**Assignment 14 Fraud Check - Decision Trees**

Use Decision Trees to prepare a model on fraud data treating those who have taxable\_income <= 30000 as "Risky" and others are "Good"

Data Description :

Undergrad : person is under graduated or not

Marital.Status : marital status of a person

Taxable.Income : Taxable income is the amount of how much tax an individual owes to the government

Work Experience : Work experience of an individual person

Urban : Whether that person belongs to urban area or not

Ans:

In [1]:

*#importing necessary libraries*

**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

df **=** pd**.**read\_csv("Fraud\_check.csv")

In [2]:

*#Viewing top 5 rows of dataframe*

df**.**head()

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 |

In [3]:

df**.**tail()

Out[3]:

|  | **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 [4]:

*#Creating dummy vairables for ['Undergrad','Marital.Status','Urban'] dropping first dummy variable*

df**=**pd**.**get\_dummies(df,columns**=**['Undergrad','Marital.Status','Urban'], drop\_first**=True**)

In [5]:

*#Creating new cols TaxInc and dividing 'Taxable.Income' cols on the basis of [10002,30000,99620] for Risky and Good*

df["TaxInc"] **=** pd**.**cut(df["Taxable.Income"], bins **=** [10002,30000,99620], labels **=** ["Risky", "Good"])

In [6]:

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 [7]:

*#After creation of new col. TaxInc also made its dummies var concating right side of df*

df **=** pd**.**get\_dummies(df,columns **=** ["TaxInc"],drop\_first**=True**)

In [8]:

*#Viewing buttom 10 observations*

df**.**tail(10)

Out[8]:

|  | **Taxable.Income** | **City.Population** | **Work.Experience** | **Undergrad\_YES** | **Marital.Status\_Married** | **Marital.Status\_Single** | **Urban\_YES** | **TaxInc\_Good** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **590** | 43018 | 85195 | 14 | 0 | 1 | 0 | 1 | 1 |
| **591** | 27394 | 132859 | 18 | 1 | 0 | 1 | 1 | 0 |
| **592** | 68152 | 75143 | 16 | 1 | 0 | 1 | 0 | 1 |
| **593** | 84775 | 131963 | 10 | 0 | 0 | 0 | 1 | 1 |
| **594** | 47364 | 97526 | 9 | 0 | 1 | 0 | 1 | 1 |
| **595** | 76340 | 39492 | 7 | 1 | 0 | 0 | 1 | 1 |
| **596** | 69967 | 55369 | 2 | 1 | 0 | 0 | 1 | 1 |
| **597** | 47334 | 154058 | 0 | 0 | 0 | 0 | 1 | 1 |
| **598** | 98592 | 180083 | 17 | 1 | 1 | 0 | 0 | 1 |
| **599** | 96519 | 158137 | 16 | 0 | 0 | 0 | 0 | 1 |

In [41]:

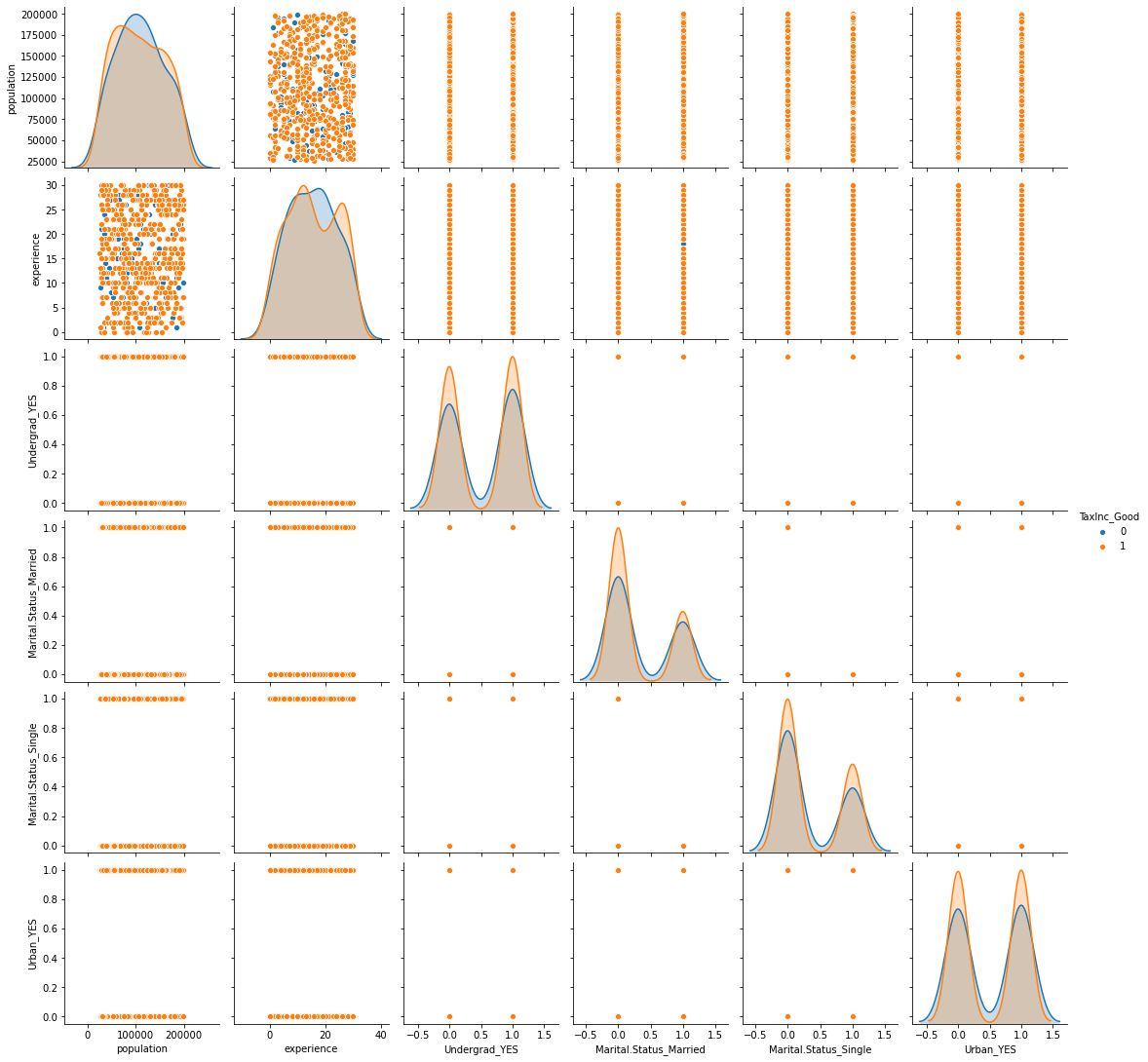
*# let's plot pair plot to visualise the attributes all at once*

**import** seaborn **as** sns

sns**.**pairplot(data**=**df, hue **=** 'TaxInc\_Good')

Out[41]:

<seaborn.axisgrid.PairGrid at 0x231e693f460>



In [9]:

*# Normalization function*

**def** norm\_func(i):

x **=** (i**-**i**.**min())**/**(i**.**max()**-**i**.**min())

**return** (x)

In [10]:

*# Normalized data frame (considering the numerical part of data)*

df\_norm **=** norm\_func(df**.**iloc[:,1:])

df\_norm**.**tail(10)

Out[10]:

| **City.Population** | **Work.Experience** | **Undergrad\_YES** | **Marital.Status\_Married** | **Marital.Status\_Single** | **Urban\_YES** | **TaxInc\_Good** |
| --- | --- | --- | --- | --- | --- | --- |
| **590** | 0.341473 | 0.466667 | 0.0 | 1.0 | 0.0 | 1.0 | 1.0 |
| **591** | 0.615406 | 0.600000 | 1.0 | 0.0 | 1.0 | 1.0 | 0.0 |
| **592** | 0.283703 | 0.533333 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 |
| **593** | 0.610256 | 0.333333 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 |
| **594** | 0.412341 | 0.300000 | 0.0 | 1.0 | 0.0 | 1.0 | 1.0 |
| **595** | 0.078811 | 0.233333 | 1.0 | 0.0 | 0.0 | 1.0 | 1.0 |
| **596** | 0.170058 | 0.066667 | 1.0 | 0.0 | 0.0 | 1.0 | 1.0 |
| **597** | 0.737240 | 0.000000 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 |
| **598** | 0.886810 | 0.566667 | 1.0 | 1.0 | 0.0 | 0.0 | 1.0 |
| **599** | 0.760683 | 0.533333 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 |

In [11]:

*# Declaring features & target*

X **=** df\_norm**.**drop(['TaxInc\_Good'], axis**=**1)

y **=** df\_norm['TaxInc\_Good']

In [12]:

**from** sklearn.model\_selection **import** train\_test\_split

In [13]:

*# Splitting data into train & test*

Xtrain, Xtest, ytrain, ytest **=** train\_test\_split(X, y, test\_size**=**0.2, random\_state**=**0)

In [14]:

*##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 [15]:

*##Droping the Taxable income variable*

df**.**drop(["Taxable.Income"],axis**=**1,inplace**=True**)

In [16]:

df**.**rename(columns**=**{"Undergrad":"undergrad","Marital.Status":"marital","City.Population":"population","Work.Experience":"experience","Urban":"urban"},inplace**=True**)

*## As we are getting error as "ValueError: could not convert string to float: 'YES'".*

*## Model.fit doesnt not consider String. So, we encode*

In [17]:

**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 [18]:

*##Splitting the data into featuers and labels*

features **=** df**.**iloc[:,0:5]

labels **=** df**.**iloc[:,5]

In [19]:

*## Collecting the column names*

colnames **=** list(df**.**columns)

predictors **=** colnames[0:5]

target **=** colnames[5]

*##Splitting the data into train and test*

In [20]:

**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,stratify **=** labels)

In [21]:

*##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)

Out[21]:

RandomForestClassifier(criterion='entropy', n\_estimators=15, n\_jobs=3,

oob\_score=True)

In [22]:

model**.**estimators\_

model**.**classes\_

model**.**n\_features\_

model**.**n\_classes\_

Out[22]:

2

In [23]:

model**.**n\_outputs\_

Out[23]:

1

In [24]:

model**.**oob\_score\_

*###74.7833%*

Out[24]:

0.5375

In [25]:

*##Predictions on train data*

prediction **=** model**.**predict(x\_train)

In [26]:

*##Accuracy*

*# For accuracy*

**from** sklearn.metrics **import** accuracy\_score

accuracy **=** accuracy\_score(y\_train,prediction)

*##98.33%*

In [27]:

np**.**mean(prediction **==** y\_train)

*##98.33%*

Out[27]:

0.9895833333333334

In [28]:

*##Confusion matrix*

**from** sklearn.metrics **import** confusion\_matrix

confusion **=** confusion\_matrix(y\_train,prediction)

In [29]:

*##Prediction on test data*

pred\_test **=** model**.**predict(x\_test)

In [30]:

*##Accuracy*

acc\_test **=**accuracy\_score(y\_test,pred\_test)

*##78.333%*

In [31]:

*## In random forest we can plot a Decision tree present in Random forest*

**from** sklearn.tree **import** export\_graphviz

**import** pydotplus

**from** six **import** StringIO

In [32]:

tree **=** model**.**estimators\_[5]

In [33]:

dot\_data **=** StringIO()

export\_graphviz(tree,out\_file **=** dot\_data, filled **=** **True**,rounded **=** **True**, feature\_names **=** predictors ,class\_names **=** target,impurity **=False**)

In [34]:

graph **=** pydotplus**.**graph\_from\_dot\_data(dot\_data**.**getvalue())

**Building Decision Tree Classifier using Entropy Criteria**

In [42]:

model **=** DecisionTreeClassifier(criterion **=** 'entropy',max\_depth**=**3)

model**.**fit(x\_train,y\_train)

Out[42]:

DecisionTreeClassifier(criterion='entropy', max\_depth=3)

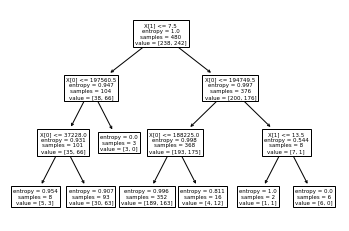
In [43]:

**from** sklearn **import** tree

In [44]:

*#PLot the decision tree*

tree**.**plot\_tree(model);



In [45]:

colnames **=** list(df**.**columns)

colnames

Out[45]:

['population',

'experience',

'Undergrad\_YES',

'Marital.Status\_Married',

'Marital.Status\_Single',

'Urban\_YES',

'TaxInc\_Good']

In [46]:

fn**=**['population','experience','Undergrad\_YES','Marital.Status\_Married','Marital.Status\_Single','Urban\_YES']

cn**=**['1', '0']

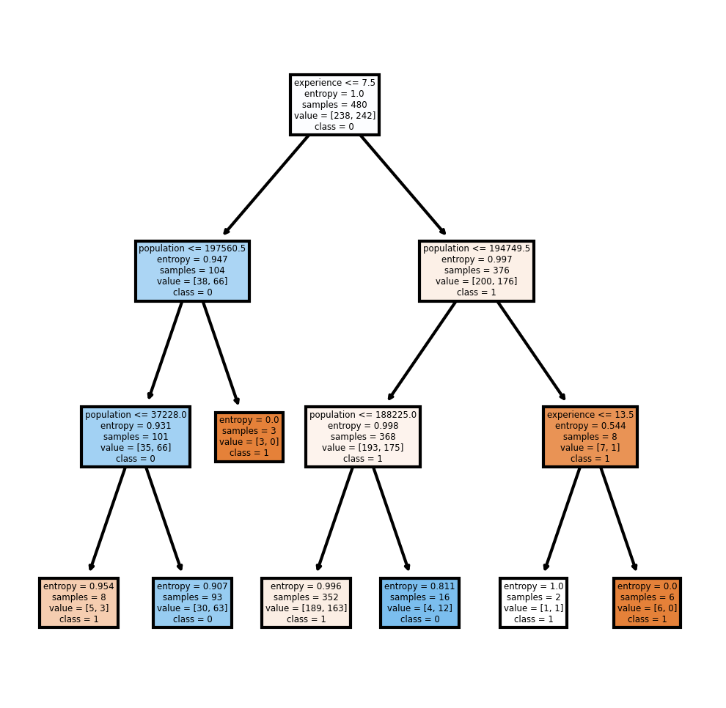
fig, axes **=** plt**.**subplots(nrows **=** 1,ncols **=** 1,figsize **=** (4,4), dpi**=**300)

tree**.**plot\_tree(model,

feature\_names **=** fn,

class\_names**=**cn,

filled **=** **True**);



In [47]:

*#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[47]:

0 85

1 35

dtype: int64

preds

Out[48]:

array([0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,

0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0,

0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,

0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,

0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,

0, 1, 0, 1, 0, 0, 0, 1, 0, 0], dtype=uint8)

In [49]:

pd**.**crosstab(y\_test,preds) *# getting the 2 way table to understand the correct and wrong predictions*

Out[49]:

| **col\_0** | **0** | **1** |
| --- | --- | --- |
| **Urban\_YES** |  |  |
| **0** | 42 | 18 |
| **1** | 43 | 17 |

In [50]:

*# Accuracy*

np**.**mean(preds**==**y\_test)

Out[50]:

0.49166666666666664

**Building Decision Tree Classifier (CART) using Gini Criteria**

In [51]:

**from** sklearn.tree **import** DecisionTreeClassifier

model\_gini **=** DecisionTreeClassifier(criterion**=**'gini', max\_depth**=**3)

In [52]:

model\_gini**.**fit(x\_train, y\_train)

Out[52]:

DecisionTreeClassifier(max\_depth=3)

In [53]:

*#Prediction and computing the accuracy*

pred**=**model**.**predict(x\_test)

np**.**mean(preds**==**y\_test)

Out[53]:

0.49166666666666664

**Decision Tree Regression Example**

In [54]:

*# Decision Tree Regression*

**from** sklearn.tree **import** DecisionTreeRegressor

In [55]:

array **=** df**.**values

X **=** array[:,0:3]

y **=** array[:,3]

In [56]:

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.33, random\_state**=**1)

In [57]:

model **=** DecisionTreeRegressor()

model**.**fit(X\_train, y\_train)

Out[57]:

DecisionTreeRegressor()

In [58]:

*#Find the accuracy*

model**.**score(X\_test,y\_test)

Out[58]:

-0.8931902985074629