Objective of the problem statement

- Use decision trees to prepare a model on fraud data treating those who have taxable_income <= 30000 as "Risky" and others are "Good"

1. Import necessary libraries

In [56]:
```
import pandas as pd
```

2. Importing data

In [57]:
```
fc_data = pd.read_csv('Fraud_check.csv')
fc_data
```

Out[57]:

	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

3. Data Understanding

3.1 Initial Analysis

In [58]:
```
fc_data.shape
```

Out[58]:
```
(600, 6)
```

In [6]:
```
fc_data.dtypes
```

Out[6]:
```
Undergrad          object
Marital.Status     object
Taxable.Income      int64
City.Population     int64
Work.Experience     int64
Urban              object
dtype: object
```

In [59]:
```
fc_data.isna().sum()
```

Out[59]:
```
Undergrad          0
Marital.Status     0
Taxable.Income     0
City.Population    0
Work.Experience    0
Urban              0
dtype: int64
```

4. Data Preparation

In [60]:
```
from sklearn import preprocessing
```

In [61]:
```
# Create numerical variable for categorical data
label_encoder = preprocessing.LabelEncoder()
fc_data['Undergrad'] = label_encoder.fit_transform(fc_data['Undergrad'])
fc_data['Marital.Status'] = label_encoder.fit_transform(fc_data['Marital.Status'])
fc_data['Urban'] = label_encoder.fit_transform(fc_data['Urban'])
```

In [62]:
```
fc_data
```

Out[62]:

	Undergrad	Marital.Status	Taxable.Income	City.Population	Work.Experience	Urban
0	0	2	68833	50047	10	1
1	1	0	33700	134075	18	1
2	0	1	36925	160205	30	1
3	1	2	50190	193264	15	1
4	0	1	81002	27533	28	0
...	...	...	...	...	...	...
595	1	0	76340	39492	7	1
596	1	0	69967	55369	2	1
597	0	0	47334	154058	0	1
598	1	1	98592	180083	17	0
599	0	0	96519	158137	16	0

In [63]:
```
fc_data.head(20)
```

Out[63]:

	Undergrad	Marital.Status	Taxable.Income	City.Population	Work.Experience	Urban
0	0	2	68833	50047	10	1
1	1	0	33700	134075	18	1
2	0	1	36925	160205	30	1
3	1	2	50190	193264	15	1
4	0	1	81002	27533	28	0
5	0	0	33329	116382	0	0
6	0	0	83357	80890	8	1
7	1	2	62774	131253	3	1
8	0	2	83519	102481	12	1
9	1	0	98152	155482	4	1
10	0	2	29732	102602	19	1
11	0	2	61063	94875	6	1
12	0	0	11794	148033	14	1
13	0	1	61830	86649	16	1
14	0	1	64070	57529	13	1
15	0	0	69869	107764	29	0
16	1	0	24987	34551	29	0
17	1	1	39476	57194	25	0
18	1	0	97957	59269	6	0
19	0	2	10987	126953	30	1

In [18]:
```
fc_data.info
```

Out[18]:
```
<bound method DataFrame.info of      Undergrad  Marital.Status  Taxable.Income
City.Population  \
0              0               2           68833            50047
1              1               0           33700           134075
2              0               1           36925           160205
3              1               2           50190           193264
4              0               1           81002            27533
..           ...             ...             ...              ...
595            1               0           76340            39492
596            1               0           69967            55369
597            0               0           47334           154058
598            1               1           98592           180083
599            0               0           96519           158137

     Work.Experience  Urban
0                 10      1
1                 18      1
2                 30      1
3                 15      1
4                 28      0
..               ...    ...
595                7      1
596                2      1
597                0      1
598               17      0
599               16      0

[600 rows x 6 columns]>
```

5. Model Building

5.1 Separate input & output

In [64]:
```
X = fc_data.drop(labels = 'Taxable.Income', axis=1)
y = fc_data[['Taxable.Income']]
```

In [65]:
```
X
```

Out[65]:

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

600 rows × 5 columns

In [66]:
```
y
```

Out[66]:

	Taxable.Income
0	68833
1	33700
2	36925
3	50190
4	81002
...	...
595	76340
596	69967
597	47334
598	98592
599	96519

600 rows × 1 columns

5.2 Train test split

In [67]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size = 0.20, random_st
```

In [68]:
```
X_train
```

Out[68]:

	Undergrad	Marital.Status	City.Population	Work.Experience	Urban
82	0	0	111068	26	1
568	0	2	150036	22	1
347	0	1	80991	0	1
544	0	2	133877	21	1
34	1	0	183767	1	1
...	...	...	...	...	...
129	1	2	65469	26	0
144	1	2	156503	29	1
72	1	0	108300	27	1
235	0	0	87541	9	0
37	0	1	66912	5	1

In [69]:
```
# for training data
X_train.shape,y_train.shape
```

Out[69]:
```
((480, 5), (480, 1))
```

In [70]:
```
# for test data
X_test.shape,y_test.shape
```

Out[70]:
```
((120, 5), (120, 1))
```

6. Model training

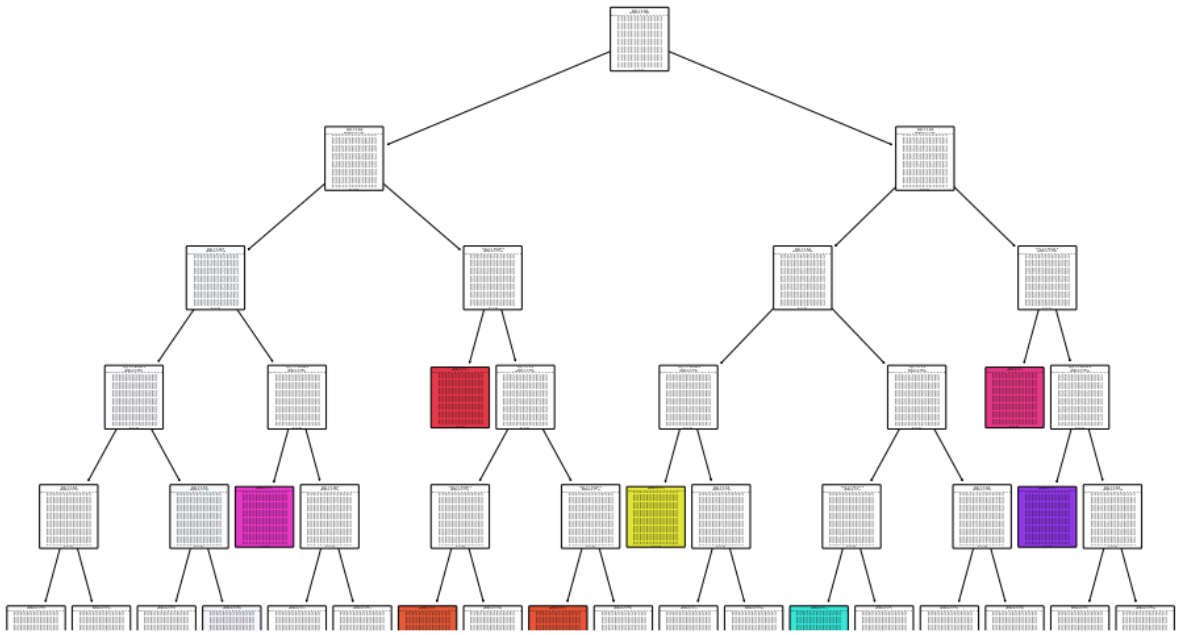
In [71]:
```
from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier(criterion='gini', max_depth=5)
dt_model.fit(X_train,y_train)
```

Out[71]:
```
DecisionTreeClassifier(max_depth=5)
```

Plot Tree

In [72]:
```
#Prepare a plot figure with set size
from sklearn.tree import plot_tree
from matplotlib import pyplot as plt
```

In [73]:
```
plt.figure(figsize=(16,10))
plot_tree(dt_model,rounded=True,filled=True)
plt.show()
```



7. Model Testing

In [29]:
```
#Training data
y_train_pred = dt_model.predict(X_train)
```

In [30]:
```
# Test data
y_test_pred = dt_model.predict(X_test)
```

8. Model Evaluation

In [31]:
```
from sklearn.metrics import accuracy_score,precision_score,recall_score,confusion
```

Training data

In [74]:
```
accuracy_score(y_train,y_train_pred)
```

Out[74]:
```
0.04583333333333333
```

In [75]:
```
confusion_matrix(y_train,y_train_pred)
```

Out[75]:
```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 1, 0, ..., 0, 0, 0],
       [0, 0, 1, ..., 0, 0, 0],
       ...,
       [0, 1, 0, ..., 0, 0, 0],
       [0, 0, 1, ..., 0, 0, 0],
       [0, 1, 0, ..., 0, 0, 0]], dtype=int64)
```

In [76]:
```
print(classification_report(y_train,y_train_pred))
```

```
54638       0.00      0.00      0.00         1
54850       0.00      0.00      0.00         1
55299       0.00      0.00      0.00         1
55308       0.00      0.00      0.00         1
55873       0.00      0.00      0.00         1
55981       0.00      0.00      0.00         1
56129       0.00      0.00      0.00         1
56166       0.00      0.00      0.00         1
56438       0.00      0.00      0.00         1
56501       0.00      0.00      0.00         1
56536       0.00      0.00      0.00         1
56555       0.00      0.00      0.00         1
56647       0.00      0.00      0.00         1
56974       0.00      0.00      0.00         1
57365       0.00      0.00      0.00         1
58199       0.00      0.00      0.00         1
58403       0.00      0.00      0.00         1
58451       0.00      0.00      0.00         1
58635       0.00      0.00      0.00         1
58751       0.00      0.00      0.00         1
```

For Testing Data

In [77]:
```
accuracy_score(y_test,y_test_pred)
```

Out[77]:
```
0.0
```

In [78]:
```
confusion_matrix(y_test,y_test_pred)
```

Out[78]:
```
array([[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, 0, ..., 0, 0, 0],
       [0, 1, 0, ..., 0, 0, 0]], dtype=int64)
```

In [80]:
```
print(classification_report(y_test,y_test_pred))
```

```
              precision    recall  f1-score   support

       10003       0.00      0.00      0.00       1.0
       10163       0.00      0.00      0.00       0.0
       10329       0.00      0.00      0.00       0.0
       10348       0.00      0.00      0.00       0.0
       10455       0.00      0.00      0.00       1.0
       11784       0.00      0.00      0.00       1.0
       11865       0.00      0.00      0.00       1.0
       12083       0.00      0.00      0.00       1.0
       12453       0.00      0.00      0.00       0.0
       12470       0.00      0.00      0.00       1.0
       12514       0.00      0.00      0.00       0.0
       13068       0.00      0.00      0.00       1.0
       14398       0.00      0.00      0.00       0.0
       14912       0.00      0.00      0.00       1.0
       15532       0.00      0.00      0.00       1.0
       15710       0.00      0.00      0.00       1.0
       16690       0.00      0.00      0.00       1.0
```

9 .Model Deployment

In [35]:
```
from pickle import dump
```

In [84]:
```
dump(dt_model,open('log_model.pkl','wb'))
```

In [85]:
```
from pickle import load
```

In [86]:
```
dt_model_pickle = load(open('log_model.pkl','rb'))
```

In [87]:
```
pickle_pred = dt_model_pickle.predict(X_test)
```

Conclusion:-We can see maximum depth of tree 5 is good as accuracy prospective & classsification is good technique for predict the sale & regression is not usual to good at this dataset