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"

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Importing Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
 # This is formatted as code
```

Reading the dataset

```
df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Fraud_check.csv")
```

```
df.head()
```

	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

```
df
```

	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

checking null values

```
df.isnull().sum()
```

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

```
df.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
```

```
df['Undergrad'].unique()
```

```
array(['NO', 'YES'], dtype=object)
```

```
df['Taxable.Income'].isnull().sum()
```

```
0
```

```
df['Taxable.Income'].unique()
```

```
array([68833, 33700, 36925, 50190, 81002, 33329, 83357, 62774, 83519,
       98152, 29732, 61063, 11794, 61830, 64070, 69869, 24987, 39476,
       97957, 10987, 88636, 14310, 78969, 92040, 38239, 31417, 55299,
       87778, 10379, 94033, 73854, 64007, 97200, 82071, 12514, 31336,
       10163, 68513, 14912, 74010, 50777, 49436, 96485, 70339, 33614,
       81079, 31532, 44034, 16264, 45706, 79616, 40137, 97703, 21917,
       81939, 19925, 34703, 78573, 10870, 65065, 94418, 26972, 91684,
       51685, 62158, 58312, 42507, 61486, 10150, 99619, 89302, 94287,
       41344, 41382, 32434, 61105, 67628, 31458, 73411, 10329, 44947,
       35610, 66564, 87348, 69098, 28449, 39277, 13747, 86652, 26741,
       77634, 99325, 85651, 15296, 10348, 41395, 22258, 79583, 22187,
       52495, 93078, 35072, 32662, 89250, 41641, 55873, 19169, 46629,
       51927, 69224, 24557, 77762, 96355, 90910, 21945, 79357, 42784,
       54638, 29944, 97318, 52022, 20391, 21016, 26679, 70682, 87326,
       15853, 75501, 97984, 53378, 93458, 56501, 46955, 33492, 35832,
       12470, 53981, 22773, 84691, 50087, 27877, 56647, 59419, 26783,
       79453, 33962, 84429, 63057, 43128, 40220, 79700, 67083, 22054,
       18192, 25222, 58635, 39114, 48738, 28496, 62209, 87677, 93659,
       56503, 27081, 48779, 91304, 49520, 41117, 18199, 76992, 98588,
       20719, 13068, 84835, 44795, 39052, 38704, 98416, 86958, 29222,
       71357, 29543, 23855, 84836, 93307, 64614, 72949, 79211, 28707,
       83229, 73620, 70819, 48313, 37481, 20851, 86079, 10933, 66905,
       36890, 19060, 61906, 31088, 70365, 56536, 16992, 14300, 51914,
       12083, 71376, 99307, 35200, 70812, 62067, 62365, 80618, 52891,
       31409, 81790, 57794, 20604, 83459, 43051, 68911, 38374, 86784,
       42488, 35922, 46610, 28764, 75170, 24370, 84083, 11784, 10003,
       98970, 77735, 46070, 27143, 56974, 99109, 32603, 24215, 33092,
       71297, 68268, 33810, 49995, 11865, 28904, 39144, 31591, 30167,
       72382, 79869, 70526, 38143, 70298, 54747, 92997, 24347, 57445,
       70645, 69540, 70712, 32900, 14326, 43865, 29046, 95088, 38586,
       30468, 42254, 32467, 68404, 58451, 86208, 62426, 63710, 25062,
       56438, 17783, 92426, 64667, 32483, 15673, 77511, 76182, 89831,
       12072, 81231, 48169, 31699, 15710, 31645, 98933, 52912, 68756,
       41087, 46798, 34475, 98334, 10900, 61445, 78796, 62141, 90945,
       88918, 62873, 70949, 31521, 59615, 52111, 34598, 61124, 59689,
       23075, 48192, 77367, 82625, 51802, 78922, 38670, 48392, 47425,
       73906, 94013, 27698, 26793, 67033, 20109, 95185, 27684, 98240,
       18032, 46470, 33301, 62800, 27520, 26094, 13571, 36239, 77082,
       71572, 30495, 49771, 32905, 21337, 16082, 43883, 15491, 48081,
       36317, 15627, 51028, 14422, 45525, 23480, 19184, 80239, 33215,
       65203, 65190, 16320, 21696, 56129, 74971, 33356, 29607, 95319,
       94413, 78963, 66865, 40427, 93733, 63543, 41184, 96652, 56555,
       36457, 26101, 36156, 49958, 68696, 97453, 68533, 96438, 50348,
       85016, 73188, 64225, 64437, 66330, 26372, 30714, 84263, 86531,
```

```
93090, 79373, 17440, 81536, 62494, 37580, 60822, 96032, 91545,
73313, 54345, 33138, 33579, 83094, 90065, 51913, 81378, 32236,
97980, 38626, 64230, 38183, 89277, 68269, 58751, 88720, 99311,
94528, 72268, 79593, 94247, 76700, 12011, 87935, 98720, 38638,
37857, 15532, 22159, 91547, 98738, 21966, 14398, 18572, 77312,
42074, 82799, 12659, 19649, 50119, 18888, 83061, 39343, 92007,
78728, 48640, 92070, 97341, 97493, 78400, 52862, 11804, 10455,
25766, 58199, 88613, 16690, 71164, 48825, 29236, 48940, 64320,
84845, 52663, 17957, 69018, 58496, 99128, 35868, 79478, 57402,
73795, 58403, 85972, 62998, 32786, 17962, 59440, 94772, 80083,
81026, 86883, 39444, 35886, 24541, 54502, 17183, 39562, 70157,
72151, 65170, 67936, 93339, 57365, 56628, 83814, 75324, 46939,
71428, 23884, 95145, 46002, 20309, 72115, 42769, 91870, 55308,
60789, 51363, 99239, 19272, 44411, 60915, 98433, 81867, 43980,
```

```
df
```

	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

```
df
```

	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

```
df['Label'] = df['Taxable.Income'].apply(lambda x: 'Risky' if x <= 30000 else 'Good')
```

```
df['Label'].unique()
```

```
array(['Good', 'Risky'], dtype=object)
```

```
df
```

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

text Converting categorical variables into numerical value

600 rows × 7 columns

```
from sklearn.preprocessing import LabelEncoder


ls = LabelEncoder()

df['Undergrad'] = ls.fit_transform(df['Undergrad'])

df['Urban'] = ls.fit_transform(df['Urban'])
df['Label'] = ls.fit_transform(df['Label'])



df['Marital.Status'] = ls.fit_transform(df['Marital.Status'])


df.head()
```

	Undergrad	Marital.Status	Taxable.Income	City.Population	Work.Experience	Urban	Label
0	0	2	68833	50047	10	1	0
1	1	0	33700	134075	18	1	0
2	0	1	36925	160205	30	1	0
3	1	2	50190	193264	15	1	0
4	0	1	81002	27533	28	0	0

```
df['Label'].value_counts()
```

```
0    476
1    124
Name: Label, dtype: int64
```

Split Data

```
x = df[['Taxable.Income', 'Marital.Status', 'Work.Experience', 'Urban', 'Undergrad']]
y = df['Label']


from sklearn.model_selection import train_test_split


x_test, x_train, y_test, y_train = train_test_split(x, y, test_size = 0.30, random_state=100)
```

Train the model

```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier()
clf.fit(x_train, y_train)
```

```
DecisionTreeClassifier()
```

Checking the accuracy of Testing model

```
from sklearn.metrics import accuracy_score
predict_test = clf.predict(x_test)

accuracy_test = accuracy_score(y_test, predict_test)
accuracy_test
```

```
0.9976190476190476
```

Checking the accuracy of Training Model

```
from sklearn.metrics import accuracy_score
predict_train = clf.predict(x_train)

accuracy_train = accuracy_score(y_train, predict_train)
accuracy_train
```

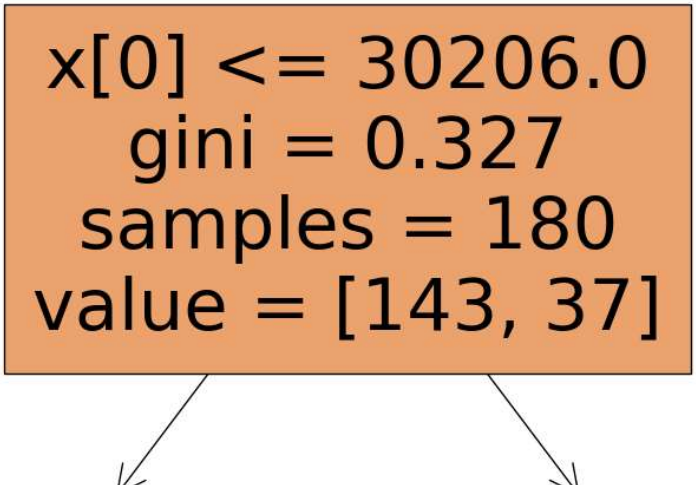
```
1.0
```

Visualizing final Decision tree

```
from sklearn import tree

plt.figure(figsize=(15,10))
tree.plot_tree(clf, filled = True)

plt.show()
```


x[0] <= 30206.0
gini = 0.327
samples = 180
value = [143, 37]