In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import  DecisionTreeClassifier
from sklearn import tree
from sklearn.metrics import classification_report
from sklearn import preprocessing
```

In [2]:
```
df = pd.read_csv("Fraud_check.csv")
df
```

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]:
```
df.head()
```

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

In [5]:
```
df.tail()
```

Out[5]:

	Undergrad	Marital.Status	Taxable.Income	City.Population	Work.Experience	Urban
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

In [6]:
```
#Creating dummy vairables for ['Undergrad','Marital.Status','Urban'] dropping fir
df=pd.get_dummies(df,columns=['Undergrad','Marital.Status','Urban'], drop_first=T
```

In [7]:
```
#Creating new cols TaxInc and dividing 'Taxable.Income' cols on the basis of [100
df["TaxInc"] = pd.cut(df["Taxable.Income"], bins = [10002,30000,99620], labels =
```

In [8]:
```
print(df)
```

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

     Marital.Status_Married  Marital.Status_Single  Urban_YES TaxInc
0                         0                      1          1   Good
1                         0                      0          1   Good
2                         1                      0          1   Good
3                         0                      1          1   Good
4                         1                      0          0   Good
..                      ...                    ...        ...    ...
595                       0                      0          1   Good
596                       0                      0          1   Good
597                       0                      0          1   Good
598                       1                      0          0   Good
599                       0                      0          0   Good

[600 rows x 8 columns]
```

Lets assume: taxable_income <= 30000 as “Risky=0” and others are “Good=1”

In [9]:
```
#After creation of new col. TaxInc also made its dummies var concating right side
df = pd.get_dummies(df,columns = ["TaxInc"],drop_first=True)
```

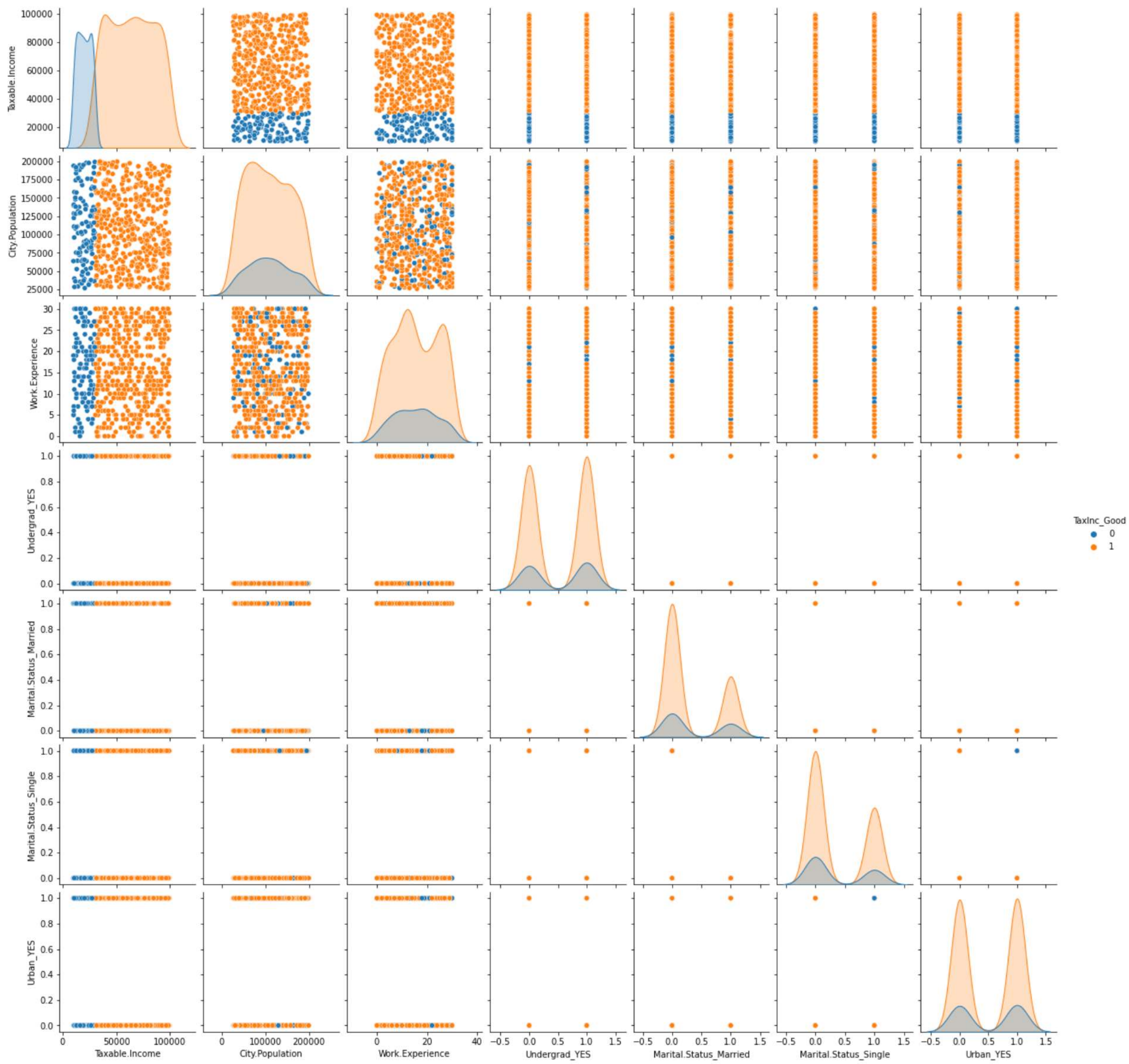
In [10]:
```
#Viewing buttom 10 observations
df.tail(10)
```

Out[10]:

	Taxable.Income	City.Population	Work.Experience	Undergrad_YES	Marital.Status_Married	Mari
590	43018	85195	14	0	1	
591	27394	132859	18	1	0	
592	68152	75143	16	1	0	
593	84775	131963	10	0	0	
594	47364	97526	9	0	1	
595	76340	39492	7	1	0	
596	69967	55369	2	1	0	
597	47334	154058	0	0	0	
598	98592	180083	17	1	1	
599	96519	158137	16	0	0	

In [11]:
```python
# let's plot pair plot to visualise the attributes all at once
import seaborn as sns
sns.pairplot(data=df, hue = 'TaxInc_Good')
```

Out[11]: <seaborn.axisgrid.PairGrid at 0x1a649429d00>



In [12]:
```python
# Normalization function
def norm_func(i):
    x = (i-i.min())/(i.max()-i.min())
    return (x)
```

In [13]:
```python
# Normalized data frame (considering the numerical part of data)
df_norm = norm_func(df.iloc[:,1:])
df_norm.tail(10)
```

Out[13]:

	City.Population	Work.Experience	Undergrad_YES	Marital.Status_Married	Marital.Status_Single
590	0.341473	0.466667	0.0	1.0	0.0
591	0.615406	0.600000	1.0	0.0	1.0
592	0.283703	0.533333	1.0	0.0	1.0
593	0.610256	0.333333	0.0	0.0	0.0
594	0.412341	0.300000	0.0	1.0	0.0
595	0.078811	0.233333	1.0	0.0	0.0
596	0.170058	0.066667	1.0	0.0	0.0
597	0.737240	0.000000	0.0	0.0	0.0
598	0.886810	0.566667	1.0	1.0	0.0
599	0.760683	0.533333	0.0	0.0	0.0

In [14]:
```python
# Declaring features & target
X = df_norm.drop(['TaxInc_Good'], axis=1)
y = df_norm['TaxInc_Good']
```

In [15]:
```python
from sklearn.model_selection import train_test_split
```

In [16]:
```python
# Splitting data into train & test
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.2, random_state
```

In [17]:
```
##Converting the Taxable income variable to bucketing.
df_norm["income"]="<=30000"
df_norm.loc[df["Taxable.Income"]>=30000,"income"]="Good"
df_norm.loc[df["Taxable.Income"]<=30000,"income"]="Risky"
```

In [18]:
```
##Droping the Taxable income variable
df.drop(["Taxable.Income"],axis=1,inplace=True)
```

In [19]:
```
df.rename(columns={"Undergrad":"undergrad","Marital.Status":"marital","City.Popul
## As we are getting error as "ValueError: could not convert string to float: 'YE
## Model.fit doesnt not consider String. So, we encode
```

In [20]:
```
from sklearn import preprocessing
le=preprocessing.LabelEncoder()
for column_name in df.columns:
    if df[column_name].dtype == object:
        df[column_name] = le.fit_transform(df[column_name])
    else:
        pass
```

In [21]:
```
##Splitting the data into featuers and labels
features = df.iloc[:,0:5]
labels = df.iloc[:,5]
```

In [22]:
```
## Collecting the column names
colnames = list(df.columns)
predictors = colnames[0:5]
target = colnames[5]
##Splitting the data into train and test
```

In [23]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(features,labels,test_size = 0.2,
```

In [24]:
```
##Model building
from sklearn.ensemble import RandomForestClassifier as RF
model = RF(n_jobs = 3,n_estimators = 15, oob_score = True, criterion = "entropy")
model.fit(x_train,y_train)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\_forest.py:541: Use
rWarning: Some inputs do not have OOB scores. This probably means too few trees
were used to compute any reliable oob estimates.
  warn("Some inputs do not have OOB scores. "
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\_forest.py:545: Run
timeWarning: invalid value encountered in true_divide
  decision = (predictions[k] /
```

Out[24]:
```
RandomForestClassifier(criterion='entropy', n_estimators=15, n_jobs=3,
                       oob_score=True)
```

In [25]:
```
model.estimators_
model.classes_
model.n_features_
model.n_classes_
```
Out[25]: 2

In [28]:
```
model.n_outputs_
```
Out[28]: 1

In [29]:
```
model.oob_score_
###74.7833%
```
Out[29]: 0.50625

In [30]:
```
##Predictions on train data
prediction = model.predict(x_train)
```
In [31]:
```
##Accuracy
# For accuracy
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_train,prediction)
##98.33%
```
In [32]:
```
np.mean(prediction == y_train)
##98.33%
```
Out[32]: 0.99375

In [33]:
```
##Confusion matrix
from sklearn.metrics import confusion_matrix
confusion = confusion_matrix(y_train,prediction)
```
In [34]:
```
##Prediction on test data
pred_test = model.predict(x_test)
```
In [35]:
```
##Accuracy
acc_test =accuracy_score(y_test,pred_test)
##78.333%
```
In [40]:
```
## In random forest we can plot a Decision tree present in Random forest
from sklearn.tree import export_graphviz
from six import StringIO
```
In [41]:
```
tree = model.estimators_[5]
```
In [44]:
```
dot_data = StringIO()
export_graphviz(tree,out_file = dot_data, filled = True,rounded = True, feature_r
```

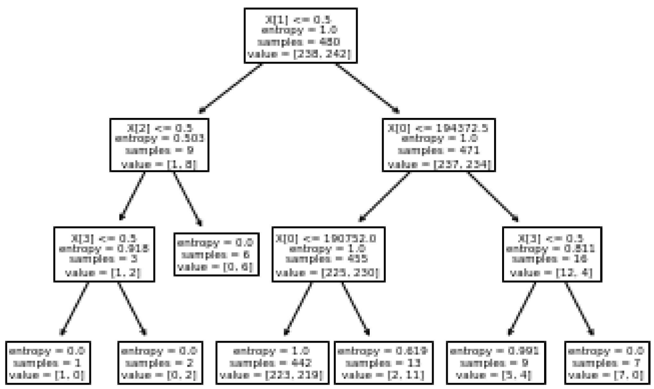
Building Decision Tree Classifier using Entropy Criteria

In [47]:
```python
model = DecisionTreeClassifier(criterion = 'entropy',max_depth=3)
model.fit(x_train,y_train)
```

Out[47]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=3)
```

In [48]:
```python
from sklearn import tree
```

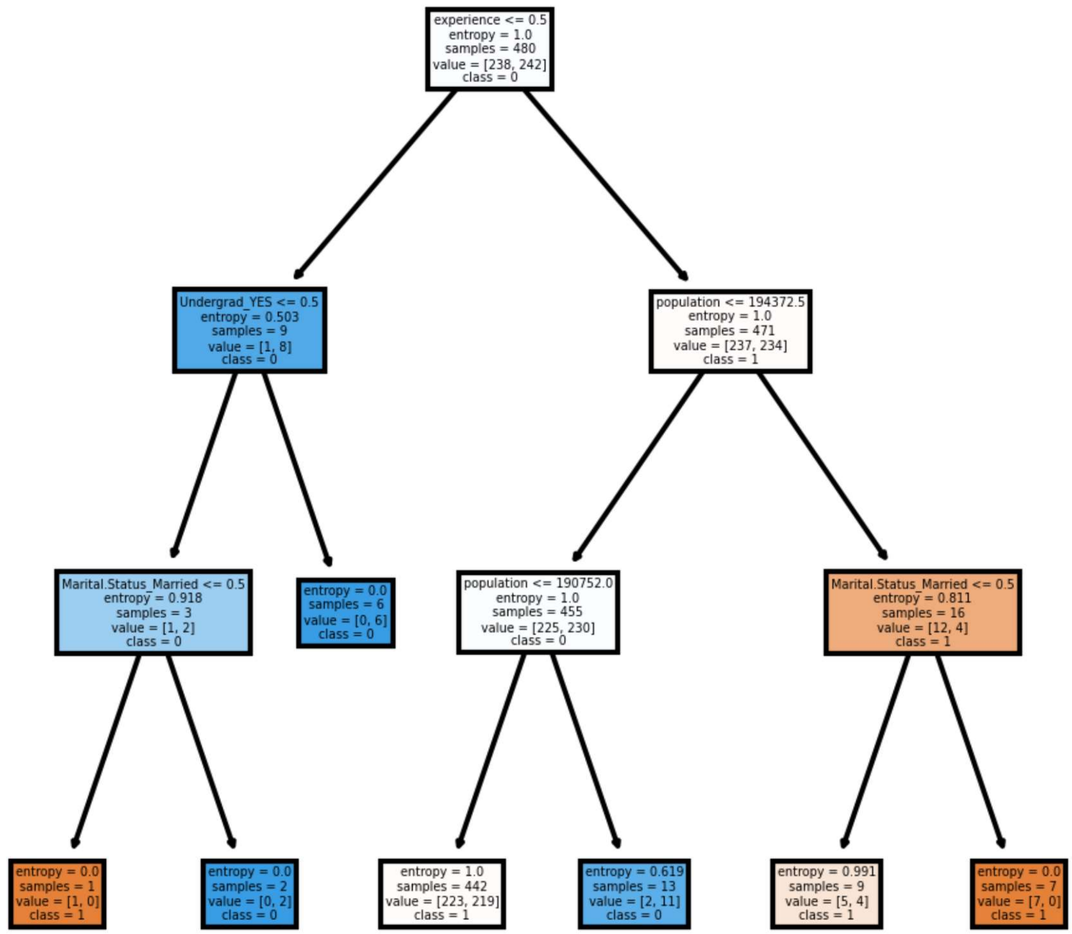
In [49]:
```python
#PLot the decision tree
tree.plot_tree(model);
```



In [50]:
```python
colnames = list(df.columns)
colnames
```

Out[50]:
```
['population',
 'experience',
 'Undergrad_YES',
 'Marital.Status_Married',
 'Marital.Status_Single',
 'Urban_YES',
 'TaxInc_Good']
```

```
In [51]: fn=['population','experience','Undergrad_YES','Marital.Status_Married','Marital.S
cn=['1', '0']
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=300)
tree.plot_tree(model,
               feature_names = fn,
               class_names=cn,
               filled = True);
```



In [52]:
```
#Predicting on test data
preds = model.predict(x_test) # predicting on test data set
pd.Series(preds).value_counts() # getting the count of each category
```

Out[52]:
```
0    114
1      6
dtype: int64
```

In [53]:
```
preds
```

Out[53]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 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, 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, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=uint8)
```

In [54]:
```
pd.crosstab(y_test,preds) # getting the 2 way table to understand the correct and
```

Out[54]:

col_0	0	1
Urban_YES		
0	57	3
1	57	3

In [55]:
```
# Accuracy
np.mean(preds==y_test)
```

Out[55]:
```
0.5
```

Building Decision Tree Classifier (CART) using Gini Criteria

In [56]:
```
from sklearn.tree import DecisionTreeClassifier
model_gini = DecisionTreeClassifier(criterion='gini', max_depth=3)
```

In [57]:
```
model_gini.fit(x_train, y_train)
```

Out[57]:
```
DecisionTreeClassifier(max_depth=3)
```

In [58]:
```
#Prediction and computing the accuracy
pred=model.predict(x_test)
np.mean(preds==y_test)
```

Out[58]:
```
0.5
```

Decision Tree Regression Example

In [59]:
```python
# Decision Tree Regression
from sklearn.tree import DecisionTreeRegressor
```

In [60]:
```python
array = df.values
X = array[:,0:3]
y = array[:,3]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_
model = DecisionTreeRegressor()
model.fit(X_train, y_train)
```

Out[60]:
```
DecisionTreeRegressor()
```

In [61]:
```python
#Find the accuracy
model.score(X_test,y_test)
```

Out[61]:
```
-0.9393656716417913
```

In [ ]:
```python

```