In [1]:

```
import pandas as pd
import numpy as np
import math
```

In [12]:

```
df = pd.read_excel('datamining.xlsx')
df
```

Out[12]:

	age	income	student	credit_rating	buys_computer
0	youth	high	no	fair	no
1	youth	high	no	excellent	no
2	middle	high	no	fair	yes
3	senior	med	no	fair	yes
4	senior	low	yes	fair	yes
5	senior	low	yes	excellent	no
6	middle	low	yes	excellent	yes
7	youth	med	no	fair	no
8	youth	low	yes	fair	yes
9	senior	med	yes	fair	yes
10	youth	med	yes	excellent	yes
11	middle	med	no	excellent	yes
12	middle	high	yes	fair	yes
13	senior	med	no	excellent	no

In [145]:

```
df.buys_computer.value_counts()
```

Out[145]:

```
yes    9
no     5
Name: buys_computer, dtype: int64
```

In [20]:

```
total_yes = df.buys_computer.value_counts()[0]
total_no = df.buys_computer.value_counts()[1]
total = df.buys_computer.count()
```

Function to calculate Info(a,b):

In [44]:

```
def info(a,b):
    sum = a+b
    return round((-(a/sum) * math.log(a/sum,2) - (b/sum) * math.log(b/sum,2)), 3
```

The expected info needed to classify a tuple in D:

In [47]:
```
info_D = info(total_yes, total_no)
print(info_D,  ' bits')    # in bits
```

```
0.94  bits
```

Exploring income column:

In [144]:
```
df.income.value_counts()
```

Out[144]:
```
med     6
low     4
high    4
Name: income, dtype: int64
```

In [120]:
```
df_income = df[['income', 'buys_computer']]
df_income
```

Out[120]:

	income	buys_computer
0	high	no
1	high	no
2	high	yes
3	med	yes
4	low	yes
5	low	no
6	low	yes
7	med	no
8	low	yes
9	med	yes
10	med	yes
11	med	yes
12	high	yes
13	med	no

Visualizing the data:

In [158]:
```
table = pd.crosstab(df_income.income, df_income.buys_computer, margins = True)
table
```

Out[158]:

buys_computer	no	yes	All
income			
high	2	2	4
low	1	3	4
med	2	4	6
All	5	9	14

Grouping the classes of income with respect to the 'buys_computer' attribute.

In [124]:
```
high_yes =len( (np.where((df_income.income == 'high') & (df_income.buys_computer
high_no = len( (np.where((df_income.income == 'high') & (df_income.buys_computer

low_yes =len( (np.where((df_income.income == 'low') & (df_income.buys_computer =
low_no = len( (np.where((df_income.income == 'low') & (df_income.buys_computer =

med_yes =len( (np.where((df_income.income == 'med') & (df_income.buys_computer =
med_no = len( (np.where((df_income.income == 'med') & (df_income.buys_computer =
```

In [146]:
```
total_high = len(df_income[df_income.income == 'high'])
total_low = len(df_income[df_income.income == 'low'])
total_med = len(df_income[df_income.income == 'med'])
```

In [165]:
```
print(high_no, high_yes, total_high)
print(low_no, low_yes, total_low)
print(med_no, med_yes, total_med)
```

```
2 2 4
1 3 4
2 4 6
```

Alternative Method to do so:

- We use the crosstable method in pandas.

In [162]:

```
HIGH_NO,HIGH_YES,TOTAL_HIGH = table.values[0][0], table.values[0][1], table.valu
LOW_NO,LOW_YES,TOTAL_LOW = table.values[1][0], table.values[1][1], table.values[
MED_NO,MED_YES,TOTAL_MED = table.values[2][0], table.values[2][1], table.values[
```

In [163]:

```
print(HIGH_NO,HIGH_YES,TOTAL_HIGH)
print(LOW_NO,LOW_YES,TOTAL_LOW)
print(MED_NO,MED_YES,TOTAL_MED)
```

```
2 2 4
1 3 4
2 4 6
```

The expected information needed to classify a tuple in D if the tuples are partioned according to income:

In [166]:

```
info_income_D = np.array([(total_high/total)*info(high_yes,high_no),
                (total_low/total)*info(low_yes,low_no),
                (total_med/total)*info(med_yes,med_no)])
info_income_D = round(sum(info_income_D), 3)

print(info_income_D, ' bits')
```

```
0.911  bits
```

The Gain by branching on 'income':

In [174]:

```
Gain_income = info_D - info_income_D
print(Gain_income.round(3), ' bits')
```

```
0.029  bits
```

Another case:

When tuples are partitioned as per the 'student' attribute:

In [168]:
```
### Visualizing the data:

table1 = pd.crosstab(df.student, df_income.buys_computer, margins = True)
table1
```

Out[168]:

buys_computer	no	yes	All
student			
no	4	3	7
yes	1	6	7
All	5	9	14

In [169]:
```
std0_no, std0_yes, std0_total = table1.values[0][0], table1.values[0][1], table1
std1_no, std1_yes, std1_total = table1.values[1][0], table1.values[1][1], table1
```

In [170]:
```
print(std0_no, std0_yes, std0_total)
print(std1_no, std1_yes, std1_total)
```

```
4 3 7
1 6 7
```

The expected information needed to classify a tuple in D if the tuples are partioned according to student:

In [172]:
```
info_student_D = (std0_total/total)*info(std0_no, std0_yes)+(std1_total/total)*i
info_student_D = round(info_student_D, 3)

print(info_student_D, ' bits')
```

```
0.788  bits
```

The Gain by branching on 'student':

In [173]:
```
Gain_student = info_D - info_student_D
print(Gain_student.round(3), ' bits')
```

```
0.152  bits
```

Another case:

When tuples are partitioned as per the 'credit_rating' attribute:

In [176]:

```
### Visualizing the data:

table2 = pd.crosstab(df.credit_rating, df_income.buys_computer, margins = True)
table2
```

Out[176]:

buys_computer credit_rating	no	yes	All
excellent	3	3	6
fair	2	6	8
All	5	9	14

In [177]:

```
exe_no, exe_yes, exe_total = table2.values[0][0], table2.values[0][1], table2.va
fair_no, fair_yes, fair_total = table2.values[1][0], table2.values[1][1], table2
```

In [178]:

```
print(exe_no, exe_yes, exe_total)
print(fair_no, fair_yes, fair_total)
```

```
3 3 6
2 6 8
```

The expected information needed to classify a tuple in D if the tuples are partioned according to 'credit_rating':

In [179]:

```
info_credit_D = (exe_total/total)*info(exe_no, exe_yes)+(fair_total/total)*info(
info_credit_D = round(info_credit_D, 3)

print(info_credit_D, ' bits')
```

```
0.892  bits
```

The Gain by branching on 'student':

In [180]:

```
Gain_credit_rating = info_D - info_credit_D
print(Gain_credit_rating.round(3), ' bits')
```

```
0.048  bits
```

The End.

Prepared by: Sagun Shakya.