PRODIGY_DS_03

Task 3.To build a decision tree classifier to predict whether a customer will purchase a product or service based on their demographic and behavioral data.

INTRODUCTION :

A Decision Tree algorithm is one of the most popular machine learning algorithms. It uses a tree like structure and their possible combinations to solve a particular problem. It belongs to the class of supervised learning algorithms where it can be used for both classification and regression purposes.

A decision tree is a structure that includes a root node, branches, and leaf nodes. Each internal node denotes a test on an attribute, each branch denotes the outcome of a test, and each leaf node holds a class label. The topmost node in the tree is the root node.

We make some assumptions while implementing the Decision-Tree algorithm. These are listed below:-

1 At the beginning, the whole training set is considered as the root.

2 Feature values need to be categorical. If the values are continuous then they are discretized prior to building the model.

3 Records are distributed recursively on the basis of attribute values.

4 Order to placing attributes as root or internal node of the tree is done by using some statistical approach.

DATA DESCRIPTION :

The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed.

Loading Data Set :

In [57]:
```
#pip install ucimlrepo
from ucimlrepo import fetch_ucirepo

# fetch dataset
bank_marketing = fetch_ucirepo(id=222)

# data (as pandas dataframes)
X = bank_marketing.data.features
y = bank_marketing.data.targets

# metadata
print(bank_marketing.metadata)

# variable information
print(bank_marketing.variables)
```

{'uci_id': 222, 'name': 'Bank Marketing', 'repository_url': 'https://archive.ics.uc
i.edu/dataset/222/bank+marketing', 'data_url': 'https://archive.ics.uci.edu/static/
public/222/data.csv', 'abstract': 'The data is related with direct marketing campai
gns (phone calls) of a Portuguese banking institution. The classification goal is t
o predict if the client will subscribe a term deposit (variable y).', 'area': 'Busi
ness', 'tasks': ['Classification'], 'characteristics': ['Multivariate'], 'num_insta
nces': 45211, 'num_features': 16, 'feature_types': ['Categorical', 'Integer'], 'dem
ographics': ['Age', 'Occupation', 'Marital Status', 'Education Level'], 'target_col
': ['y'], 'index_col': None, 'has_missing_values': 'yes', 'missing_values_symbol':
'NaN', 'year_of_dataset_creation': 2014, 'last_updated': 'Fri Aug 18 2023', 'datase
t_doi': '10.24432/C5K306', 'creators': ['S. Moro', 'P. Rita', 'P. Cortez'], 'intro_
paper': {'title': 'A data-driven approach to predict the success of bank telemarket
ing', 'authors': 'Sérgio Moro, P. Cortez, P. Rita', 'published_in': 'Decision Suppo
rt Systems', 'year': 2014, 'url': 'https://www.semanticscholar.org/paper/cab8605288
2d126d43f72108c6cb41b295cc8a9e', 'doi': '10.1016/j.dss.2014.03.001'}, 'additional_i
nfo': {'summary': "The data is related with direct marketing campaigns of a Portugu
ese banking institution. The marketing campaigns were based on phone calls. Often,
more than one contact to the same client was required, in order to access if the pr
oduct (bank term deposit) would be ('yes') or not ('no') subscribed. \n\nThere are
four datasets: \n1) bank-additional-full.csv with all examples (41188) and 20 input
s, ordered by date (from May 2008 to November 2010), very close to the data analyze
d in [Moro et al., 2014]\n2) bank-additional.csv with 10% of the examples (4119), r
andomly selected from 1), and 20 inputs.\n3) bank-full.csv with all examples and 17
inputs, ordered by date (older version of this dataset with less inputs). \n4) ban
k.csv with 10% of the examples and 17 inputs, randomly selected from 3 (older versi
on of this dataset with less inputs). \nThe smallest datasets are provided to test
more computationally demanding machine learning algorithms (e.g., SVM). \n\nThe cla
ssification goal is to predict if the client will subscribe (yes/no) a term deposit
(variable y).", 'purpose': None, 'funded_by': None, 'instances_represent': None, 'r
ecommended_data_splits': None, 'sensitive_data': None, 'preprocessing_description':
None, 'variable_info': 'Input variables:\n   # bank client data:\n   1 - age (numer
ic)\n   2 - job : type of job (categorical: "admin.","unknown","unemployed","manage
ment","housemaid","entrepreneur","student",\n
"blue-collar","self-employed","retired","technician","services") \n   3 - marital :
marital status (categorical: "married","divorced","single"; note: "divorced" means
divorced or widowed)\n   4 - education (categorical: "unknown","secondary","primar
y","tertiary")\n   5 - default: has credit in default? (binary: "yes","no")\n   6 -
balance: average yearly balance, in euros (numeric) \n   7 - housing: has housing l
oan? (binary: "yes","no")\n   8 - loan: has personal loan? (binary: "yes","no")\n
related with the last contact of the current campaign:\n   9 - contact: contact c
ommunication type (categorical: "unknown","telephone","cellular") \n 10 - day: las
t contact day of the month (numeric)\n 11 - month: last contact month of year (cat
egorical: "jan", "feb", "mar", ..., "nov", "dec")\n 12 - duration: last contact du
ration, in seconds (numeric)\n   # other attributes:\n 13 - campaign: number of co
ntacts performed during this campaign and for this client (numeric, includes last c
ontact)\n 14 - pdays: number of days that passed by after the client was last cont
acted from a previous campaign (numeric, -1 means client was not previously contact
ed)\n 15 - previous: number of contacts performed before this campaign and for thi
s client (numeric)\n 16 - poutcome: outcome of the previous marketing campaign (ca
tegorical: "unknown","other","failure","success")\n\n Output variable (desired tar
get):\n 17 - y - has the client subscribed a term deposit? (binary: "yes","no")\n
', 'citation': None}}

	name	role	type	demographic
0	age	Feature	Integer	Age
1	job	Feature	Categorical	Occupation
2	marital	Feature	Categorical	Marital Status
3	education	Feature	Categorical	Education Level
4	default	Feature	Binary	None
5	balance	Feature	Integer	None

6	housing	Feature	Binary	None	
7	loan	Feature	Binary	None	
8	contact	Feature	Categorical	None	
9	day_of_week	Feature	Date	None	
10	month	Feature	Date	None	
11	duration	Feature	Integer	None	
12	campaign	Feature	Integer	None	
13	pdays	Feature	Integer	None	
14	previous	Feature	Integer	None	
15	poutcome	Feature	Categorical	None	
16	y	Target	Binary	None	

	description	units	missing_values
0	None	None	no
1	type of job (categorical: 'admin.','blue-colla...	None	no
2	marital status (categorical: 'divorced','marri...	None	no
3	(categorical: 'basic.4y','basic.6y','basic.9y'...	None	no
4	has credit in default?	None	no
5	average yearly balance	euros	no
6	has housing loan?	None	no
7	has personal loan?	None	no
8	contact communication type (categorical: 'cell...	None	yes
9	last contact day of the week	None	no
10	last contact month of year (categorical: 'jan'...	None	no
11	last contact duration, in seconds (numeric). ...	None	no
12	number of contacts performed during this campa...	None	no
13	number of days that passed by after the client...	None	yes
14	number of contacts performed before this campa...	None	no
15	outcome of the previous marketing campaign (ca...	None	yes
16	has the client subscribed a term deposit?	None	no

Import Libraries :

In [61]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import LabelEncoder
```

Understanding and Describing Data Set :

In [63]:
```
Banking_data = X.copy()
Banking_data['y'] = y
Banking_data.head()
```

Out[63]:

	age	job	marital	education	default	balance	housing	loan	contact	day_of_week
0	58	management	married	tertiary	no	2143	yes	no	NaN	5
1	44	technician	single	secondary	no	29	yes	no	NaN	5
2	33	entrepreneur	married	secondary	no	2	yes	yes	NaN	5
3	47	blue-collar	married	NaN	no	1506	yes	no	NaN	5
4	33	NaN	single	NaN	no	1	no	no	NaN	5

In [65]:
```
Banking_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   age          45211 non-null  int64
 1   job          44923 non-null  object
 2   marital      45211 non-null  object
 3   education    43354 non-null  object
 4   default      45211 non-null  object
 5   balance      45211 non-null  int64
 6   housing      45211 non-null  object
 7   loan         45211 non-null  object
 8   contact      32191 non-null  object
 9   day_of_week  45211 non-null  int64
 10  month        45211 non-null  object
 11  duration     45211 non-null  int64
 12  campaign     45211 non-null  int64
 13  pdays        45211 non-null  int64
 14  previous     45211 non-null  int64
 15  poutcome     8252 non-null   object
 16  y            45211 non-null  object
dtypes: int64(7), object(10)
memory usage: 5.9+ MB
```

In [66]:
```
Banking_data.describe()
```

Out[66]:

	age	balance	day_of_week	duration	campaign	pdays	
count	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	4521
mean	40.936210	1362.272058	15.806419	258.163080	2.763841	40.197828	
std	10.618762	3044.765829	8.322476	257.527812	3.098021	100.128746	
min	18.000000	-8019.000000	1.000000	0.000000	1.000000	-1.000000	
25%	33.000000	72.000000	8.000000	103.000000	1.000000	-1.000000	
50%	39.000000	448.000000	16.000000	180.000000	2.000000	-1.000000	
75%	48.000000	1428.000000	21.000000	319.000000	3.000000	-1.000000	
max	95.000000	102127.000000	31.000000	4918.000000	63.000000	871.000000	27

Frequency distribution of values in variables

In [67]:
```
col_names = Banking_data.columns


for col in col_names:

    print(df[col].value_counts())
```

```
age
32    2085
31    1996
33    1972
34    1930
35    1894
      ...
93       2
90       2
95       2
88       2
94       1
Name: count, Length: 77, dtype: int64
job
blue-collar      9732
management       9458
technician       7597
admin.           5171
services         4154
retired          2264
self-employed    1579
entrepreneur     1487
unemployed       1303
housemaid        1240
student           938
Name: count, dtype: int64
marital
married     27214
single      12790
divorced     5207
Name: count, dtype: int64
education
secondary    23202
tertiary     13301
primary       6851
Name: count, dtype: int64
default
no     44396
yes      815
Name: count, dtype: int64
balance
 0        3514
 1         195
 2         156
 4         139
 3         134
          ...
-381         1
 4617        1
 20584       1
 4358        1
 16353       1
Name: count, Length: 7168, dtype: int64
housing
yes    25130
no     20081
Name: count, dtype: int64
loan
no     37967
```

```
yes     7244
Name: count, dtype: int64
contact
cellular     29285
telephone     2906
Name: count, dtype: int64
day_of_week
20    2752
18    2308
21    2026
17    1939
6     1932
5     1910
14    1848
8     1842
28    1830
7     1817
19    1757
29    1745
15    1703
12    1603
13    1585
30    1566
9     1561
11    1479
4     1445
16    1415
2     1293
27    1121
3     1079
26    1035
23     939
22     905
25     840
31     643
10     524
24     447
1      322
Name: count, dtype: int64
month
may    13766
jul     6895
aug     6247
jun     5341
nov     3970
apr     2932
feb     2649
jan     1403
oct      738
sep      579
mar      477
dec      214
Name: count, dtype: int64
duration
124     188
90      184
89      177
104     175
122     175
```

```
       ...
1833      1
1545      1
1352      1
1342      1
1556      1
Name: count, Length: 1573, dtype: int64
campaign
1     17544
2     12505
3      5521
4      3522
5      1764
6      1291
7       735
8       540
9       327
10      266
11      201
12      155
13      133
14       93
15       84
16       79
17       69
18       51
19       44
20       43
21       35
22       23
25       22
23       22
24       20
29       16
28       16
26       13
31       12
27       10
32        9
30        8
33        6
34        5
36        4
35        4
43        3
38        3
37        2
50        2
41        2
46        1
58        1
55        1
63        1
51        1
39        1
44        1
Name: count, dtype: int64
pdays
-1      36954
```

```
 182      167
 92       147
 91       126
 183      126
         ...
 449        1
 452        1
 648        1
 595        1
 530        1
Name: count, Length: 559, dtype: int64
previous
0      36954
1       2772
2       2106
3       1142
4        714
5        459
6        277
7        205
8        129
9         92
10        67
11        65
12        44
13        38
15        20
14        19
17        15
16        13
19        11
20         8
23         8
18         6
22         6
24         5
27         5
21         4
29         4
25         4
30         3
38         2
37         2
26         2
28         2
51         1
275        1
58         1
32         1
40         1
55         1
35         1
41         1
Name: count, dtype: int64
poutcome
failure    4901
other      1840
success    1511
Name: count, dtype: int64
```

```
y
0    39922
1     5289
Name: count, dtype: int64
```

This is our target variable

In [68]:
```
Banking_data['y'].value_counts()
```

Out[68]:
```
y
no     39922
yes     5289
Name: count, dtype: int64
```

We will drop rows with missing target variables

In [70]:
```
Banking_data= Banking_data.dropna(subset=['y'])
Banking_data.head()
```

Out[70]:

	age	job	marital	education	default	balance	housing	loan	contact	day_of_week
0	58	management	married	tertiary	no	2143	yes	no	NaN	5
1	44	technician	single	secondary	no	29	yes	no	NaN	5
2	33	entrepreneur	married	secondary	no	2	yes	yes	NaN	5
3	47	blue-collar	married	NaN	no	1506	yes	no	NaN	5
4	33	NaN	single	NaN	no	1	no	no	NaN	5

We will Encode the target variable 'y'

In [82]:
```
le = LabelEncoder()
Banking_data['y'] = le.fit_transform(Banking_data['y'])
Banking_data['y'].head()
```

Out[82]:
```
0    0
1    0
2    0
3    0
4    0
Name: y, dtype: int64
```

We will Declare feature vector and target variable

In [81]:
```
X = Banking_data.drop('y', axis=1)
y = Banking_data['y']
print(X.head())
print(y.head())
```

```
   age           job  marital  education default  balance housing loan  \
0   58    management  married   tertiary      no     2143     yes   no
1   44    technician   single  secondary      no       29     yes   no
2   33  entrepreneur  married  secondary      no        2     yes  yes
3   47   blue-collar  married        NaN      no     1506     yes   no
4   33           NaN   single        NaN      no        1      no   no

  contact  day_of_week month  duration  campaign  pdays  previous poutcome
0     NaN            5   may       261         1     -1         0      NaN
1     NaN            5   may       151         1     -1         0      NaN
2     NaN            5   may        76         1     -1         0      NaN
3     NaN            5   may        92         1     -1         0      NaN
4     NaN            5   may       198         1     -1         0      NaN
0    0
1    0
2    0
3    0
4    0
Name: y, dtype: int64
```

We will Encode categorical variables

In [84]:
```
X_encoded = pd.get_dummies(X_imputed)
X_encoded.head()
```

Out[84]:

	age_18	age_19	age_20	age_21	age_22	age_23	age_24	age_25	age_26	age_27	...	previou
0	False	False	False	False	False	False	False	False	False	False	...	
1	False	False	False	False	False	False	False	False	False	False	...	
2	False	False	False	False	False	False	False	False	False	False	...	
3	False	False	False	False	False	False	False	False	False	False	...	
4	False	False	False	False	False	False	False	False	False	False	...	

5 rows × 9537 columns

We will Split data into separate training and test set

In [85]:
```
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.2, ra
X_train.shape, X_test.shape
```

Out[85]:
```
((36168, 9537), (9043, 9537))
```

Decision Tree Classifier with criterion gini index

In [86]:
```
clf_gini = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=0)
```

In [87]:
```
print(clf_gini)
```

```
DecisionTreeClassifier(max_depth=3, random_state=0)
```

In [88]:
```
clf_gini.fit(X_train, y_train)
```

Out[88]:
```
▾                DecisionTreeClassifier

DecisionTreeClassifier(max_depth=3, random_state=0)
```

Predict the Test set results with criterion gini index

In [89]:
```
y_pred_gini = clf_gini.predict(X_test)
```

Check accuracy score with criterion gini index

In [91]:
```
print('Model accuracy score with criterion gini index: {0:0.4f}'. format(accuracy_s
```

```
Model accuracy score with criterion gini index: 0.8904
```

Compare the train-set and test-set accuracy

In [92]:
```
y_pred_train_gini = clf_gini.predict(X_train)

y_pred_train_gini
```

Out[92]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [93]:
```
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pre
```

```
Training-set accuracy score: 0.8943
```

Check for overfitting and underfitting

In [94]:
```
print('Training set score: {:.4f}'.format(clf_gini.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(clf_gini.score(X_test, y_test)))
```

```
Training set score: 0.8943
Test set score: 0.8904
```

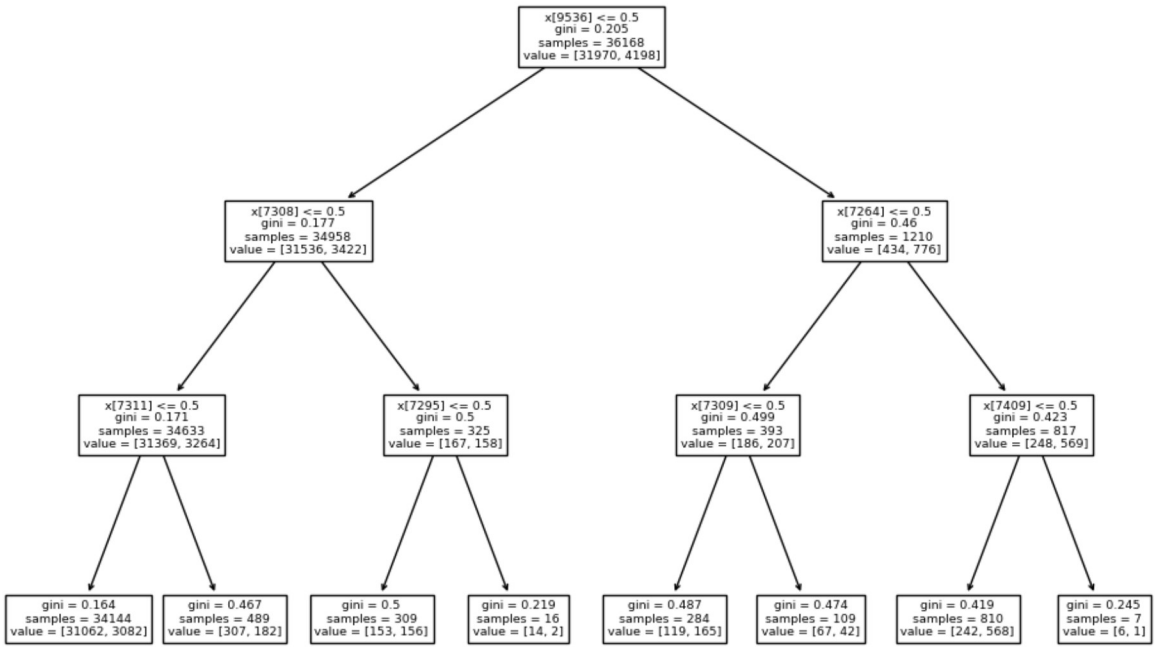
Visualize decision-trees

In [98]:
```
plt.figure(figsize=(12,8))

from sklearn import tree

tree.plot_tree(clf_gini.fit(X_train, y_train))
```

Out[98]:
```
[Text(0.5, 0.875, 'x[9536] <= 0.5\ngini = 0.205\nsamples = 36168\nvalue = [31970, 4
198]'),
 Text(0.25, 0.625, 'x[7308] <= 0.5\ngini = 0.177\nsamples = 34958\nvalue = [31536,
3422]'),
 Text(0.125, 0.375, 'x[7311] <= 0.5\ngini = 0.171\nsamples = 34633\nvalue = [31369,
3264]'),
 Text(0.0625, 0.125, 'gini = 0.164\nsamples = 34144\nvalue = [31062, 3082]'),
 Text(0.1875, 0.125, 'gini = 0.467\nsamples = 489\nvalue = [307, 182]'),
 Text(0.375, 0.375, 'x[7295] <= 0.5\ngini = 0.5\nsamples = 325\nvalue = [167, 15
8]'),
 Text(0.3125, 0.125, 'gini = 0.5\nsamples = 309\nvalue = [153, 156]'),
 Text(0.4375, 0.125, 'gini = 0.219\nsamples = 16\nvalue = [14, 2]'),
 Text(0.75, 0.625, 'x[7264] <= 0.5\ngini = 0.46\nsamples = 1210\nvalue = [434, 77
6]'),
 Text(0.625, 0.375, 'x[7309] <= 0.5\ngini = 0.499\nsamples = 393\nvalue = [186, 20
7]'),
 Text(0.5625, 0.125, 'gini = 0.487\nsamples = 284\nvalue = [119, 165]'),
 Text(0.6875, 0.125, 'gini = 0.474\nsamples = 109\nvalue = [67, 42]'),
 Text(0.875, 0.375, 'x[7409] <= 0.5\ngini = 0.423\nsamples = 817\nvalue = [248, 56
9]'),
 Text(0.8125, 0.125, 'gini = 0.419\nsamples = 810\nvalue = [242, 568]'),
 Text(0.9375, 0.125, 'gini = 0.245\nsamples = 7\nvalue = [6, 1]')]
```



Decision Tree Classifier with criterion entropy

Same Steps are repeated for entropy criterion

In [109…
```
clf_en = DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)


# fit the model
clf_en.fit(X_train, y_train)
```

Out[109]:
```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)
```

In [110…
```
y_pred_en = clf_en.predict(X_test)
```

In [111…
```
from sklearn.metrics import accuracy_score

print('Model accuracy score with criterion entropy: {0:0.4f}'. format(accuracy_scor
```

```
Model accuracy score with criterion entropy: 0.8910
```

In [112…
```
y_pred_train_en = clf_en.predict(X_train)

y_pred_train_en
```

Out[112]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [113…
```
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pre
```

```
Training-set accuracy score: 0.8941
```

In [114…
```
print('Training set score: {:.4f}'.format(clf_en.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(clf_en.score(X_test, y_test)))
```

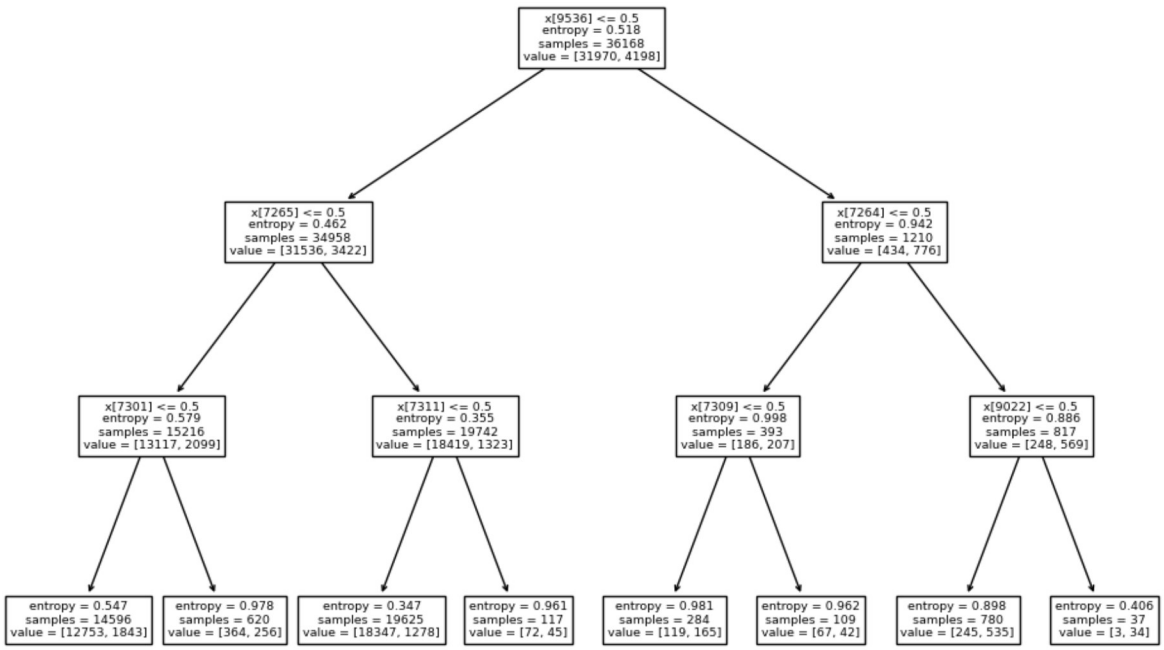
```
Training set score: 0.8941
Test set score: 0.8910
```

In [115…
```
plt.figure(figsize=(12,8))

from sklearn import tree

tree.plot_tree(clf_en.fit(X_train, y_train))
```

Out[115]:
```
[Text(0.5, 0.875, 'x[9536] <= 0.5\nentropy = 0.518\nsamples = 36168\nvalue = [3197
0, 4198]'),
 Text(0.25, 0.625, 'x[7265] <= 0.5\nentropy = 0.462\nsamples = 34958\nvalue = [315
36, 3422]'),
 Text(0.125, 0.375, 'x[7301] <= 0.5\nentropy = 0.579\nsamples = 15216\nvalue = [13
117, 2099]'),
 Text(0.0625, 0.125, 'entropy = 0.547\nsamples = 14596\nvalue = [12753, 1843]'),
 Text(0.1875, 0.125, 'entropy = 0.978\nsamples = 620\nvalue = [364, 256]'),
 Text(0.375, 0.375, 'x[7311] <= 0.5\nentropy = 0.355\nsamples = 19742\nvalue = [18
419, 1323]'),
 Text(0.3125, 0.125, 'entropy = 0.347\nsamples = 19625\nvalue = [18347, 1278]'),
 Text(0.4375, 0.125, 'entropy = 0.961\nsamples = 117\nvalue = [72, 45]'),
 Text(0.75, 0.625, 'x[7264] <= 0.5\nentropy = 0.942\nsamples = 1210\nvalue = [434,
776]'),
 Text(0.625, 0.375, 'x[7309] <= 0.5\nentropy = 0.998\nsamples = 393\nvalue = [186,
207]'),
 Text(0.5625, 0.125, 'entropy = 0.981\nsamples = 284\nvalue = [119, 165]'),
 Text(0.6875, 0.125, 'entropy = 0.962\nsamples = 109\nvalue = [67, 42]'),
 Text(0.875, 0.375, 'x[9022] <= 0.5\nentropy = 0.886\nsamples = 817\nvalue = [248,
569]'),
 Text(0.8125, 0.125, 'entropy = 0.898\nsamples = 780\nvalue = [245, 535]'),
 Text(0.9375, 0.125, 'entropy = 0.406\nsamples = 37\nvalue = [3, 34]')]
```



Confusion matrix

A confusion matrix is a tool for summarizing the performance of a classification algorithm. A confusion matrix will give us a clear picture of classification model performance and the types of errors produced by the model. It gives us a summary of correct and incorrect predictions broken down by each category. The summary is represented in a tabular form.

In [116…

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred_en)

print('Confusion matrix\n\n', cm)
```

```
Confusion matrix

 [[7863   89]
 [ 897  194]]
```

Classification Report

Classification report is another way to evaluate the classification model performance. It displays the precision, recall, f1 and support scores for the model.

In [117…

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred_en))
```

	precision	recall	f1-score	support
0	0.90	0.99	0.94	7952
1	0.69	0.18	0.28	1091
accuracy			0.89	9043
macro avg	0.79	0.58	0.61	9043
weighted avg	0.87	0.89	0.86	9043

Results and Interpretation

1.In this project, I build a Decision-Tree Classifier model to predict the whether a customer will purchase a product or not. I build two models, one with criterion gini index and another one with criterion entropy. The model yields a very good performance as indicated by the model accuracy in both the cases which was found to be 0.89.

2.In the model with criterion gini index, the training-set accuracy score is 0.8943 while the test-set accuracy to be 0.8904. These two values are quite comparable. So, there is no sign of overfitting.

3.Similarly, in the model with criterion entropy, the training-set accuracy score is 0.8941 while the test-set accuracy to be 0.8910.We get the same values as in the case with criterion gini. So, there is no sign of overfitting.

4.In both the cases, the training-set and test-set accuracy score is the same. It may happen because of small dataset.

5.The confusion matrix and classification report yields very good model performance.

Thank You For Reading!