```
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.pyplot import figure
import plotly.graph_objs as go
from plotly.offline import init_notebook_mode, iplot
from sklearn.utils import shuffle
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
import time
import os

for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# Any results you write to the current directory are saved as output.
```

```
df=pd.read_csv('Training_Set.csv')
vdf=pd.read_csv('Validation_Set.csv')
```

```
df.head(10)
```

	Attribute 1 (a1)	Attribute 2 (a2)	Class Label
0	2	11	2
1	2	13	2
2	2	15	2
3	2	27	1
4	2	39	1
5	4	11	1
6	4	13	1
7	4	15	1
8	4	27	2
9	4	39	2

```
# Let's understand the type of values in each column of our dataframe 'df'.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 3 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Attribute 1 (a1)  30 non-null     int64
 1   Attribute 2 (a2)  30 non-null     int64
 2   Class Label       30 non-null     int64
dtypes: int64(3)
memory usage: 848.0 bytes
```

```
vdf.head(10)
```

	Attribute 1 (a1)	Attribute 2 (a2)	True Class Label	Class Label as predicted by the decision tree	Unnamed: 4	Unnamed: 5	Unnamed: 6
0	2	35	1	1	NaN	NaN	NaN
1	12	13	2	1	NaN	NaN	NaN
2	-4	45	2	2	NaN	NaN	NaN
3	2	17	2	2	NaN	NaN	NaN

```
df.shape
```

```
(30, 3)
```

```
df
```

	Attribute 1 (a1)	Attribute 2 (a2)	Class Label
0	2	11	2
1	2	13	2
2	2	15	2
3	2	27	1
4	2	39	1
5	4	11	1
6	4	13	1
7	4	15	1
8	4	27	2
9	4	39	2
10	6	11	1
11	6	13	1
12	6	15	1
13	6	27	2
14	6	39	2
15	8	11	1
16	8	13	1
17	8	15	1
18	8	27	2
19	8	39	2
20	10	11	1
21	10	13	1
22	10	15	1
23	10	27	2
24	10	39	2
25	12	11	2
26	12	13	1
27	12	15	2
28	12	27	2

```
vdf.shape
```

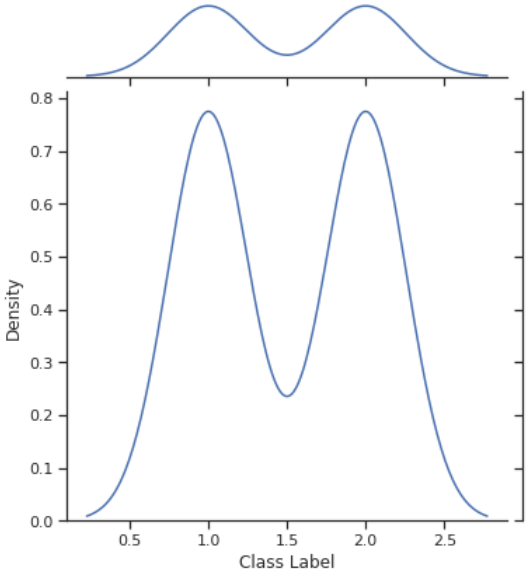
```
(4, 7)
```

```
x=df.iloc[:,:-1].values
y=df.iloc[:,2].values
```

```
sns.set(style="ticks")
f = sns.countplot(x="Class Label", data=df, palette="bwr")
plt.show()
```


```
sns.jointplot(x="Class Label",  data=df, kind="kde")
```

```
<seaborn.axisgrid.JointGrid at 0x7f79e9670358>
```




```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier

X_Train, X_Test, Y_Train, Y_Test = train_test_split(x, y, test_size = 0.30, random_state = 101)
```

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.pyplot import figure
from sklearn.utils import shuffle
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
import time
import os
```

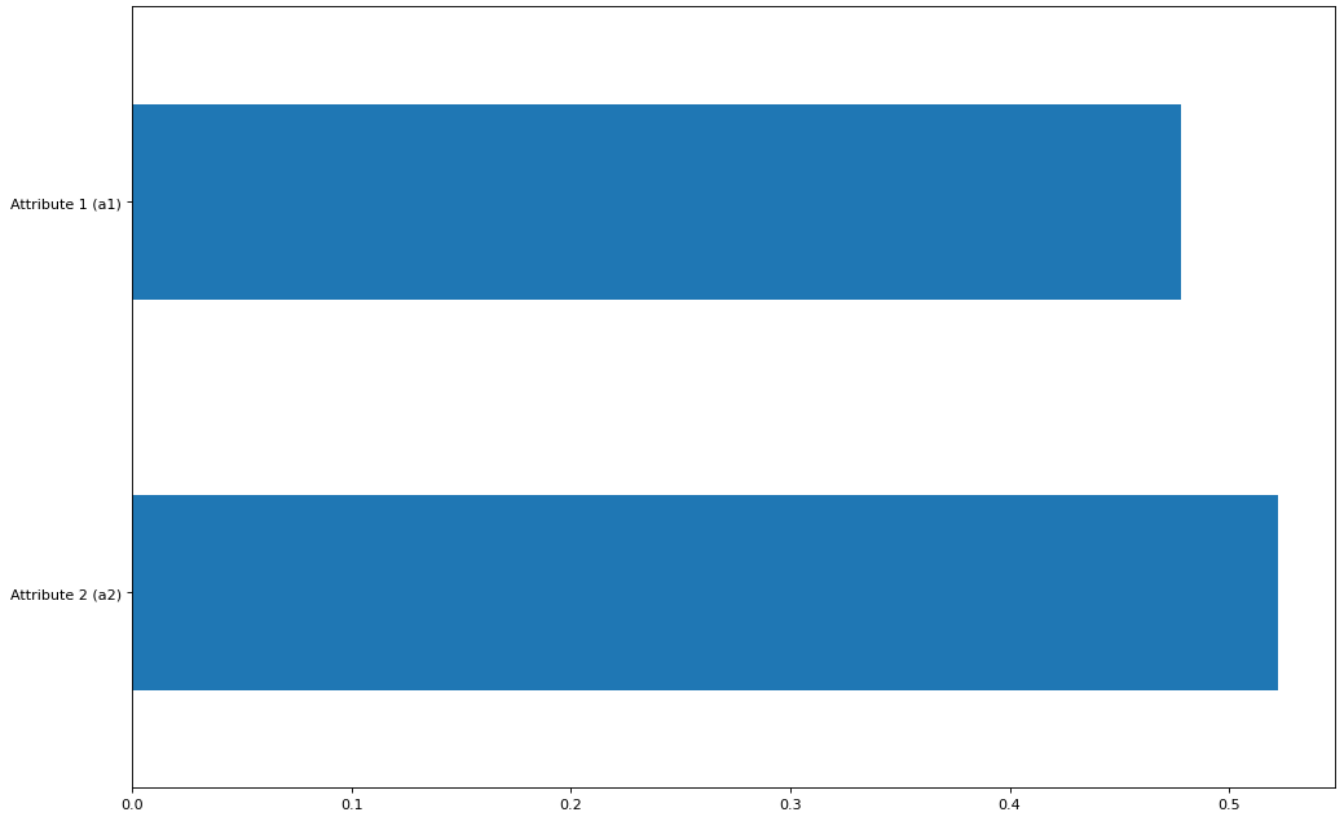
```
start = time.process_time()
trainedforest = RandomForestClassifier(n_estimators=700).fit(X_Train,Y_Train)
print(time.process_time() - start)
predictionforest = trainedforest.predict(X_Test)
print(confusion_matrix(Y_Test,predictionforest))
print(classification_report(Y_Test,predictionforest))
```

```
0.8528058340000007
[[4 0]
 [2 3]]
              precision    recall  f1-score   support

           1       0.67      1.00      0.80         4
           2       1.00      0.60      0.75         5

    accuracy                           0.78         9
   macro avg       0.83      0.80      0.77         9
weighted avg       0.85      0.78      0.77         9
```

```
figure(num=None, figsize=(15, 10), dpi=80, facecolor='w', edgecolor='k')
X=df.iloc[:,:-1]
feat_importances = pd.Series(trainedforest.feature_importances_, index= X.columns)
feat_importances.nlargest(19).plot(kind='barh')
```

```
feat_importances.nlargest(19).plot(kind='barh')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f79f4591940>
```




```
from sklearn.feature_selection import RFE

model = RandomForestClassifier(n_estimators=700)
rfe = RFE(model, 2)
start = time.process_time()
RFE_X_Train = rfe.fit_transform(X_Train,Y_Train)
RFE_X_Test = rfe.transform(X_Test)
rfe = rfe.fit(RFE_X_Train,Y_Train)
print(time.process_time() - start)
print("Overall Accuracy using RFE: ", rfe.score(RFE_X_Test,Y_Test))
```

```
1.6915798720000002
Overall Accuracy using RFE:  0.7777777777777778
```

```
model = RandomForestClassifier(n_estimators=700)
rfe = RFE(model, 6)
RFE_X_Train = rfe.fit_transform(X_Train,Y_Train)
model.fit(RFE_X_Train,Y_Train)
print("Number of Features: ", rfe.n_features_)
print("Selected Features: ")
colcheck = pd.Series(rfe.support_,index = list(X.columns))
colcheck[colcheck == True].index
```

```
Number of Features:  2
Selected Features:
Index(['Attribute 1 (a1)', 'Attribute 2 (a2)'], dtype='object')
```

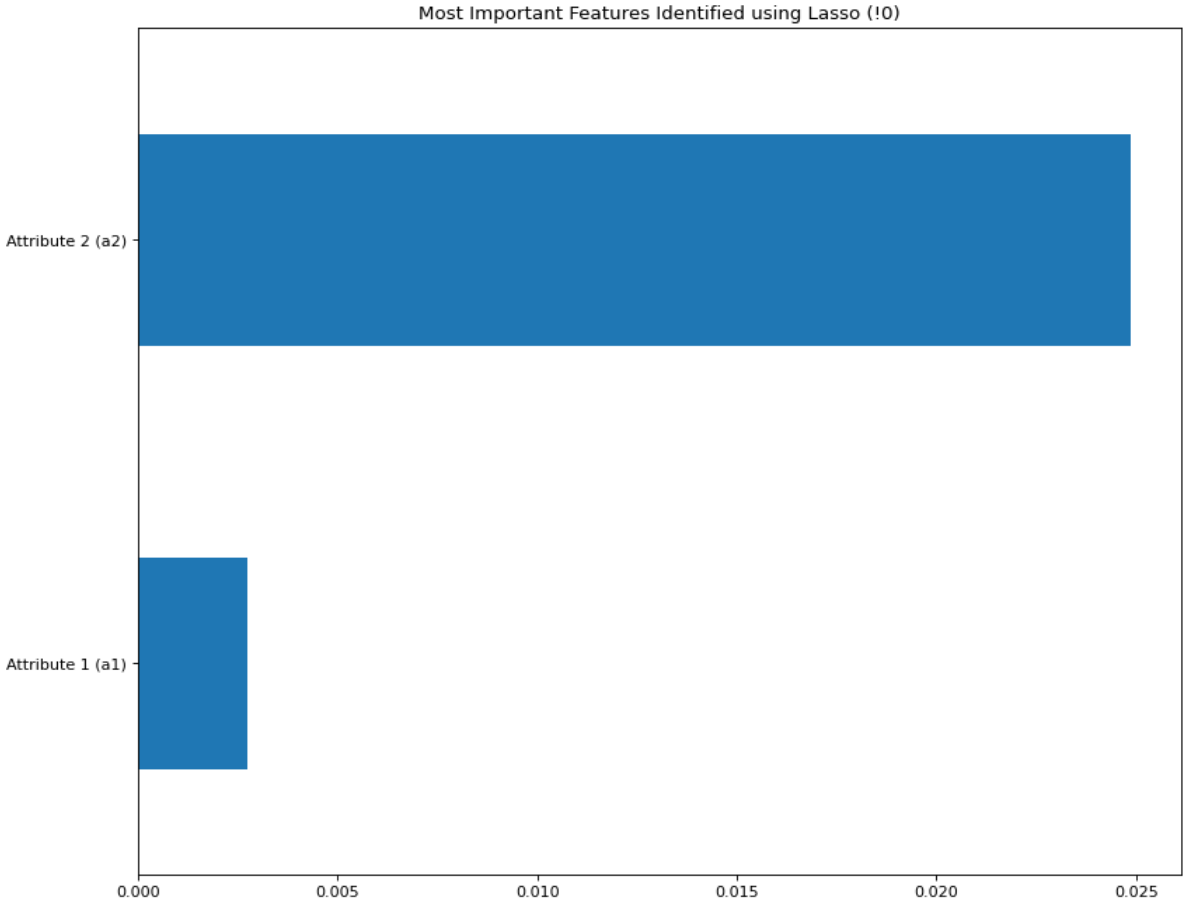
```
from sklearn.linear_model import LassoCV

regr = LassoCV(cv=5, random_state=101)
regr.fit(X_Train,Y_Train)
print("LassoCV Best Alpha Scored: ", regr.alpha_)
print("LassoCV Model Accuracy: ", regr.score(X_Test, Y_Test))
model_coef = pd.Series(regr.coef_, index = list(X.columns))
print("Variables Eliminated: ", str(sum(model_coef == 0)))
print("Variables Kept: ", str(sum(model_coef != 0)))
```

```
LassoCV Best Alpha Scored:  0.037404737808922554
LassoCV Model Accuracy:  0.07722197136342657
```

```
     Variables Eliminated:  0
     Variables Kept:  2
```

```
figure(num=None, figsize=(12, 10), dpi=80, facecolor='w', edgecolor='k')

top_coef = model_coef.sort_values()
top_coef[top_coef != 0].plot(kind = "barh")
plt.title("Most Important Features Identified using Lasso (!0)")
```

```
     Text(0.5, 1.0, 'Most Important Features Identified using Lasso (!0)')
```




```
# Splitting the data into train and test
X_train, X_test, y_train, y_test = train_test_split(x, y,
                                                    test_size=0.20,
                                                    random_state = 99)


# Importing decision tree classifier from sklearn library
from sklearn.tree import DecisionTreeClassifier

# Fitting the decision tree with default hyperparameters, apart from
# max_depth which is 5 so that we can plot and read the tree.
dt_default = DecisionTreeClassifier(max_depth=5)
dt_default.fit(X_train, y_train)
```

```
     DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                            max_depth=5, max_features=None, max_leaf_nodes=None,
                            min_impurity_decrease=0.0, min_impurity_split=None,
                            min_samples_leaf=1, min_samples_split=2,
                            min_weight_fraction_leaf=0.0, presort='deprecated',
                            random_state=None, splitter='best')
```

```
# Let's check the evaluation metrics of our default model

# Importing classification report and confusion matrix from sklearn metrics
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Making predictions
y_pred_default = dt_default.predict(X_test)

# Printing classification report
print(classification_report(y_test, y_pred_default))
```

```
                  precision    recall  f1-score   support
```

```
                1       0.75      1.00      0.86         3
                2       1.00      0.67      0.80         3

         accuracy                           0.83         6
        macro avg       0.88      0.83      0.83         6
     weighted avg       0.88      0.83      0.83         6
```

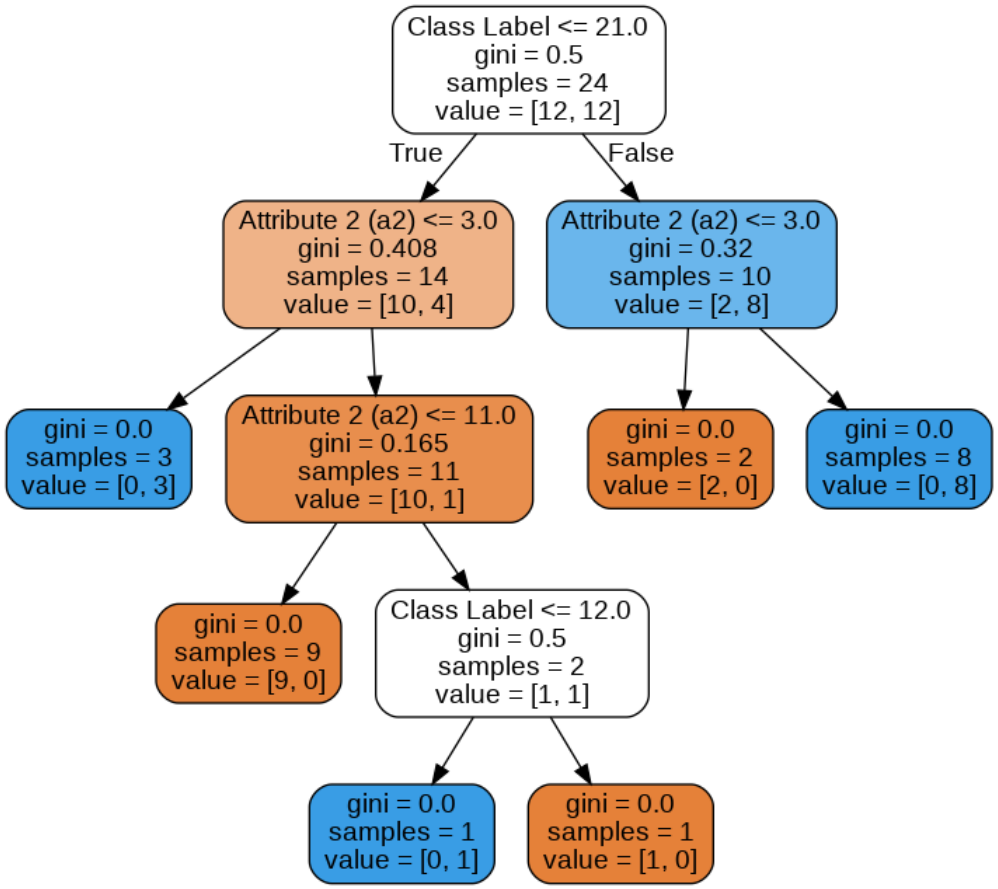
```
# Printing confusion matrix and accuracy
print(confusion_matrix(y_test,y_pred_default))
print(accuracy_score(y_test,y_pred_default))
```

```
[[3 0]
 [1 2]]
0.8333333333333334
```

```
# Importing required packages for visualization
from IPython.display import Image
from sklearn.externals.six import StringIO
from sklearn.tree import export_graphviz
import pydotplus, graphviz

# Putting features
features = list(df.columns[1:])
features
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/externals/six.py:31: FutureWarning:

The module is deprecated in version 0.21 and will be removed in version 0.23 since we've dropped support for Python 2.7. Plea

['Attribute 2 (a2)', 'Class Label']
```

```
# plotting tree with max_depth=3
dot_data = StringIO()
export_graphviz(dt_default, out_file=dot_data,
                feature_names=features, filled=True,rounded=True)

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```




```
# model with optimal hyperparameters
```

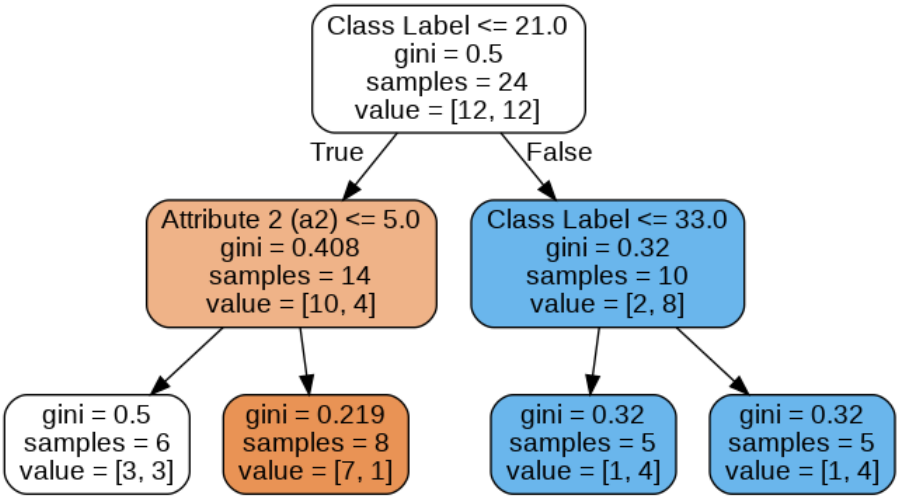
```
clf_gini = DecisionTreeClassifier(criterion = "gini",
                                  random_state = 100,
                                  max_depth=20,
                                  min_samples_leaf=5,
                                  min_samples_split=10)
clf_gini.fit(X_train, y_train)
```

```
     DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                            max_depth=20, max_features=None, max_leaf_nodes=None,
                            min_impurity_decrease=0.0, min_impurity_split=None,
                            min_samples_leaf=5, min_samples_split=10,
                            min_weight_fraction_leaf=0.0, presort='deprecated',
                            random_state=100, splitter='best')
```

```
# accuracy score
clf_gini.score(X_test,y_test)
```

```
     0.8333333333333334
```

```
# plotting the tree
dot_data = StringIO()
export_graphviz(clf_gini, out_file=dot_data,feature_names=features,filled=True,rounded=True)

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```




```
# classification metrics
from sklearn.metrics import classification_report,confusion_matrix
y_pred = clf_gini.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
                  precision    recall  f1-score   support

               1       0.75      1.00      0.86         3
               2       1.00      0.67      0.80         3

        accuracy                           0.83         6
       macro avg       0.88      0.83      0.83         6
    weighted avg       0.88      0.83      0.83         6
```

```
# GridSearchCV to find optimal min_samples_split
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV


# specify number of folds for k-fold CV
n_folds = 5

# parameters to build the model on
parameters = {'min_samples_split': range(5, 200, 20)}

# model with optimal hyperparameters
dtree = DecisionTreeClassifier(criterion = "gini",
                                  random_state = 100,
                                  max_depth=20,
                                  min_samples_leaf=5,
                                  min_samples_split=10)
```

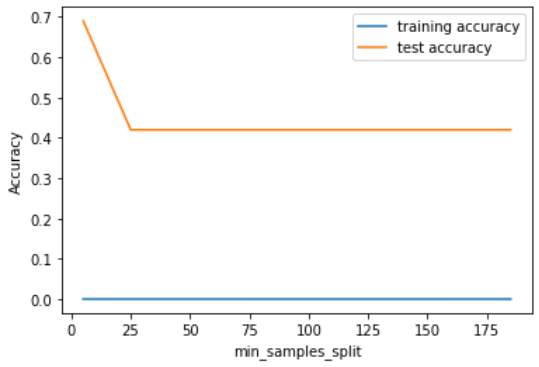
```
# fit tree on training data
tree = GridSearchCV(dtree, parameters,
                    cv=n_folds,
                   scoring="accuracy")
tree.fit(X_train, y_train)
```

```
     GridSearchCV(cv=5, error_score=nan,
                  estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                                   criterion='gini', max_depth=20,
                                                   max_features=None,
                                                   max_leaf_nodes=None,
                                                   min_impurity_decrease=0.0,
                                                   min_impurity_split=None,
                                                   min_samples_leaf=5,
                                                   min_samples_split=10,
                                                   min_weight_fraction_leaf=0.0,
                                                   presort='deprecated',
                                                   random_state=100,
                                                   splitter='best'),
                  iid='deprecated', n_jobs=None,
                  param_grid={'min_samples_split': range(5, 200, 20)},
                  pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                  scoring='accuracy', verbose=0)
```

```
# scores of GridSearch CV
scores = tree.cv_results_
pd.DataFrame(scores).head()
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_min_samples_split	params	split0_test_sco
0	0.000821	0.000454	0.000299	0.000068	5	{'min_samples_split': 5}	0
1	0.000546	0.000020	0.000253	0.000007	25	{'min_samples_split': 25}	0
2	0.000598	0.000056	0.000270	0.000022	45	{'min_samples_split': 45}	0
3	0.000586	0.000078	0.000275	0.000032	65	{'min_samples_split': 65}	0
4	0.000530	0.000016	0.000244	0.000006	85	{'min_samples_split': 85}	0

```
# plotting accuracies with min_samples_leaf
plt.figure()
plt.plot(scores["param_min_samples_split"],
         scores["mean_train_score"],
         label="training accuracy")
plt.plot(scores["param_min_samples_split"],
         scores["mean_test_score"],
         label="test accuracy")
plt.xlabel("min_samples_split")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```




```
# confusion matrix
print(confusion_matrix(y_test,y_pred))
```

```
[[3 0]
 [1 2]]
```