In [1]:
```
import os
os.getcwd()
```

Out[1]:
```
'C:\\Users\\kapil\\Documents'
```

In [2]:
```
import pandas as pd
```

In [4]:
```
# import the dataset
df = pd.read_csv('Heart.csv')
```

In [5]:
```
df.head()
```

Out[5]:

	Unnamed: 0	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Sl
0	1	63	1	typical	145	233	1	2	150	0	2.3	
1	2	67	1	asymptomatic	160	286	0	2	108	1	1.5	
2	3	67	1	asymptomatic	120	229	0	2	129	1	2.6	
3	4	37	1	nonanginal	130	250	0	0	187	0	3.5	
4	5	41	0	nontypical	130	204	0	2	172	0	1.4	

In [6]:
```
# a)Shape of data
```

In [8]:
```
df.shape
```

Out[8]:
```
(303, 15)
```

In [9]:
```
#To find the null values/missing values in dataset
df.isnull()
```

Out[9]:

	Unnamed: 0	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak
0	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...
298	False	False	False	False	False	False	False	False	False	False	False
299	False	False	False	False	False	False	False	False	False	False	False
300	False	False	False	False	False	False	False	False	False	False	False
301	False	False	False	False	False	False	False	False	False	False	False
302	False	False	False	False	False	False	False	False	False	False	False

303 rows × 15 columns

In [10]:
```
# To find how many null values
```

In [11]:
```
df.isnull().sum()
```

Out[11]:
```
Unnamed: 0    0
Age           0
Sex           0
ChestPain     0
RestBP        0
Chol          0
Fbs           0
RestECG       0
MaxHR         0
ExAng         0
Oldpeak       0
Slope         0
Ca            4
Thal          2
AHD           0
dtype: int64
```

In [12]:
```
# Another way
df.count()
```

Out[12]:
```
Unnamed: 0    303
Age           303
Sex           303
ChestPain     303
RestBP        303
Chol          303
Fbs           303
RestECG       303
MaxHR         303
ExAng         303
Oldpeak       303
Slope         303
Ca            299
Thal          301
AHD           303
dtype: int64
```

In [13]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 15 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Unnamed: 0  303 non-null    int64  
 1   Age         303 non-null    int64  
 2   Sex         303 non-null    int64  
 3   ChestPain   303 non-null    object 
 4   RestBP      303 non-null    int64  
 5   Chol        303 non-null    int64  
 6   Fbs         303 non-null    int64  
 7   RestECG     303 non-null    int64  
 8   MaxHR       303 non-null    int64  
 9   ExAng       303 non-null    int64  
 10  Oldpeak     303 non-null    float64
 11  Slope       303 non-null    int64  
 12  Ca          299 non-null    float64
 13  Thal        301 non-null    object 
 14  AHD         303 non-null    object 
dtypes: float64(2), int64(10), object(3)
memory usage: 35.6+ KB
```

In [14]:
```
# Find the datatypes
```

In [15]:
```
df.dtypes
```

Out[15]:
```
Unnamed: 0      int64
Age             int64
Sex             int64
ChestPain      object
RestBP          int64
Chol            int64
Fbs             int64
RestECG         int64
MaxHR           int64
ExAng           int64
Oldpeak       float64
Slope           int64
Ca            float64
Thal           object
AHD            object
dtype: object
```

In [16]:
```
#Finding out zeros where there is true written are 0 values
df == 0
```

Out[16]:

	Unnamed: 0	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak
0	False	False	False	False	False	False	False	False	False	True	False
1	False	False	False	False	False	False	True	False	False	False	False
2	False	False	False	False	False	False	True	False	False	False	False
3	False	False	False	False	False	False	True	True	False	True	False
4	False	False	True	False	False	False	True	False	False	True	False
...	...	...	...	...	...	...	...	...	...	...	...
298	False	False	False	False	False	False	True	True	False	True	False
299	False	False	False	False	False	False	False	True	False	True	False
300	False	False	False	False	False	False	True	True	False	False	False
301	False	False	True	False	False	False	True	False	False	True	True
302	False	False	False	False	False	False	True	True	False	True	True

303 rows × 15 columns

In [17]:
```
#To see 0 values directly
df[df==0]
```

Out[17]:

	Unnamed: 0	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	S
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0	NaN	
1	NaN	NaN	NaN	NaN	NaN	NaN	0.0	NaN	NaN	NaN	NaN	
2	NaN	NaN	NaN	NaN	NaN	NaN	0.0	NaN	NaN	NaN	NaN	
3	NaN	NaN	NaN	NaN	NaN	NaN	0.0	0.0	NaN	0.0	NaN	
4	NaN	NaN	0.0	NaN	NaN	NaN	0.0	NaN	NaN	0.0	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	
298	NaN	NaN	NaN	NaN	NaN	NaN	0.0	0.0	NaN	0.0	NaN	
299	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0	NaN	0.0	NaN	
300	NaN	NaN	NaN	NaN	NaN	NaN	0.0	0.0	NaN	NaN	NaN	
301	NaN	NaN	0.0	NaN	NaN	NaN	0.0	NaN	NaN	0.0	0.0	
302	NaN	NaN	NaN	NaN	NaN	NaN	0.0	0.0	NaN	0.0	0.0	

303 rows × 15 columns

In [18]:
```
# Finding mean age of patient
df.columns
```

Out[18]:
```
Index(['Unnamed: 0', 'Age', 'Sex', 'ChestPain', 'RestBP', 'Chol', 'Fbs',
       'RestECG', 'MaxHR', 'ExAng', 'Oldpeak', 'Slope', 'Ca', 'Thal', 'AHD'],
      dtype='object')
```

In [19]:
```
df['Age']
```

Out[19]:
```
0      63
1      67
2      67
3      37
4      41
      ..
298    45
299    68
300    57
301    57
302    38
Name: Age, Length: 303, dtype: int64
```

In [20]:
```
# Find the mean age
df['Age'].mean()
```

Out[20]:
```
54.43894389438944
```

In [22]:
```
# Extract the only Age, Sex, ChestPain, RestBP,Chol,Randomly divide dataset in trai
newdf = df[['Age','Sex','ChestPain','RestBP','Chol']]
```

In [23]:
```
newdf
```

Out[23]:

	Age	Sex	ChestPain	RestBP	Chol
0	63	1	typical	145	233
1	67	1	asymptomatic	160	286
2	67	1	asymptomatic	120	229
3	37	1	nonanginal	130	250
4	41	0	nontypical	130	204
...	...	...	...	...	...
298	45	1	typical	110	264
299	68	1	asymptomatic	144	193
300	57	1	asymptomatic	130	131
301	57	0	nontypical	130	236
302	38	1	nonanginal	138	175

303 rows × 5 columns

In [24]:
```
# Cross Validation
from sklearn.model_selection import train_test_split
```

In [25]:
```
train,test = train_test_split(df,random_state=0,test_size=0.25)
```

In [26]:
```
train.shape
```

Out[26]:
```
(227, 15)
```

In [27]:
```
test.shape
```

Out[27]:
```
(76, 15)
```

In [28]:
```
# Through the diagnosis test I predicted 100 report as COVID posiive, but only 45 c
# Total 50 people in my sample were actully COVID positive. I have total 500 sample
# Create confusion matrix based on above data and find
# 1. Accuracy 2. Precision  3. Recall  4. F-1 Score
```

In [29]:
```
import numpy as np
```

In [30]:
```
actual = list(np.ones(45)) + list(np.zeros(55))
```

In [31]:
```
np.array(actual)
```

Out[31]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

In [32]:
```
predicted = list(np.ones(40)) + list(np.zeros(52)) + list(np.ones(8))
```

In [33]:
```
np.array(predicted)
```

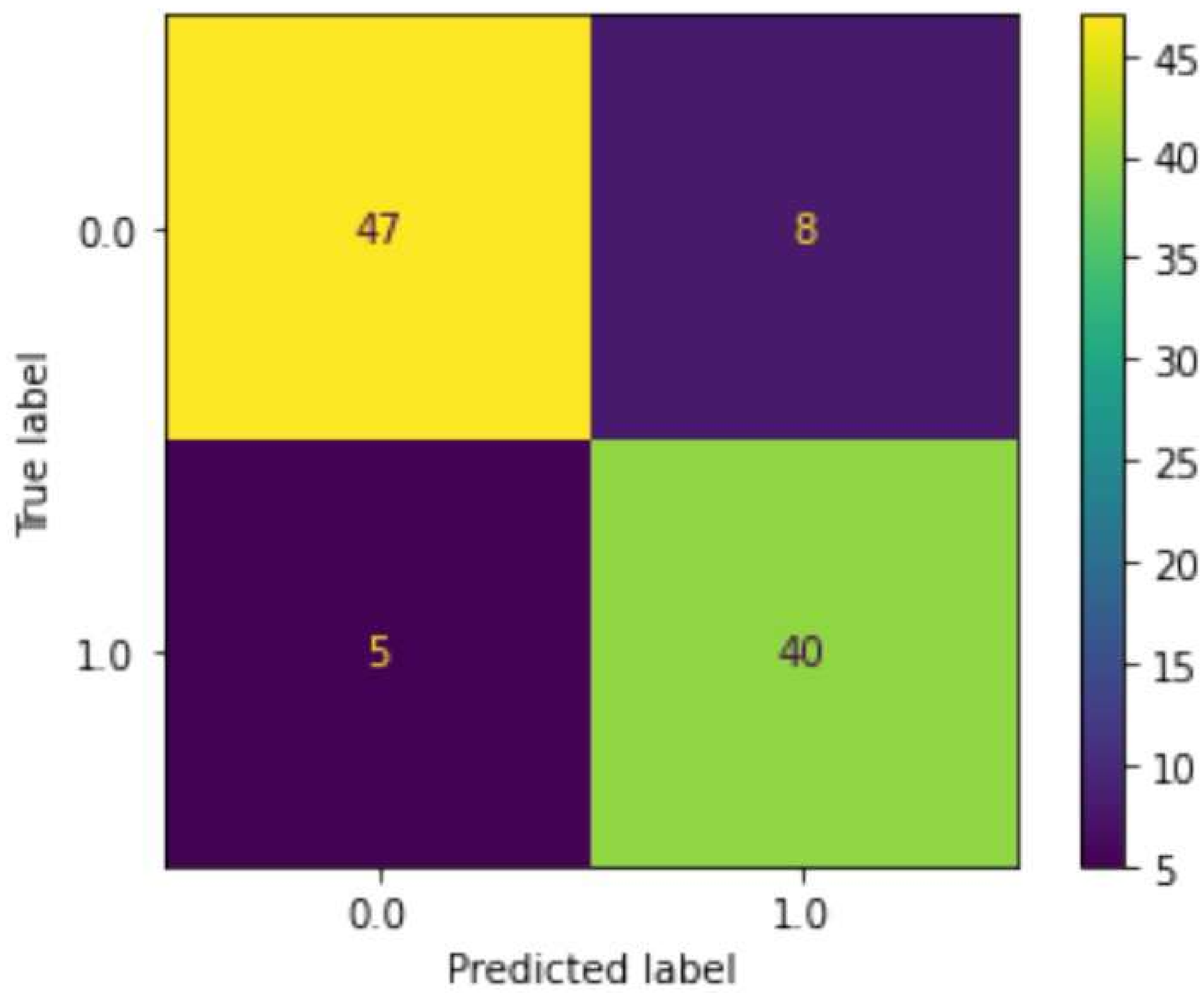
Out[33]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1.])
```

In [34]:
```
# Now if we match the above actual values with predicted values sequentially one by
# we will find that 1 mapped with 1, 1 mapped with 0, 0 mapped with 0 and 0 mapped
# To draw the matrix of it is called confusion matrix
```

In [35]:
```
from sklearn.metrics import ConfusionMatrixDisplay
```

In [36]:
```
ConfusionMatrixDisplay.from_predictions(actual,predicted)
```

Out[36]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2b318dc5cd0>
```



In [37]:
```
# in above matrics actual 1's matching with predicted 1's = 40
#                  actual 0's matching with predicted 1's = 8
#                  actual 1's matching with predicted 0's = 5
#                  actual 0's matching with predicted 0's = 47
```

In [38]:
```
from sklearn.metrics import classification_report
```

In [39]:
```
print(classification_report(actual,predicted))
```

```
              precision    recall  f1-score   support

         0.0       0.90      0.85      0.88        55
         1.0       0.83      0.89      0.86        45

    accuracy                           0.87       100
   macro avg       0.87      0.87      0.87       100
weighted avg       0.87      0.87      0.87       100
```

In [40]:
```
# Recall means individual class accuracy
#47 matching out of 55
# so 47/55 = 0.85
# and 40/45 = 0.89
# precision is check columnwise matrix
# so first column 47+5 =52   i.e  47/52 = 0.90
# and second column 40/48 = 0.83
# f-1 score is harmonic mean of precision and recall
# (0.90+0.85)/2 = 0.875 = 0.88
# (0.83+0.89)/2= 0.86
```

In [ ]:
```

```

In [1]:
```
#Download Temperatures of INDIA dataset from kaggle.com
# Apply Linear Regression using suitable library function and
# predict the Month-wise temperature
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
Matplotlib is building the font cache; this may take a moment.
```

In [2]:
```
df = pd.read_csv('temperatures.csv')
```

In [3]:
```
df
```

Out[3]:

	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAI
0	1901	22.40	24.14	29.07	31.91	33.41	33.18	31.21	30.39	30.47	29.97	27.31	24.49	28.9(
1	1902	24.93	26.58	29.77	31.78	33.73	32.91	30.92	30.73	29.80	29.12	26.31	24.04	29.2
2	1903	23.44	25.03	27.83	31.39	32.91	33.00	31.34	29.98	29.85	29.04	26.08	23.65	28.4
3	1904	22.50	24.73	28.21	32.02	32.64	32.07	30.36	30.09	30.04	29.20	26.36	23.63	28.4
4	1905	22.00	22.83	26.68	30.01	33.32	33.25	31.44	30.68	30.12	30.67	27.52	23.82	28.3(
...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
112	2013	24.56	26.59	30.62	32.66	34.46	32.44	31.07	30.76	31.04	30.27	27.83	25.37	29.8
113	2014	23.83	25.97	28.95	32.74	33.77	34.15	31.85	31.32	30.68	30.29	28.05	25.08	29.7
114	2015	24.58	26.89	29.07	31.87	34.09	32.48	31.88	31.52	31.55	31.04	28.10	25.67	29.9(
115	2016	26.94	29.72	32.62	35.38	35.72	34.03	31.64	31.79	31.66	31.98	30.11	28.01	31.6
116	2017	26.45	29.46	31.60	34.95	35.84	33.82	31.88	31.72	32.22	32.29	29.60	27.18	31.4

117 rows × 18 columns

In [4]:
```
df.head()
```

Out[4]:

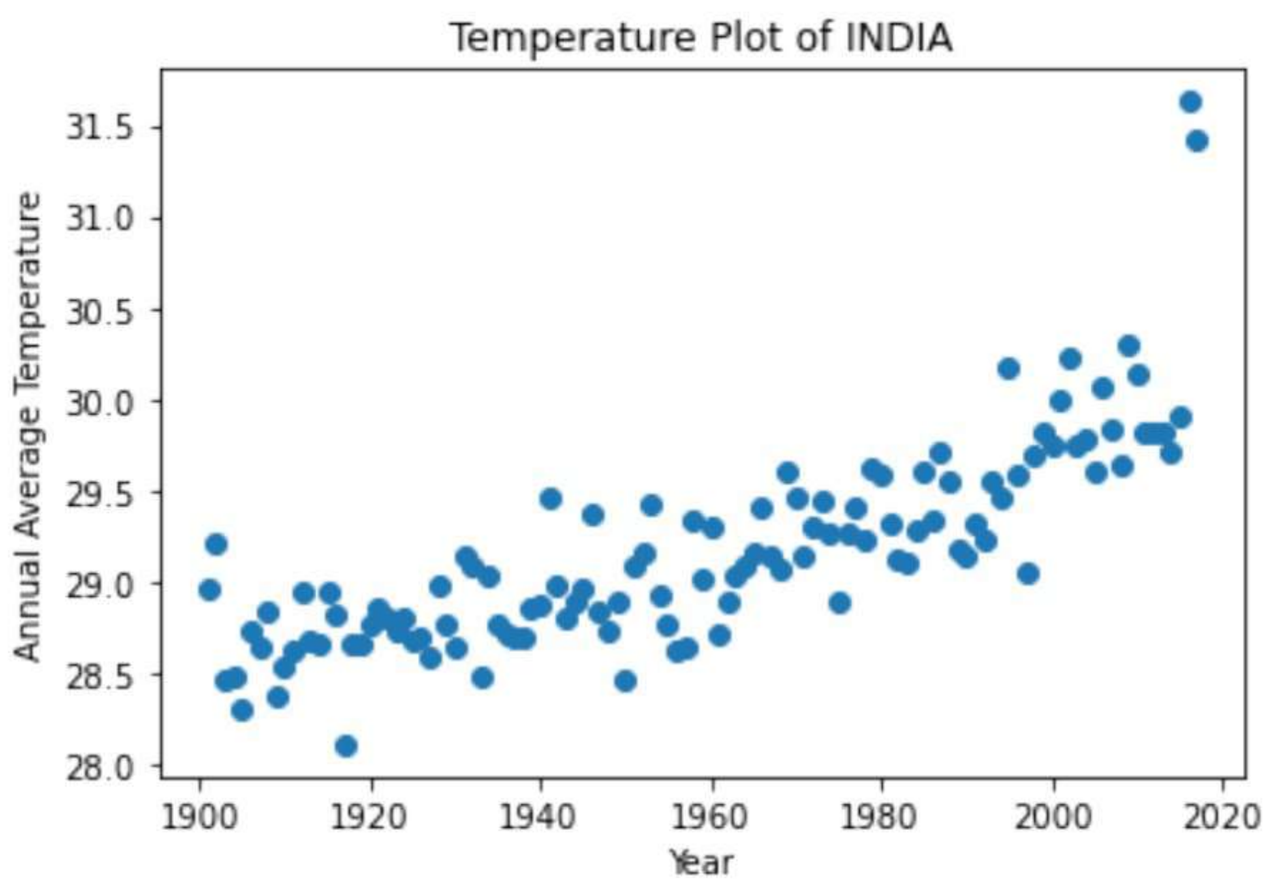
	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAL
0	1901	22.40	24.14	29.07	31.91	33.41	33.18	31.21	30.39	30.47	29.97	27.31	24.49	28.96
1	1902	24.93	26.58	29.77	31.78	33.73	32.91	30.92	30.73	29.80	29.12	26.31	24.04	29.22
2	1903	23.44	25.03	27.83	31.39	32.91	33.00	31.34	29.98	29.85	29.04	26.08	23.65	28.47
3	1904	22.50	24.73	28.21	32.02	32.64	32.07	30.36	30.09	30.04	29.20	26.36	23.63	28.49
4	1905	22.00	22.83	26.68	30.01	33.32	33.25	31.44	30.68	30.12	30.67	27.52	23.82	28.30

In [5]:
```
x = df['YEAR']
```

In [6]:
```
y = df['ANNUAL']
```

In [8]:
```
#plt.figure(figsize=(16,9))
plt.title('Temperature Plot of INDIA')
plt.xlabel('Year')
plt.ylabel('Annual Average Temperature')
plt.scatter(x,y)
```

Out[8]:
```
<matplotlib.collections.PathCollection at 0x14c7bb7fdc0>
```



In [10]:
```
x = x.values
```

In [11]:
```
x = x.reshape(117,1)
```

In [12]:
```
x.shape
```

Out[12]:
```
(117, 1)
```

In [17]:
```
from sklearn.linear_model import LinearRegression
```

In [18]:
```
#Now we are going to train regression model of M/c Learning
regressor = LinearRegression()
```

In [19]:
```
regressor.fit(x,y)
#Model done
```

Out[19]:
```
LinearRegression()
```

In [20]:
```
#Now we will find 'm' value from y = mx + c
regressor.coef_
```

Out[20]:
```
array([0.01312158])
```

In [21]:
```
#Now we will find 'c' value from y = mx + c
regressor.intercept_
```

Out[21]:
```
3.4761897126187016
```

In [25]:
```
regressor.predict([[2120]])
```

Out[25]:
```
array([31.29394211])
```

In [30]:
```
# Assess the performance of regression models using MSE, MAE and R-Square metrics
predicted = regressor.predict(x)
```

In [27]:
```
predicted
```

Out[27]:
```
array([28.4203158 , 28.43343739, 28.44655897, 28.45968055, 28.47280213,
       28.48592371, 28.49904529, 28.51216687, 28.52528846, 28.53841004,
       28.55153162, 28.5646532 , 28.57777478, 28.59089636, 28.60401794,
       28.61713952, 28.63026111, 28.64338269, 28.65650427, 28.66962585,
       28.68274743, 28.69586901, 28.70899059, 28.72211218, 28.73523376,
       28.74835534, 28.76147692, 28.7745985 , 28.78772008, 28.80084166,
       28.81396324, 28.82708483, 28.84020641, 28.85332799, 28.86644957,
       28.87957115, 28.89269273, 28.90581431, 28.91893589, 28.93205748,
       28.94517906, 28.95830064, 28.97142222, 28.9845438 , 28.99766538,
       29.01078696, 29.02390855, 29.03703013, 29.05015171, 29.06327329,
       29.07639487, 29.08951645, 29.10263803, 29.11575961, 29.1288812 ,
       29.14200278, 29.15512436, 29.16824594, 29.18136752, 29.1944891 ,
       29.20761068, 29.22073227, 29.23385385, 29.24697543, 29.26009701,
       29.27321859, 29.28634017, 29.29946175, 29.31258333, 29.32570492,
       29.3388265 , 29.35194808, 29.36506966, 29.37819124, 29.39131282,
       29.4044344 , 29.41755599, 29.43067757, 29.44379915, 29.45692073,
       29.47004231, 29.48316389, 29.49628547, 29.50940705, 29.52252864,
       29.53565022, 29.5487718 , 29.56189338, 29.57501496, 29.58813654,
       29.60125812, 29.6143797 , 29.62750129, 29.64062287, 29.65374445,
       29.66686603, 29.67998761, 29.69310919, 29.70623077, 29.71935236,
       29.73247394, 29.74559552, 29.7587171 , 29.77183868, 29.78496026,
       29.79808184, 29.81120342, 29.82432501, 29.83744659, 29.85056817,
       29.86368975, 29.87681133, 29.88993291, 29.90305449, 29.91617608,
       29.92929766, 29.94241924])
```

In [28]:
```
y
```

Out[28]:
```
0      28.96
1      29.22
2      28.47
3      28.49
4      28.30
       ...
112    29.81
113    29.72
114    29.90
115    31.63
116    31.42
Name: ANNUAL, Length: 117, dtype: float64
```

In [32]:
```
# Mean Absolute Error
import numpy as np
```

```
np.mean(abs(y - predicted))
```

Out[32]:
```
0.22535284978630413
```

In [33]:
```
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y,predicted)
```

Out[33]:
```
0.22535284978630413
```

In [34]:
```
# Mean Squared Error
np.mean((y - predicted) ** 2)
```

Out[34]:
```
0.10960795229110352
```

In [35]:
```
from sklearn.metrics import mean_squared_error
mean_squared_error(y,predicted)
```

Out[35]:
```
0.10960795229110352
```

In [36]:
```
# R-Square Error : How much linearity in this model?
from sklearn.metrics import r2_score
r2_score(y,predicted)
```

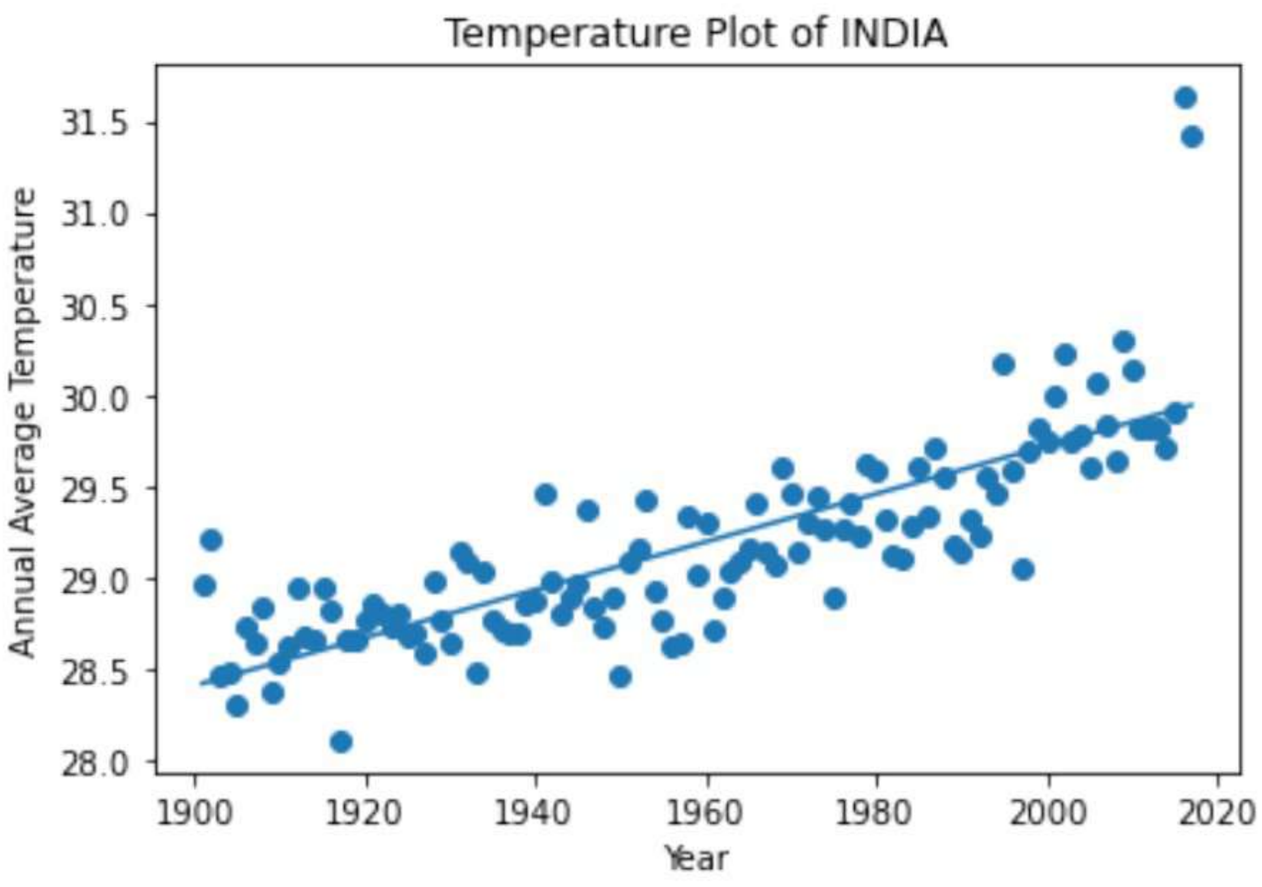
Out[36]:
```
0.64180789127836 82
```

In [37]:
```
regressor.score(x,y)
```

Out[37]:
```
0.64180789127836 82
```

In [38]:
```
# Visualize the regression model
plt.title('Temperature Plot of INDIA')
plt.xlabel('Year')
plt.ylabel('Annual Average Temperature')
plt.scatter(x,y,label = 'actual')
plt.plot(x,predicted, label = 'predicted')
```

Out[38]:
```
[<matplotlib.lines.Line2D at 0x14c7c28f6a0>]
```



In [ ]:

n [2]:
```
import pandas as pd
import seaborn as sns
```

n [3]:
```
df = pd.read_csv('Admission_Predict.csv')
```

n [4]:
```
df.head()
```

ut[4]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4.5	4.5	9.65	1	0.92
1	2	324	107	4	4.0	4.5	8.87	1	0.76
2	3	316	104	3	3.0	3.5	8.00	1	0.72
3	4	322	110	3	3.5	2.5	8.67	1	0.80
4	5	314	103	2	2.0	3.0	8.21	0	0.65

n [5]:
```
df.shape
```

ut[5]:
```
(500, 9)
```

n [6]:
```
from sklearn.preprocessing import Binarizer
```

n [7]:
```
bi = Binarizer(threshold=0.75)
df['Chance of Admit '] = bi.fit_transform(df[['Chance of Admit ']])
```

n [8]:
```
df.head()
```

ut[8]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4.5	4.5	9.65	1	1.0
1	2	324	107	4	4.0	4.5	8.87	1	1.0
2	3	316	104	3	3.0	3.5	8.00	1	0.0
3	4	322	110	3	3.5	2.5	8.67	1	1.0
4	5	314	103	2	2.0	3.0	8.21	0	0.0

n [9]:
```
x = df.drop('Chance of Admit ', axis =1)
y = df['Chance of Admit ']
```

n [10]:
```
x
```

ut[10]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
0	1	337	118	4	4.5	4.5	9.65	1

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
1	2	324	107	4	4.0	4.5	8.87	1
2	3	316	104	3	3.0	3.5	8.00	1
3	4	322	110	3	3.5	2.5	8.67	1
4	5	314	103	2	2.0	3.0	8.21	0
...	...	...	...	...	...	...	...	...
495	496	332	108	5	4.5	4.0	9.02	1
496	497	337	117	5	5.0	5.0	9.87	1
497	498	330	120	5	4.5	5.0	9.56	1
498	499	312	103	4	4.0	5.0	8.43	0
499	500	327	113	4	4.5	4.5	9.04	0

500 rows × 8 columns

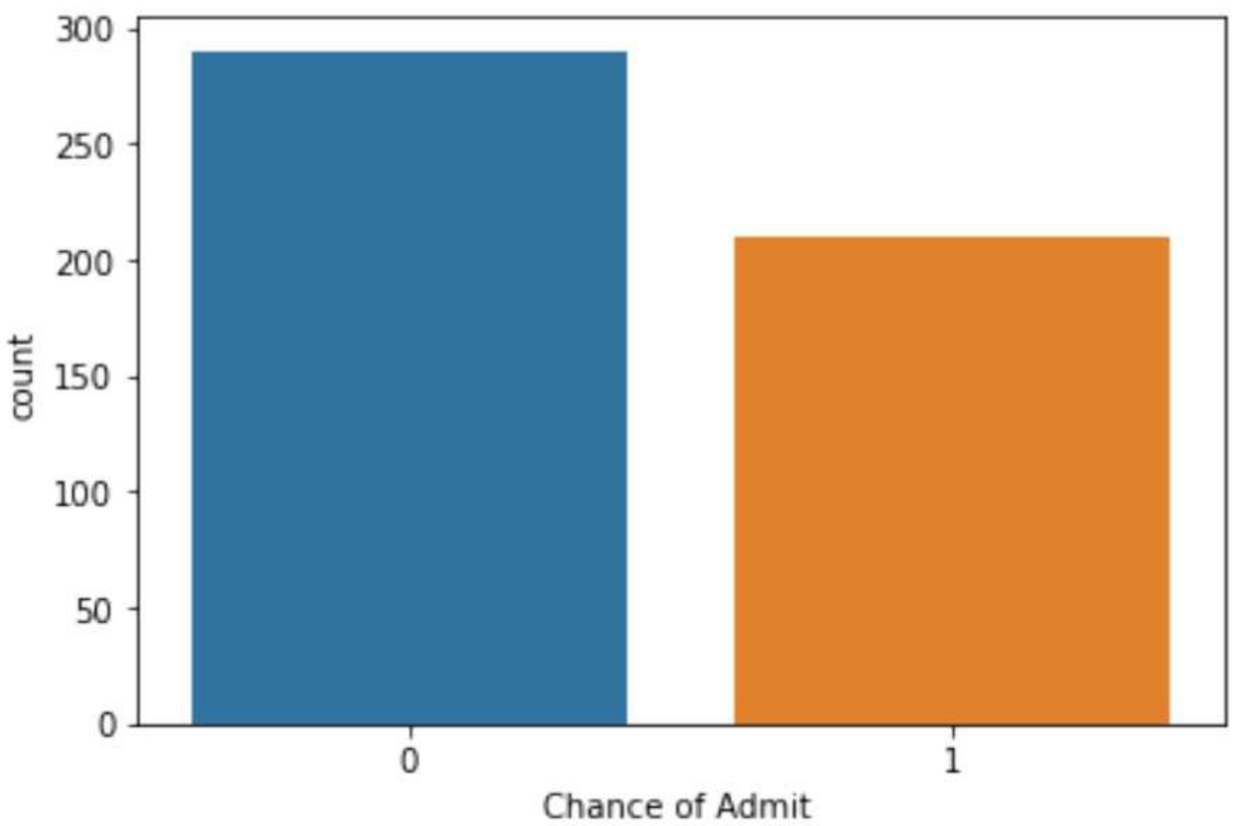
n [11]:
```
y= y.astype('int')
```

n [12]:
```
y
```

ut[12]:
```
0      1
1      1
2      0
3      1
4      0
      ..
495    1
496    1
497    1
498    0
499    1
Name: Chance of Admit , Length: 500, dtype: int32
```

n [13]:
```
sns.countplot(x=y)
```

ut[13]:
```
<AxesSubplot:xlabel='Chance of Admit ', ylabel='count'>
```



n [31]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,random_state=0, test_size =0
```

n [15]:
```
x_train.shape
```

ut[15]:
```
(375, 8)
```

n [16]:
```
x_test.shape
```

ut[16]:
```
(125, 8)
```

n [17]:
```
y_train.shape
```

ut[17]:
```
(375,)
```

n [18]:
```
y_test.shape
```

ut[18]:
```
(125,)
```

n [32]:
```
from sklearn.tree import DecisionTreeClassifier
```

n [33]:
```
classifier = DecisionTreeClassifier(random_state=0)
```

n [34]:
```
classifier.fit(x_train,y_train)
```

ut[34]:
```
DecisionTreeClassifier(random_state=0)
```

n [35]:
```
y_pred = classifier.predict(x_test)
```

n [36]:
```
result = pd.DataFrame({'actual' : y_test,'predicted':y_pred})
```

n [37]:
```
result
```

ut[37]:

	actual	predicted
90	0	0
254	1	1
283	1	1
445	1	1
461	0	0
...	...	...
430	0	0
49	1	0

	actual	predicted
134	1	1
365	1	1
413	0	0

125 rows × 2 columns

In [44]:

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
C:\Users\OSLAB~1\AppData\Local\Temp/ipykernel_3204/1472053813.py in <module>
----> 1 cm = confusion_matrix(y_test, predictions, labels=classifier.classes_)

NameError: name 'confusion_matrix' is not defined
```

In [42]:

```
from sklearn.metrics import ConfusionMatrixDisplay, accuracy_score
```

In [39]:

```
from sklearn.metrics import classification_report
```

In [ ]:

In [43]:

```
accuracy_score(y_test,y_pred)
```

Out[43]:

```
0.96
```

In [50]:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred,labels = classifier.classes_)
```

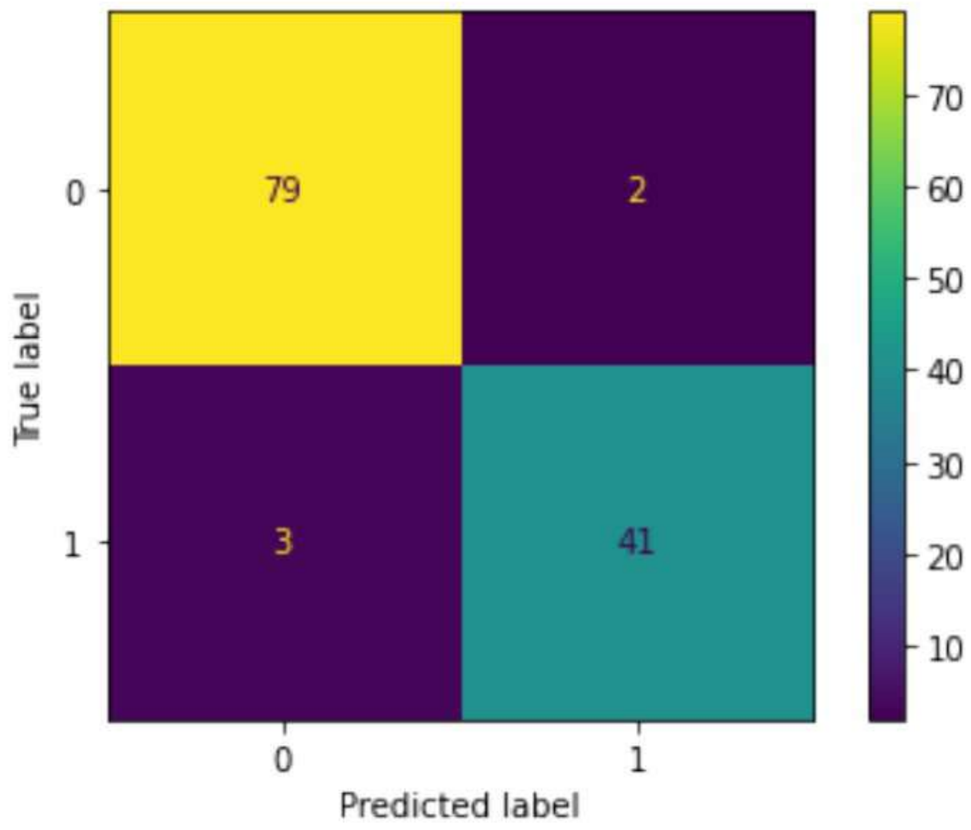
In [51]:

```
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels = classifier.classe
```

In [52]:

```
disp.plot()
```

ut[52]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1c7f4a76c70>



n [54]:
```
accuracy_score(y_test, y_pred)
```

ut[54]: 0.96

n [55]:
```
print(classification_report(y_test, y_pred))
```

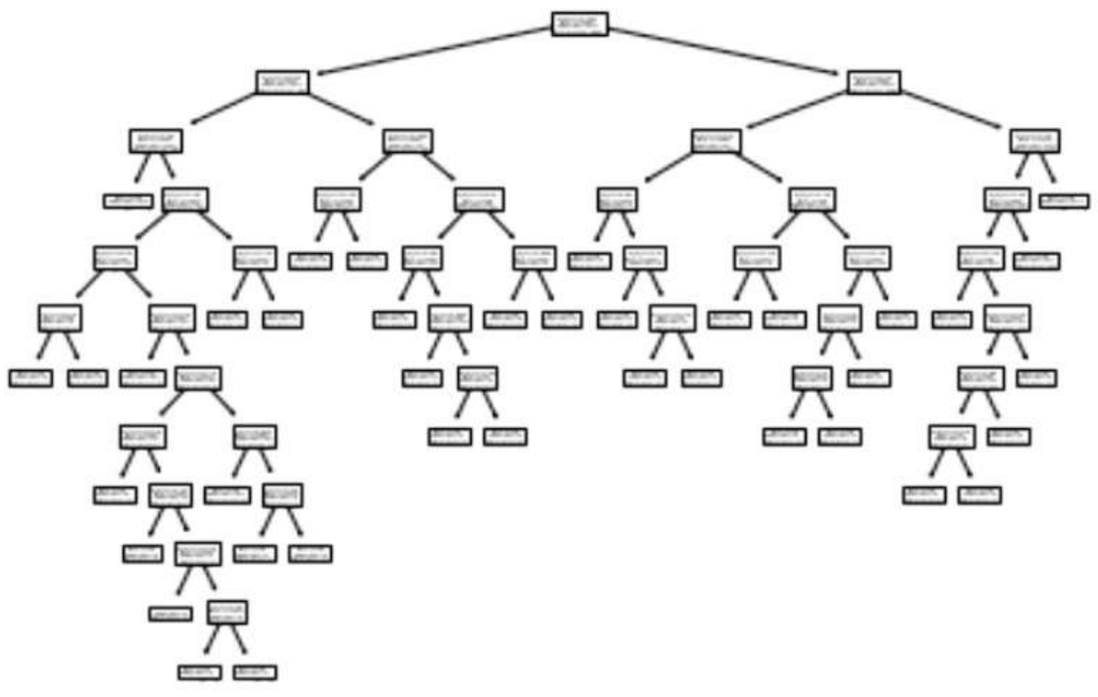
```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97        81
           1       0.95      0.93      0.94        44

    accuracy                           0.96       125
   macro avg       0.96      0.95      0.96       125
weighted avg       0.96      0.96      0.96       125
```

n [68]:
```
new = [[140,300,110,5,4.5,4.5,9.2,1]]
```

n [69]:
```
classifier.predict(new)[0]
```

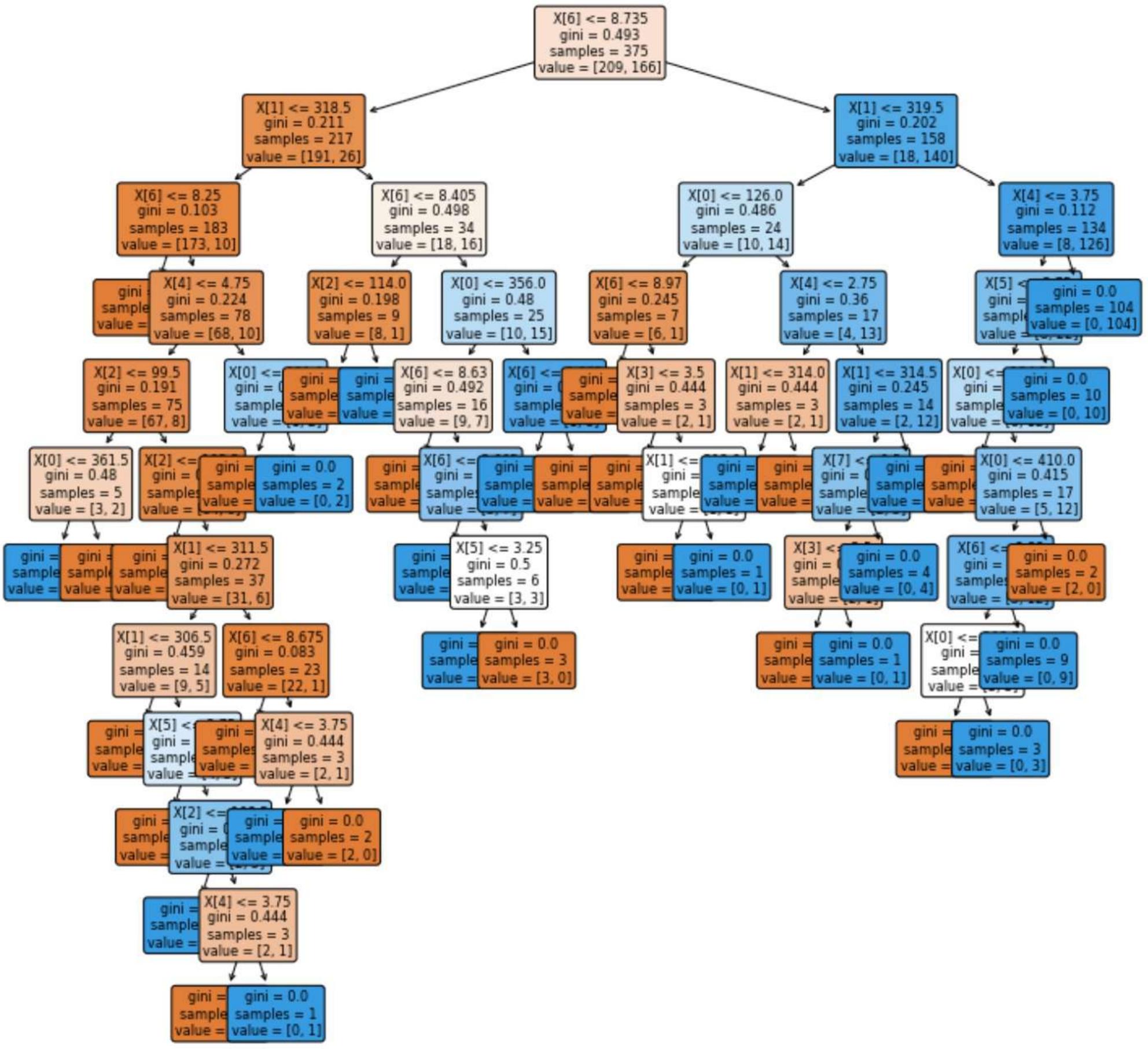
ut[69]: 1

n [70]:
```
from sklearn.tree import plot_tree
plot_tree(classifier, );
```



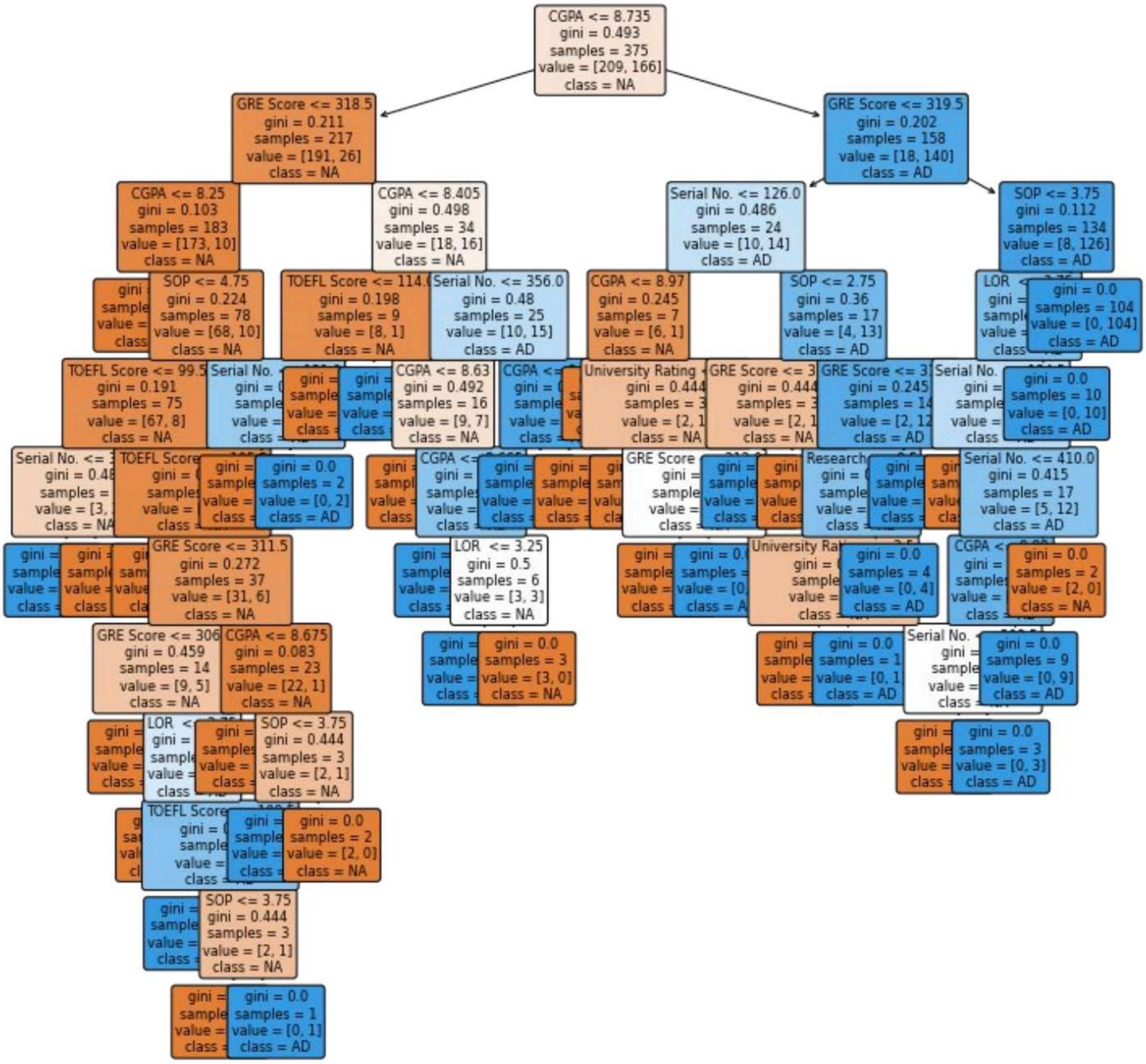
n [73]:

```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
plt.figure(figsize=(12,12))
plot_tree(classifier, fontsize=8, filled=True, rounded = True);
```



n [74]:

```
plt.figure(figsize=(12,12))
plot_tree(classifier, fontsize=8, filled=True, rounded = True, feature_names=x.colum
```


CGPA <= 8.735
gini = 0.493
samples = 375
value = [209, 166]
class = NA
GRE Score <= 318.5
gini = 0.211
samples = 217
value = [191, 26]
class = NA
GRE Score <= 319.5
gini = 0.202
samples = 158
value = [18, 140]
class = AD
CGPA <= 8.25
gini = 0.103
samples = 183
value = [173, 10]
class = NA
CGPA <= 8.405
gini = 0.498
samples = 34
value = [18, 16]
class = NA
Serial No. <= 126.0
gini = 0.486
samples = 24
value = [10, 14]
class = AD
SOP <= 3.75
gini = 0.112
samples = 134
value = [8, 126]
class = AD
SOP <= 4.75
gini = 0.224
samples = 78
value = [68, 10]
class = NA
TOEFL Score <= 114.
gini = 0.198
samples = 9
value = [8, 1]
class = NA
Serial No. <= 356.0
gini = 0.48
samples = 25
value = [10, 15]
class = AD
CGPA <= 8.97
gini = 0.245
samples = 7
value = [6, 1]
class = NA
SOP <= 2.75
gini = 0.36
samples = 17
value = [4, 13]
class = AD
gini = 0.0
samples = 104
value = [0, 104]
class = AD
TOEFL Score <= 99.5
gini = 0.191
samples = 75
value = [67, 8]
class = NA
CGPA <= 8.63
gini = 0.492
samples = 16
value = [9, 7]
class = NA
University Rating <
gini = 0.444
samples = 3
value = [2, 1
class = NA
GRE Score <= 3
gini = 0.444
samples = 3
value = [2, 1
class = NA
GRE Score <= 31
gini = 0.245
samples = 14
value = [2, 12
class = AD
Serial No.
gini = 0.0
samples = 10
value = [0, 10]
class = AD
Serial No. <= 3
gini = 0.48
samples =
value = [3,
class = NA
TOEFL Score
gini = 0.0
samples = 2
value = [0, 2]
class = AD
CGPA <=
gini = 0.415
samples = 17
value = [5, 12]
class = AD
GRE Score
gini = 0.0
samples = 4
value = [0, 4]
class = AD
GRE Score <= 311.5
gini = 0.272
samples = 37
value = [31, 6]
class = NA
LOR <= 3.25
gini = 0.5
samples = 6
value = [3, 3]
class = NA
GRE Score <= 306
gini = 0.459
samples = 14
value = [9, 5]
class = NA
CGPA <= 8.675
gini = 0.083
samples = 23
value = [22, 1]
class = NA
SOP <= 3.75
gini = 0.444
samples = 3
value = [2, 1]
class = NA
gini = 0.0
samples = 3
value = [3, 0]
class = NA
gini = 0.0
samples = 2
value = [2, 0]
class = NA
SOP <= 3.75
gini = 0.444
samples = 3
value = [2, 1]
class = NA
gini = 0.0
samples = 9
value = [0, 9]
class = AD
gini = 0.0
samples = 3
value = [0, 3]
class = AD
gini = 0.0
samples = 1
value = [0, 1]
class = AD

n [ ]:

In [118…
```
import pandas as pd
```

In [119…
```
df = pd.read_csv('SMSSpamCollection', sep = '\t', names = ['label','text'])
```

In [120…
```
df
```

Out[120…

	label	text
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will ü b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its name

5572 rows × 2 columns

In [121…
```
df.shape
```

Out[121…
```
(5572, 2)
```

In [13]:
```
#Now our data should be in number format
#our data is in text format we need to convert
```

```
#before that we need to use some NLP methods here
#we need to delete some unnecessory things from the data means data cleaning
#like punctuation, stopwords like was, the, I , Any, for, he , then etc
# we need to do stemming as well like remove ed from trusted etc
```

In [122…

```
#install nltk natural language tool kit
!pip install nltk
```

```
Requirement already satisfied: nltk in c:\programdata\anaconda3\lib\site-packages (3.6.5)
Requirement already satisfied: click in c:\programdata\anaconda3\lib\site-packages (from nltk) (8.0.3)
Requirement already satisfied: joblib in c:\programdata\anaconda3\lib\site-packages (from nltk) (1.1.0)
Requirement already satisfied: regex>=2021.8.3 in c:\programdata\anaconda3\lib\site-packages (from nltk) (2021.8.3)
Requirement already satisfied: tqdm in c:\programdata\anaconda3\lib\site-packages (from nltk) (4.62.3)
Requirement already satisfied: colorama in c:\programdata\anaconda3\lib\site-packages (from click->nltk) (0.4.4)
```

In [123…

```
import nltk
```

In [124…

```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to C:\Users\OS
[nltk_data]     LAB\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[124…

```
True
```

In [125…

```
sent = 'Hello friends! How are you?'
```

In [19]:

```
#first process is tokenization i.e. symbols separation
```

In [126…

```
from nltk import word_tokenize
```

In [127…

```
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to C:\Users\OS
[nltk_data]     LAB\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[127…
```
True
```

In [128…
```
nltk.word_tokenize(sent)
```

Out[128…
```
['Hello', 'friends', '!', 'How', 'are', 'you', '?']
```

In [129…
```
from nltk.corpus import stopwords
swords = stopwords.words('english')
```

In [11]:
```
swords
```

Out[11]:
```
['i',
 'me',
 'my',
 'myself',
 'we',
 'our',
 'ours',
 'ourselves',
 'you',
 "you're",
 "you've",
 "you'll",
 "you'd",
 'your',
 'yours',
 'yourself',
 'yourselves',
 'he',
 'him',
 'his',
 'himself',
 'she',
 "she's",
 'her',
 'hers',
 'herself',
 'it',
 "it's",
 'its',
```

```
'mightn',
"mightn't",
'mustn',
"mustn't",
'needn',
"needn't",
'shan',
"shan't",
'shouldn',
"shouldn't",
'wasn',
"wasn't",
'weren',
"weren't",
'won',
"won't",
'wouldn',
"wouldn't"]
```

In [130…
```
clean = [word for word in word_tokenize(sent) if word not in swords]
```

In [131…
```
clean
```

Out[131…
```
['Hello', 'friends', '!', 'How', '?']
```

In [14]:
```
#Stemming
```

In [132…
```
from nltk.stem import PorterStemmer
```

In [133…
```
ps = PorterStemmer()
```

In [134…
```
clean = [ps.stem(word) for word in word_tokenize(sent) if word not in swords]
```

In [135…
```
clean
```

Out[135… ['hello', 'friend', '!', 'how', '?']

In [21]:
```
sent1 = 'Hello friends! How are you? We will be learning Python today.'
```

In [136…
```
def clean_text(sent):
    tokens = word_tokenize(sent)
    clean = [word for word in tokens if word.isdigit() or word.isalpha()]
    clean = [ps.stem(word) for word in clean if word not in swords]
    return clean
```

In [137…
```
clean_text(sent1)
```

Out[137… ['hello', 'friend', 'how', 'we', 'learn', 'python', 'today']

In [27]:
```
#Above we learned the Preprocessing
```

In [30]:
```
# preprocessing method to use text data is TF*IDF vectorizer
```

In [31]:
```
#TF*IDF algo is used to weigh a keyword in any document and assign the importance to that
# keyword based on the number of times it appears in the document
# Put simply, the higher the TF*IDF score (weight), the rarer and more importan the term, and vice versa
# Each word or term has its respective TF and IDF score.
#The product of the TF and IDF scores of a term is called the TF*IDF weight of that term.
#The TF(Term Frequency) of a word is the number of times it appears in a doc.
#You can understand that you are using a term too often or too infrequently.
# TF(t) = (Number of times term t appears in a doc)/(Total number of terms in the doc)

# The IDF (Inverse Doc Frequency) of a word is the measure of how significant that term is in
#the whole corpus.

# IDF(t) = log_e(Total number of documents/Number of documents with term t in it)
```

In [138…
```
# PreProcessing
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [139…
```
tfidf = TfidfVectorizer(analyzer = clean_text)
```

In [140…
```
x = df['text']
y = df['label']
```

In [141…
```
#tranform into numbers
x_new = tfidf.fit_transform(x)
```

In [142…
```
x.shape
```

Out[142…
```
(5572,)
```

In [143…
```
x_new.shape
```

Out[143…
```
(5572, 6513)
```

In [144…
```
x_new
```

Out[144…
```
<5572x6513 sparse matrix of type '<class 'numpy.float64'>'
        with 52573 stored elements in Compressed Sparse Row format>
```

In [ ]:
```
#We have done vectorization after cleaning
```

In [145…
```
tfidf.get_feature_names()
```

Out[145…
```
['0',
 '008704050406',
 '0089',
 '0121',
```

```
'01223585236',
'01223585334',
'0125698789',
'02',
'0207',
'02072069400',
'02073162414',
'02085076972',
'021',
'050703',
'0578',
'06',
'07008009200',
'07046744435',
'07090201529',
'07090298926',
'07099833605',
'07123456789',
'0721072',
'07732584351',
'07734396839',
'07742676969',
'07753741225',
'07786200117',
'078',
'07801543489',
'07808',
'07808247860',
'07808726822',
'07815296484',
'07821230901',
'078498',
'07973788240',
'0800',
'08000407165',
'08000776320',
'08000839402',
'08000930705',
'08000938767',
'08001950382',
'08002888812',
'08002986030',
'08002986906',
'08002988890',
```

```
'bbq',
'bc',
'bcaz',
'bck',
'bcm',
'bcoz',
'bcum',
'bcz',
'bday',
'be',
'beach',
'bead',
'bear',
'beat',
'beauti',
'bec',
'becau',
'becaus',
'becausethey',
'becom',
'becoz',
'becz',
'bed',
'bedbut',
'bedreal',
'bedrm',
'bedroom',
'beeen',
...]
```

In [66]:
```
# Cross Validation
```

In [146…
```
y.value_counts()
```

Out[146…
```
ham     4825
spam     747
Name: label, dtype: int64
```

In [147…
```
from sklearn.model_selection import train_test_split
```

In [148…
```
x_train, x_test, y_train, y_test = train_test_split(x_new, y, random_state=0, test_size=0.25)
```

In [150…
```
x_train.shape
```

Out[150…
```
(4179, 6513)
```

In [151…
```
x_test.shape
```

Out[151…
```
(1393, 6513)
```

In [152…
```
from sklearn.naive_bayes import GaussianNB
```

In [153…
```
nb = GaussianNB()
```

In [154…
```
nb.fit(x_train.toarray(), y_train)
```

Out[154…
```
GaussianNB()
```

In [155…
```
y_pred = nb.predict(x_test.toarray())
```

In [156…
```
y_test.value_counts()
```

Out[156…
```
ham     1208
spam     185
Name: label, dtype: int64
```

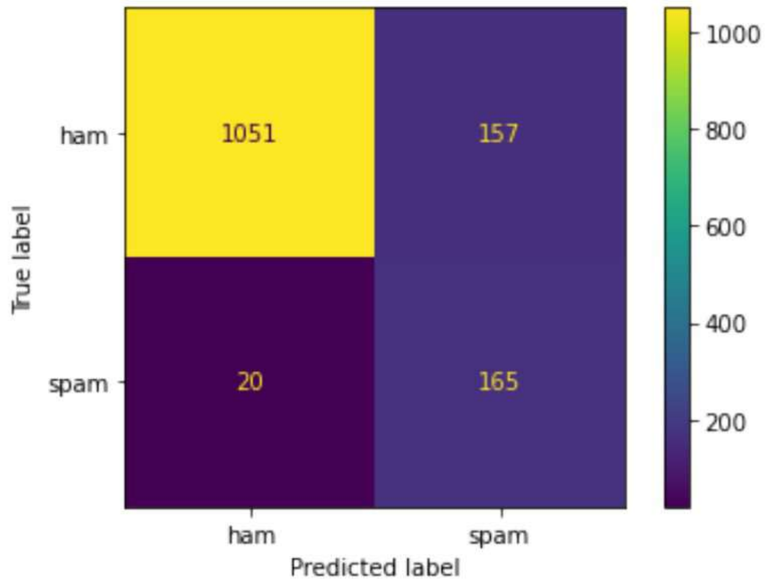
In [157…
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

In [ ]:
```

```

In [158…
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred,labels = nb.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels = nb.classes_)
```

In [159…
```
disp.plot()
```

Out[159…
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x213d539bdc0>
```



In [160…
```
from sklearn.metrics import classification_report, accuracy_score
```

In [161…
```
accuracy_score(y_test, y_pred)
```

Out[161…
```
0.8729361091170137
```

In [162…
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         ham       0.98      0.87      0.92      1208
        spam       0.51      0.89      0.65       185
```

```
              accuracy                           0.87      1393
             macro avg       0.75      0.88      0.79      1393
          weighted avg       0.92      0.87      0.89      1393
```

In [163…
```
from sklearn.ensemble import RandomForestClassifier
```

In [164…
```
rf = RandomForestClassifier(random_state=0)
```

In [165…
```
rf.fit(x_train, y_train)
```

Out[165…
```
RandomForestClassifier(random_state=0)
```

In [167…
```
y_pred = rf.predict(x_test)
```

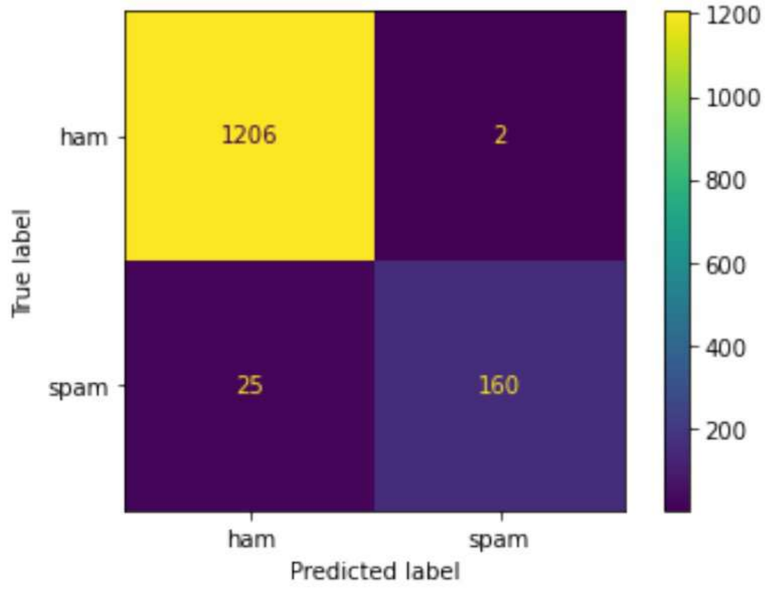
In [168…
```
ConfusionMatrixDisplay.from_predictions(y_test,y_pred);
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
C:\Users\OSLAB~1\AppData\Local\Temp/ipykernel_17028/394578003.py in <module>
----> 1 ConfusionMatrixDisplay.from_predictions(y_test,y_pred);

AttributeError: type object 'ConfusionMatrixDisplay' has no attribute 'from_predictions'
```

In [169…
```
cm = confusion_matrix(y_test, y_pred,labels = rf.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels = rf.classes_)
```

In [170…
```
disp.plot()
```

Out[170…
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x213d8b68fa0>
```



In [171…

```
accuracy_score(y_test, y_pred)
```

Out[171…

```
0.9806173725771715
```

In [172…

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         ham       0.98      1.00      0.99      1208
        spam       0.99      0.86      0.92       185

    accuracy                           0.98      1393
   macro avg       0.98      0.93      0.96      1393
weighted avg       0.98      0.98      0.98      1393
```

In [173…

```
from sklearn.linear_model import LogisticRegression
log = LogisticRegression()
log.fit(x_train, y_train)
y_pred = log.predict(x_test)
accuracy_score(y_test, y_pred)
```

Out[173… 0.9641062455132807

In [117…
```
# RandomForest accuracy looks good
```

In [ ]:
```
# hyperparameter tuning and evaluate the model
# any algorithm we passing parameters that parameters we need to decide ideally
```

In [174…
```
from sklearn.model_selection import GridSearchCV
```

In [ ]:
```
#Gridsearch is a class of cross validation
# we need to create object of that class first
```

In [ ]:
```
#https://scikit-learn.org/stable/modules/generated/
#sklearn.ensemble.RandomForestClassifier.html#randomforestclassifier
# see two parameters gini and entropy
```

In [179…
```
params= {
    'criterion': ['gini','entropy'],
    'max_features': ['sqrt','log2'],
    'random_state': [0,1,2,3,4],
    'class_weight': ['balanced','balanced_subsample']
}
```

In [180…
```
grid = GridSearchCV(rf,param_grid=params, cv = 5,scoring='accuracy')
```

In [181…
```
# GridSearch cross validation will search the ideal values
#for the parameters given in params above
```

In [182…
```
grid.fit(x_train, y_train)
```

Out[182…
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=0),
             param_grid={'class_weight': ['balanced', 'balanced_subsample'],
```

```
                    'criterion': ['gini', 'entropy'],
                    'max_features': ['sqrt', 'log2'],
                    'random_state': [0, 1, 2, 3, 4]},
             scoring='accuracy')
```

In [183…
```
# Above may take 5 to 10 minutes
```

In [186…
```
grid.best_estimator_
```

Out[186…
```
RandomForestClassifier(class_weight='balanced_subsample', max_features='sqrt',
                       random_state=1)
```

In [ ]:
```
#Above you can see estimated parameters
```

In [187…
```
rf = grid.best_estimator_
```

In [188…
```
y_pred = rf.predict(x_test)
```

In [190…
```
accuracy_score(y_test,y_pred)
```

Out[190…
```
0.9770279971284996
```

In [ ]:
```
# so using hyper parameter tuning we can find the accuracy of the algorithm
# as well as model performance and finding ideal values for parameters
```