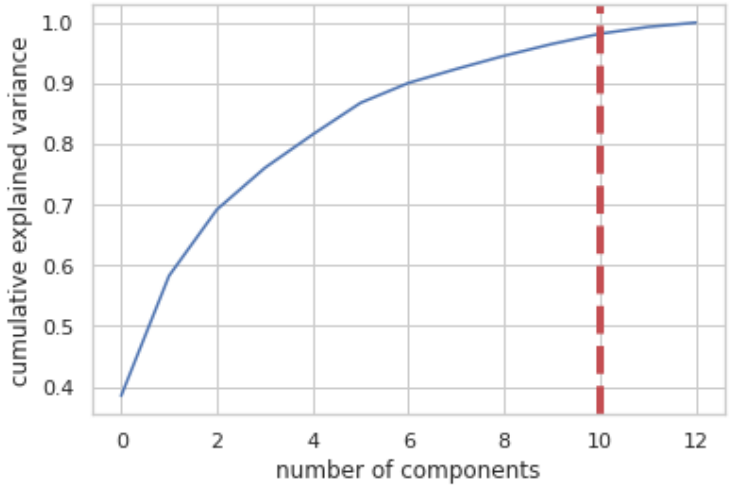



```
None
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
```

```
# WINE DATASET
# Multi Layer Perceptron(With Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
```

```
pca_test = PCA(n_components=13)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters

pca = PCA(n_components=10)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)


# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

###############################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

###############################################################################
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

###############################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)
```

```
y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))


import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

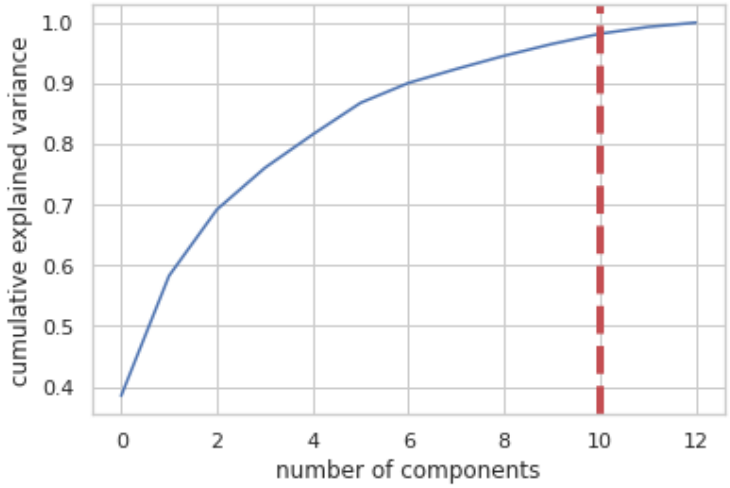



```
None
Parameters currently in use:

{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
```

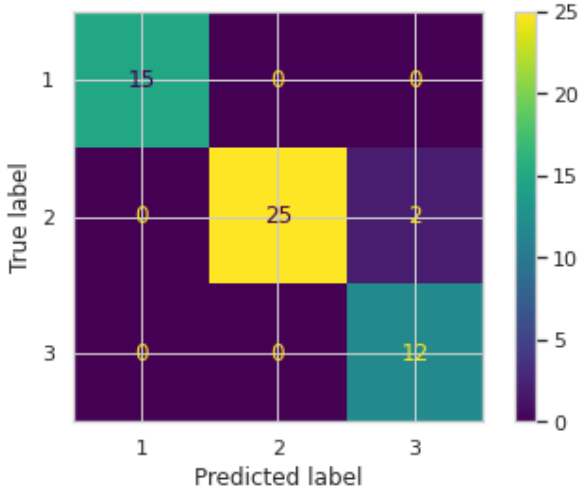
```
# WINE DATASET
# SVM(Without Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)


# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=13)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters

pca = PCA(n_components=10)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)


# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
```

```
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```



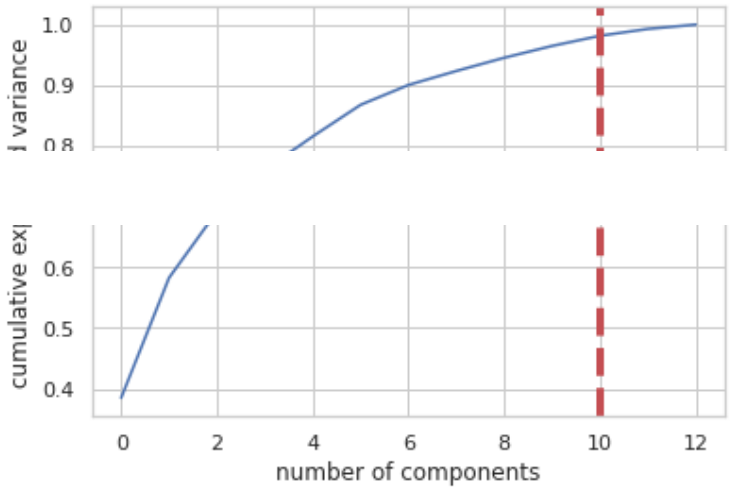

```
None
Confusion Matrix:
[[15  0  0]
 [ 0 25  2]
 [ 0  0 12]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        15
           2       1.00      0.93      0.96        27
           3       0.86      1.00      0.92        12

    accuracy                           0.96        54
   macro avg       0.95      0.98      0.96        54
weighted avg       0.97      0.96      0.96        54

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9629629629629629
```



```
# WINE DATASET
# SVM(With Tuning)[70-30 split]


import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315


df.columns = col_name


X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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


# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=13)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters

pca = PCA(n_components=10)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)
```

```
# Classification
from sklearn.svm import SVC

classifier = SVC()

################################################################################
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

################################################################################
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly',

pprint(param_grid)

################################################################################

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, y_pred))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```




```
None
Parameters currently in use:

{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
```