1. Import Libraries

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
```

In [2]:

```
fraud_check = pd.read_csv('Fraud_check.csv')
fraud_check
```

Out[2]:

	Undergrad	Marital.Status	Taxable.Income	City.Population	Work.Experience	Urban
0	NO	Single	68833	50047	10	YES
1	YES	Divorced	33700	134075	18	YES
2	NO	Married	36925	160205	30	YES
3	YES	Single	50190	193264	15	YES
4	NO	Married	81002	27533	28	NO
...	...	...	...	...	...	...
595	YES	Divorced	76340	39492	7	YES
596	YES	Divorced	69967	55369	2	YES
597	NO	Divorced	47334	154058	0	YES
598	YES	Married	98592	180083	17	NO
599	NO	Divorced	96519	158137	16	NO

600 rows × 6 columns

In [3]:

```
fraud_check
```

Out[3]:

	Undergrad	Marital.Status	Taxable.Income	City.Population	Work.Experience	Urban
0	NO	Single	68833	50047	10	YES
1	YES	Divorced	33700	134075	18	YES
2	NO	Married	36925	160205	30	YES
3	YES	Single	50190	193264	15	YES
4	NO	Married	81002	27533	28	NO
...	...	...	...	...	...	...
595	YES	Divorced	76340	39492	7	YES
596	YES	Divorced	69967	55369	2	YES
597	NO	Divorced	47334	154058	0	YES
598	YES	Married	98592	180083	17	NO
599	NO	Divorced	96519	158137	16	NO

600 rows × 6 columns

In [4]:

```
fraud_check.shape
```

Out[4]:

```
(600, 6)
```

In [5]:

```
fraud_check.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 600 entries, 0 to 599
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Undergrad        600 non-null    object
 1   Marital.Status   600 non-null    object
 2   Taxable.Income   600 non-null    int64 
 3   City.Population  600 non-null    int64 
 4   Work.Experience  600 non-null    int64 
 5   Urban            600 non-null    object
dtypes: int64(3), object(3)
memory usage: 28.2+ KB
```

In [6]:

```
fraud_check.dtypes
```

Out[6]:

```
Undergrad          object
Marital.Status     object
Taxable.Income      int64
City.Population     int64
Work.Experience     int64
Urban              object
dtype: object
```

In [7]:

```
fraud_check.isna().sum()
```

Out[7]:

```
Undergrad          0
Marital.Status     0
Taxable.Income     0
City.Population    0
Work.Experience    0
Urban              0
dtype: int64
```

In [8]:

```
fraud_check['Taxable.Income'].max()
```

Out[8]:

```
99619
```

In [9]:

```
fraud_check['Taxable.Income'].min()
```

Out[9]:

```
10003
```

In [10]:

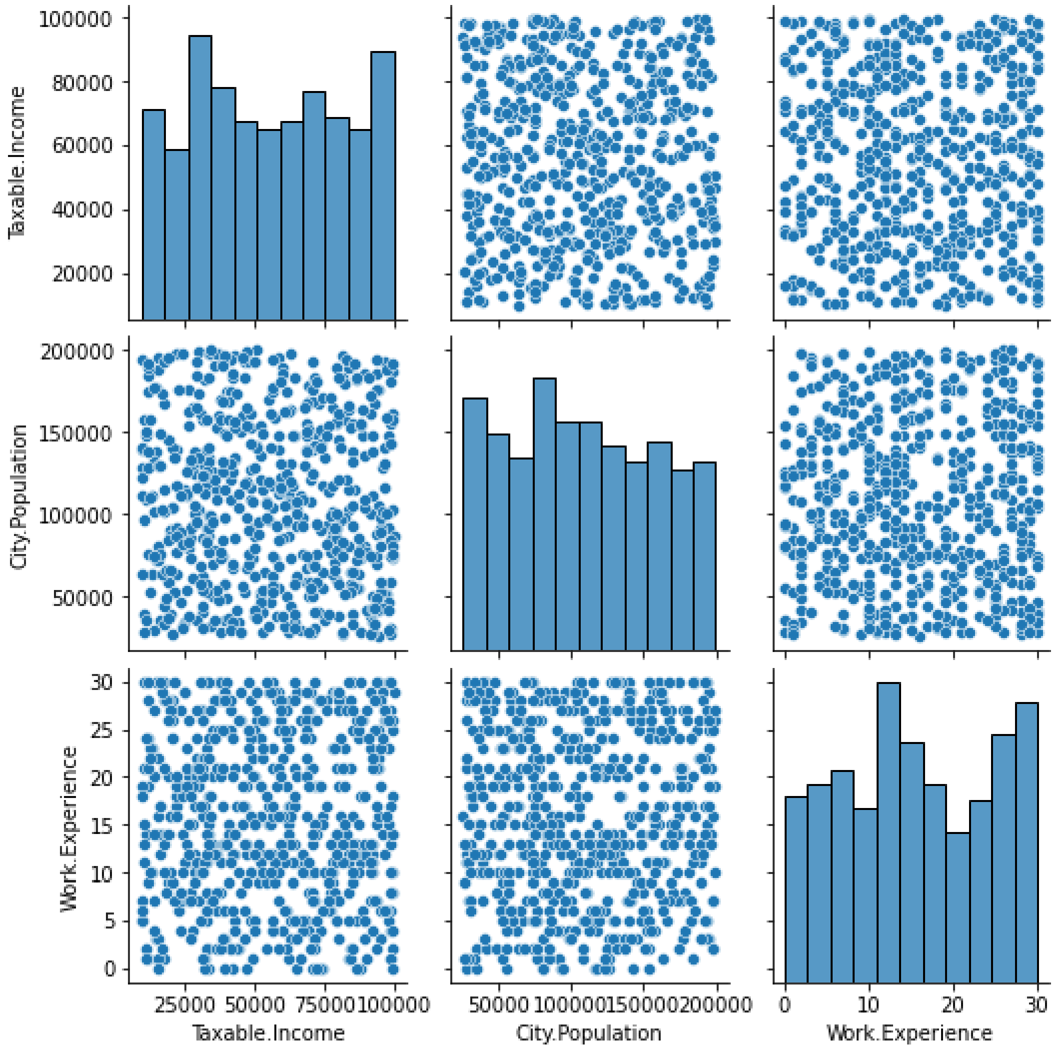
```
fraud_check.describe(include='all')
```

Out[10]:

	Undergrad	Marital.Status	Taxable.Income	City.Population	Work.Experience	Urban
count	600	600	600.000000	600.000000	600.000000	600
unique	2	3	NaN	NaN	NaN	2
top	YES	Single	NaN	NaN	NaN	YES
freq	312	217	NaN	NaN	NaN	302
mean	NaN	NaN	55208.375000	108747.368333	15.558333	NaN
std	NaN	NaN	26204.827597	49850.075134	8.842147	NaN
min	NaN	NaN	10003.000000	25779.000000	0.000000	NaN
25%	NaN	NaN	32871.500000	66966.750000	8.000000	NaN
50%	NaN	NaN	55074.500000	106493.500000	15.000000	NaN
75%	NaN	NaN	78611.750000	150114.250000	24.000000	NaN
max	NaN	NaN	99619.000000	199778.000000	30.000000	NaN

In [11]:

```
sns.pairplot(fraud_check)
fraud_check["TaxInc"] = pd.cut(fraud_check["Taxable.Income"],bins = [10003, 30000, 99620],l
fraud_check_2 = fraud_check.drop(columns=["Taxable.Income"])
```



In [12]:

```
fraud_check_2 = pd.get_dummies(fraud_check_2.drop(columns=["TaxInc"]))
fraud_check_2
```

Out[12]:

	City.Population	Work.Experience	Undergrad_NO	Undergrad_YES	Marital.Status_Divorced
0	50047	10	1	0	0
1	134075	18	0	1	1
2	160205	30	1	0	0
3	193264	15	0	1	0
4	27533	28	1	0	0
...	...	...	...	...	...
595	39492	7	0	1	1
596	55369	2	0	1	1
597	154058	0	1	0	1
598	180083	17	0	1	0
599	158137	16	1	0	1

600 rows × 9 columns

In [13]:

```
fraud_check_final = pd.concat([fraud_check_2, fraud_check["TaxInc"]],axis=1)
fraud_check_final.dtypes
```

Out[13]:

```
City.Population              int64
Work.Experience              int64
Undergrad_NO                 uint8
Undergrad_YES                uint8
Marital.Status_Divorced      uint8
Marital.Status_Married       uint8
Marital.Status_Single        uint8
Urban_NO                     uint8
Urban_YES                    uint8
TaxInc                     float64
dtype: object
```

In [14]:

```
fraud_check_final
```

Out[14]:

	City.Population	Work.Experience	Undergrad_NO	Undergrad_YES	Marital.Status_Divorced
0	50047	10	1	0	0
1	134075	18	0	1	1
2	160205	30	1	0	0
3	193264	15	0	1	0
4	27533	28	1	0	0
...	...	...	...	...	...
595	39492	7	0	1	1
596	55369	2	0	1	1
597	154058	0	1	0	1
598	180083	17	0	1	0
599	158137	16	1	0	1

600 rows × 10 columns

In [15]:

```
fraud_check_final.isnull().sum()
```

Out[15]:

```
City.Population            0
Work.Experience            0
Undergrad_NO               0
Undergrad_YES              0
Marital.Status_Divorced    0
Marital.Status_Married     0
Marital.Status_Single      0
Urban_NO                   0
Urban_YES                  0
TaxInc                     1
dtype: int64
```

In [16]:

```
# Create a DataFrame from dictionary

kl = pd.DataFrame(fraud_check_final)
kl
```

Out[16]:

	City.Population	Work.Experience	Undergrad_NO	Undergrad_YES	Marital.Status_Divorced
0	50047	10	1	0	0
1	134075	18	0	1	1
2	160205	30	1	0	0
3	193264	15	0	1	0
4	27533	28	1	0	0
...	...	...	...	...	...
595	39492	7	0	1	1
596	55369	2	0	1	1
597	154058	0	1	0	1
598	180083	17	0	1	0
599	158137	16	1	0	1

600 rows × 10 columns

In [17]:

```
#Finding the mean of the column having NaN
mean_value=kl['TaxInc'].median()
mean_value
```

Out[17]:

```
1.0
```

In [18]:

```
# Replace NaNs in column TaxInc with the
# mean of values in the same column
kl['TaxInc'].fillna(value=mean_value, inplace=True)
print('Updated Dataframe:')
print(kl)
```

```
Updated Dataframe:
     City.Population  Work.Experience  Undergrad_NO  Undergrad_YES  \
0              50047               10             1              0
1             134075               18             0              1
2             160205               30             1              0
3             193264               15             0              1
4              27533               28             1              0
..               ...              ...           ...            ...
595            39492                7             0              1
596            55369                2             0              1
597           154058                0             1              0
598           180083               17             0              1
599           158137               16             1              0

     Marital.Status_Divorced  Marital.Status_Married  Marital.Status_Single
\
0                          0                       0                      1
1                          1                       0                      0
2                          0                       1                      0
3                          0                       0                      1
4                          0                       1                      0
..                       ...                     ...                    ...
595                        1                       0                      0
596                        1                       0                      0
597                        1                       0                      0
598                        0                       1                      0
599                        1                       0                      0

     Urban_NO  Urban_YES  TaxInc
0           0          1     1.0
1           0          1     1.0
2           0          1     1.0
3           0          1     1.0
4           1          0     1.0
..        ...        ...     ...
595         0          1     1.0
596         0          1     1.0
597         0          1     1.0
598         1          0     1.0
599         1          0     1.0

[600 rows x 10 columns]
```

In [19]:

```
kl.dtypes
```

Out[19]:

```
City.Population              int64
Work.Experience              int64
Undergrad_NO                 uint8
Undergrad_YES                uint8
Marital.Status_Divorced      uint8
Marital.Status_Married       uint8
Marital.Status_Single        uint8
Urban_NO                     uint8
Urban_YES                    uint8
TaxInc                     float64
dtype: object
```

In [20]:

```
# converting 'Weight' from float to int
kl['TaxInc'] = kl['TaxInc'].astype(int)
```

In [21]:

```
# displaying the datatypes
display(kl.dtypes)
```

```
City.Population            int64
Work.Experience            int64
Undergrad_NO               uint8
Undergrad_YES              uint8
Marital.Status_Divorced    uint8
Marital.Status_Married     uint8
Marital.Status_Single      uint8
Urban_NO                   uint8
Urban_YES                  uint8
TaxInc                     int32
dtype: object
```

In [22]:

```
kl.isnull().sum()
```

Out[22]:

```
City.Population            0
Work.Experience            0
Undergrad_NO               0
Undergrad_YES              0
Marital.Status_Divorced    0
Marital.Status_Married     0
Marital.Status_Single      0
Urban_NO                   0
Urban_YES                  0
TaxInc                     0
dtype: int64
```

In [23]:

```
colnames =list(kl.columns)
```

In [24]:

```
colnames
```

Out[24]:

```
['City.Population',
 'Work.Experience',
 'Undergrad_NO',
 'Undergrad_YES',
 'Marital.Status_Divorced',
 'Marital.Status_Married',
 'Marital.Status_Single',
 'Urban_NO',
 'Urban_YES',
 'TaxInc']
```

In [25]:

```
predictors = colnames[:9]
predictors
target = colnames[9]
target
```

Out[25]:

```
'TaxInc'
```

In [26]:

```
# Separating independent and dependent Veriables
x = kl[predictors]
x.shape
y = kl[target]
```

In [27]:

```
x
```

Out[27]:

	City.Population	Work.Experience	Undergrad_NO	Undergrad_YES	Marital.Status_Divorced
0	50047	10	1	0	0
1	134075	18	0	1	1
2	160205	30	1	0	0
3	193264	15	0	1	0
4	27533	28	1	0	0
...	...	...	...	...	...
595	39492	7	0	1	1
596	55369	2	0	1	1
597	154058	0	1	0	1
598	180083	17	0	1	0
599	158137	16	1	0	1

600 rows × 9 columns

In [28]:

```
y
```

Out[28]:

```
0      1
1      1
2      1
3      1
4      1
      ..
595    1
596    1
597    1
598    1
599    1
Name: TaxInc, Length: 600, dtype: int32
```

2.Decision Tree Building

In [29]:

```
from sklearn.model_selection import train_test_split
train , test =train_test_split(kl ,test_size=0.3)
kl ['TaxInc'].unique()
```

Out[29]:

```
array([1, 0])
```

In [30]:

```
from sklearn.tree import DecisionTreeClassifier
dc_model = DecisionTreeClassifier(criterion = "entropy")
dc_model.fit(train[predictors], train[[target]])
```

Out[30]:

```
DecisionTreeClassifier(criterion='entropy')
```

In [31]:

```
#prediction
pred = dc_model.predict(test[predictors])
pred
type(pred)
```

Out[31]:

```
numpy.ndarray
```

In [32]:

```
pd.Series(pred).value_counts()
141/(141+39)
pd.crosstab(test[target],pred)
pd.Series(dc_model.predict(train[predictors])).reset_index(drop=True)
np.mean(pd.Series(train.TaxInc).reset_index(drop=True)==pd.Series(dc_model.predict(train[pr
np.mean(pred==test.TaxInc)
```

Out[32]:

```
0.6666666666666666
```

In [33]:

```
pd.crosstab(test[target],pred)  #64%
temp = pd.Series(dc_model.predict(train[predictors])).reset_index(drop=True)
np.mean(pd.Series(train.TaxInc).reset_index(drop=True)==pd.Series(dc_model.predict(train[pr
np.mean(pred==test.TaxInc)
```

Out[33]:

```
0.6666666666666666
```

In [34]:

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_jobs=3, oob_score=True, n_estimators=15,criterion=('gini'))
```

In [35]:

```
np.shape(kl)
```

Out[35]:

```
(600, 10)
```

In [36]:

```
np.shape(kl) # 600,100 => Shape
len(x)
```

Out[36]:

```
600
```

In [37]:

```
len(y)
```

Out[37]:

```
600
```

In [38]:

```
kl.describe()
```

Out[38]:

	City.Population	Work.Experience	Undergrad_NO	Undergrad_YES	Marital.Status_Divorced
count	600.000000	600.000000	600.000000	600.000000	600.00000(
mean	108747.368333	15.558333	0.480000	0.520000	0.31500(
std	49850.075134	8.842147	0.500017	0.500017	0.46490(
min	25779.000000	0.000000	0.000000	0.000000	0.00000(
25%	66966.750000	8.000000	0.000000	0.000000	0.00000(
50%	106493.500000	15.000000	0.000000	1.000000	0.00000(
75%	150114.250000	24.000000	1.000000	1.000000	1.00000(
max	199778.000000	30.000000	1.000000	1.000000	1.00000(

In [39]:

```
kl.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 600 entries, 0 to 599
Data columns (total 10 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   City.Population          600 non-null    int64
 1   Work.Experience          600 non-null    int64
 2   Undergrad_NO             600 non-null    uint8
 3   Undergrad_YES            600 non-null    uint8
 4   Marital.Status_Divorced  600 non-null    uint8
 5   Marital.Status_Married   600 non-null    uint8
 6   Marital.Status_Single    600 non-null    uint8
 7   Urban_NO                 600 non-null    uint8
 8   Urban_YES                600 non-null    uint8
 9   TaxInc                   600 non-null    int32
dtypes: int32(1), int64(2), uint8(7)
memory usage: 15.9 KB
```

In [40]:

```
type([x])
type([y])
y1 = pd.DataFrame(y)
type(y1)
import warnings
warnings.filterwarnings('ignore')
```

In [41]:

```
rf.fit(x,y1)
rf.predict(x)
```

Out[41]:

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0,
       1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1,
       1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
       1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0,
       1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1,
       0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 0, 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, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0,
       1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1])
```

In [44]:

```
kl['rf_pred']=rf.predict(x)
```

In [45]:

```
kl
```

Out[45]:

<.Experience	Undergrad_NO	Undergrad_YES	Marital.Status_Divorced	Marital.Status_Married	Marit
10	1	0	0	0	
18	0	1	1	0	
30	1	0	0	1	
15	0	1	0	0	
28	1	0	0	1	
...	...	...	...	...	
7	0	1	1	0	
2	0	1	1	0	
0	1	0	1	0	
17	0	1	0	1	
16	1	0	1	0	

In [47]:

```
cols=['rf_pred','TaxInc']
cols
```

Out[47]:

```
['rf_pred', 'TaxInc']
```

In [48]:

```
kl[cols].head()
```

Out[48]:

	rf_pred	TaxInc
0	1	1
1	1	1
2	1	1
3	1	1
4	1	1

In [49]:

```
kl['TaxInc']
```

Out[49]:

```
0      1
1      1
2      1
3      1
4      1
      ..
595    1
596    1
597    1
598    1
599    1
Name: TaxInc, Length: 600, dtype: int32
```

In [50]:

```
from sklearn.metrics import confusion_matrix
```

In [51]:

```
confusion_matrix(kl['TaxInc'],kl['rf_pred'])
```

Out[51]:

```
array([[117,   6],
       [  0, 477]], dtype=int64)
```

In [52]:

```
pd.crosstab(kl['TaxInc'],kl['rf_pred'])
```

Out[52]:

rf_pred	0	1
TaxInc		
0	117	6
1	0	477

In [53]:

```
kl['rf_pred']
```

Out[53]:

```
0      1
1      1
2      1
3      1
4      1
      ..
595    1
596    1
597    1
598    1
599    1
Name: rf_pred, Length: 600, dtype: int32
```

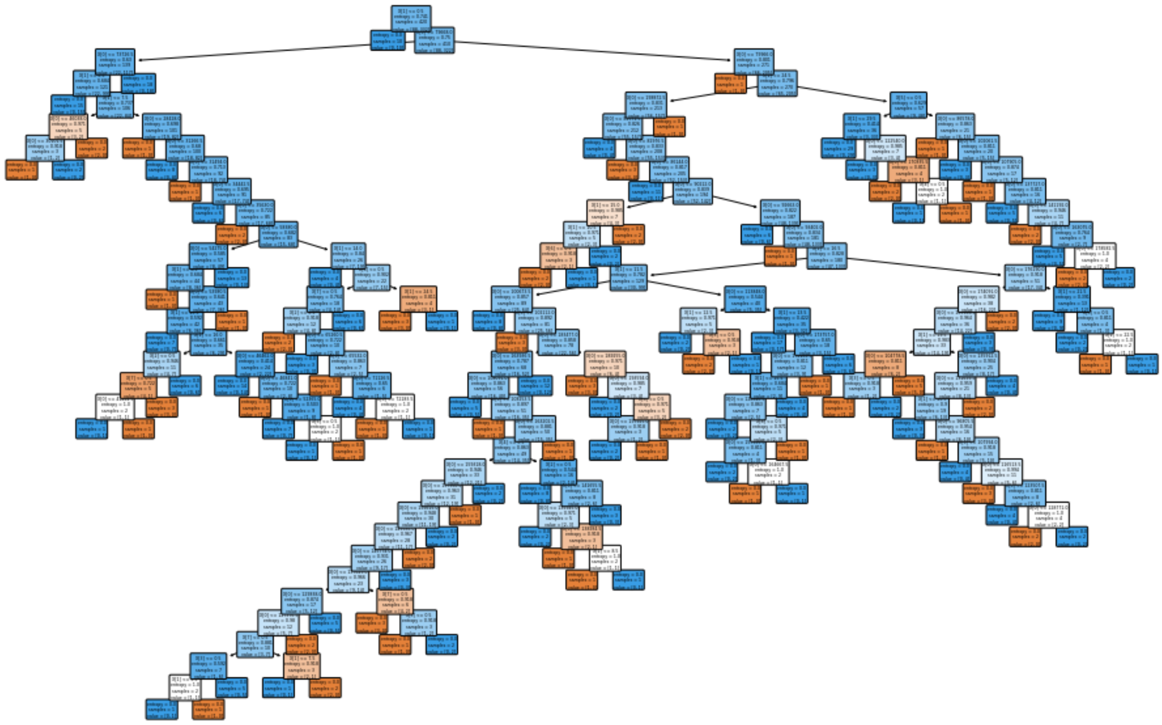
In [54]:

```
print('Accuracy',(477+117)/(477+117+6+0)*100)
```

```
Accuracy 99.0
```

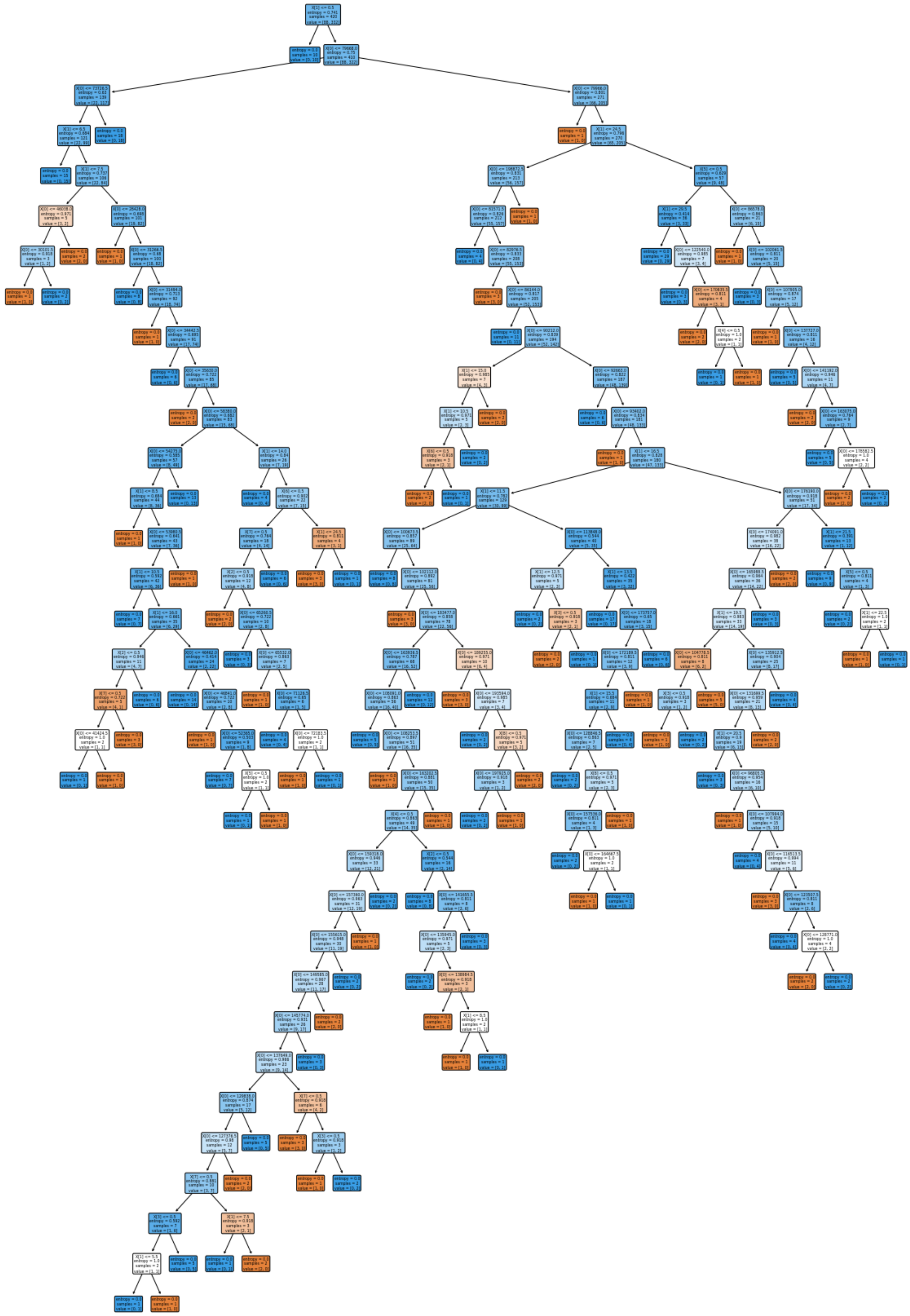
In [55]:

```
# prepare a plot figure with set size
from sklearn.tree import plot_tree
from matplotlib import pyplot as plt
plt.figure(figsize=(16,10))
plot_tree(dc_model,rounded=True,filled=True)
plt.show()
```



In [56]:

```
plt.figure(figsize=(20,30))
plot_tree(dc_model,rounded=True,filled=True)
plt.show()
```



In [ ]:

```

```