Import neccessery libraries

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import metrics
from sklearn import externals
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')
```

problem

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

Import data

In [3]:

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

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

Data understanding

In [4]:
```
fraud_data.shape
```

Out[4]:
```
(600, 6)
```

In [5]:
```
fraud_data.dtypes
```

Out[5]:
```
Undergrad           object
Marital.Status      object
Taxable.Income       int64
City.Population      int64
Work.Experience      int64
Urban               object
dtype: object
```

In [6]:
```
fraud_data.isna().sum()
```

Out[6]:
```
Undergrad          0
Marital.Status     0
Taxable.Income     0
City.Population    0
Work.Experience    0
Urban              0
dtype: int64
```

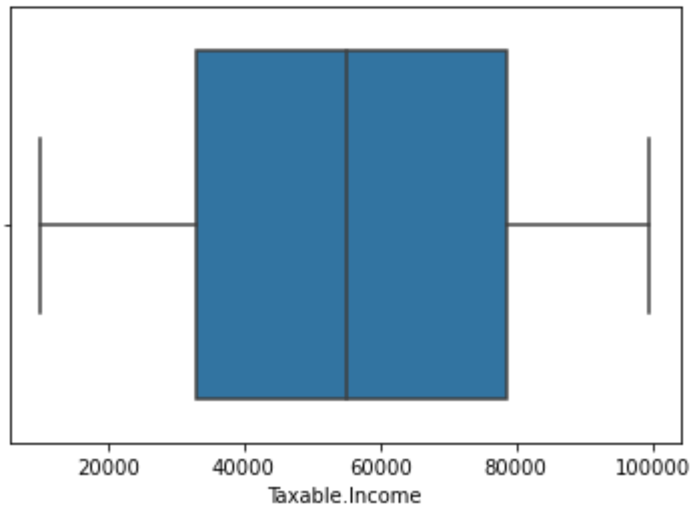
In [7]:
```
fraud_data.describe().T
```

Out[7]:

	count	mean	std	min	25%	50%	75%	
Taxable.Income	600.0	55208.375000	26204.827597	10003.0	32871.50	55074.5	78611.75	9
City.Population	600.0	108747.368333	49850.075134	25779.0	66966.75	106493.5	150114.25	19
Work.Experience	600.0	15.558333	8.842147	0.0	8.00	15.0	24.00	

Outlier check

In [8]:
```
ax = sns.boxplot(fraud_data['Taxable.Income'])
```

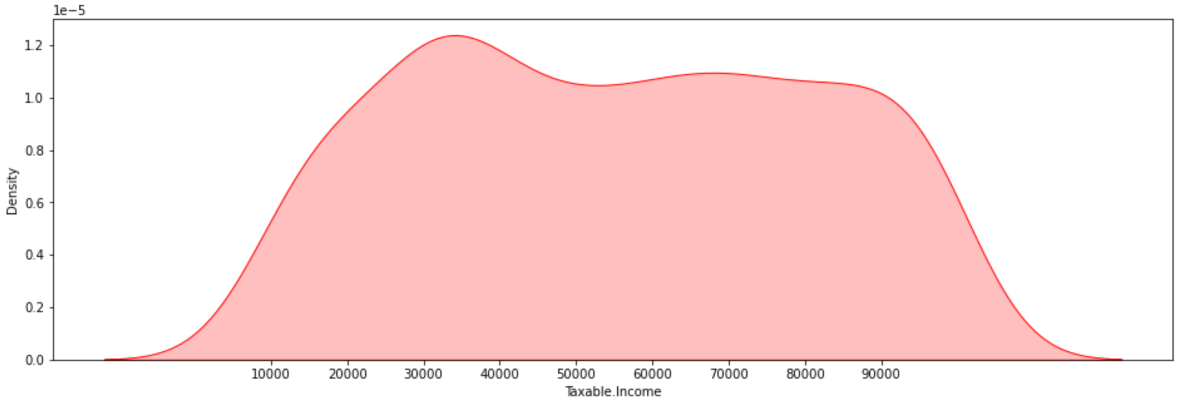


There are no outliers in the data

In [10]:
```
plt.rcParams["figure.figsize"] = 10,8
```

In [11]:
```
plt.figure(figsize=(16,5))
print("Skew: {}".format(fraud_data['Taxable.Income'].skew()))
print("Kurtosis: {}".format(fraud_data['Taxable.Income'].kurtosis()))
ax = sns.kdeplot(fraud_data['Taxable.Income'],shade=True,color='r')
plt.xticks([i for i in range(10000,100000,10000)])
plt.show()
```

```
Skew: 0.030014788906377175
Kurtosis: -1.1997824607083138
```

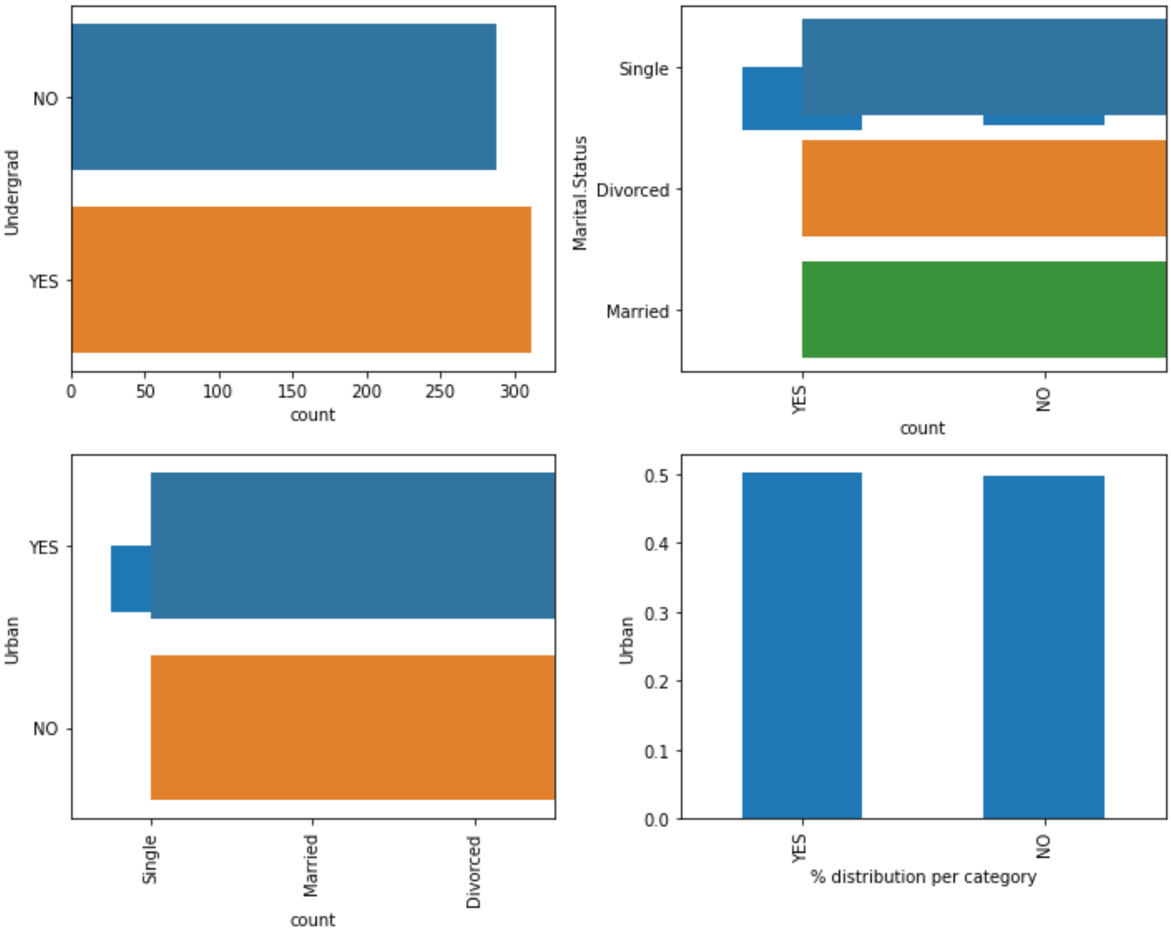


The data is Skwed on the right

The data has negative Kurtosis

In [13]:
```
obj_colum = fraud_data.select_dtypes(include='object').columns.tolist()
```

In [15]:
```
for i,col in enumerate(obj_colum,1):
    plt.subplot(2,2,i)
    sns.countplot(data=fraud_data,y=col)
    plt.subplot(2,2,i+1)
    fraud_data[col].value_counts(normalize=True).plot.bar()
    plt.ylabel(col)
    plt.xlabel('% distribution per category')
plt.tight_layout()
plt.show()
```

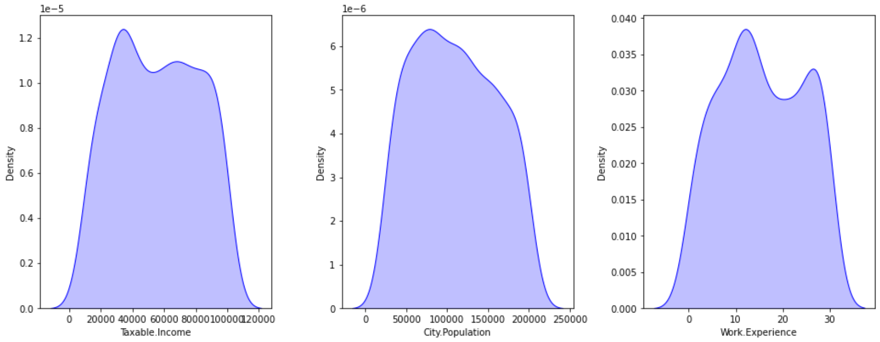


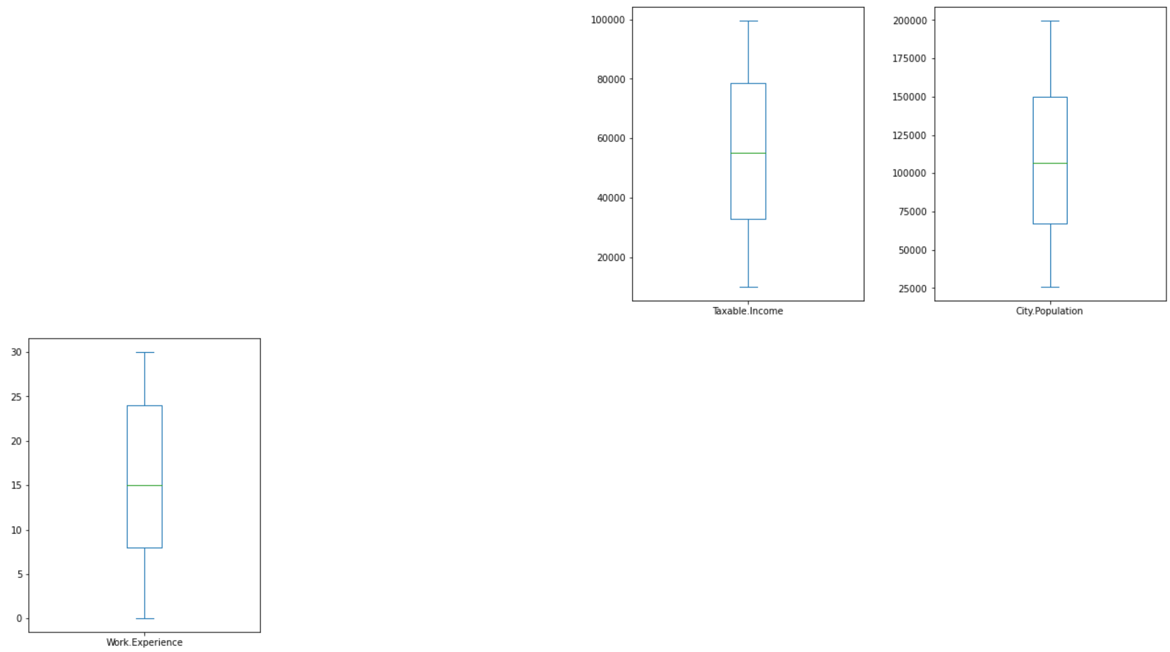
In [17]:

```
num_columns = fraud_data.select_dtypes(exclude='object').columns.tolist()
```

In [20]:

```
plt.figure(figsize=(18,40))
for i,col in enumerate(num_columns,1):
    plt.subplot(8,4,i)
    sns.kdeplot(fraud_data[col],color='b',shade=True)
    plt.subplot(8,4,i+10)
    fraud_data[col].plot.box()
plt.tight_layout()
plt.show()
num_data = fraud_data[num_columns]
pd.DataFrame(data=[num_data.skew(),num_data.kurtosis()],index=['skewness',
```





Out[20]:

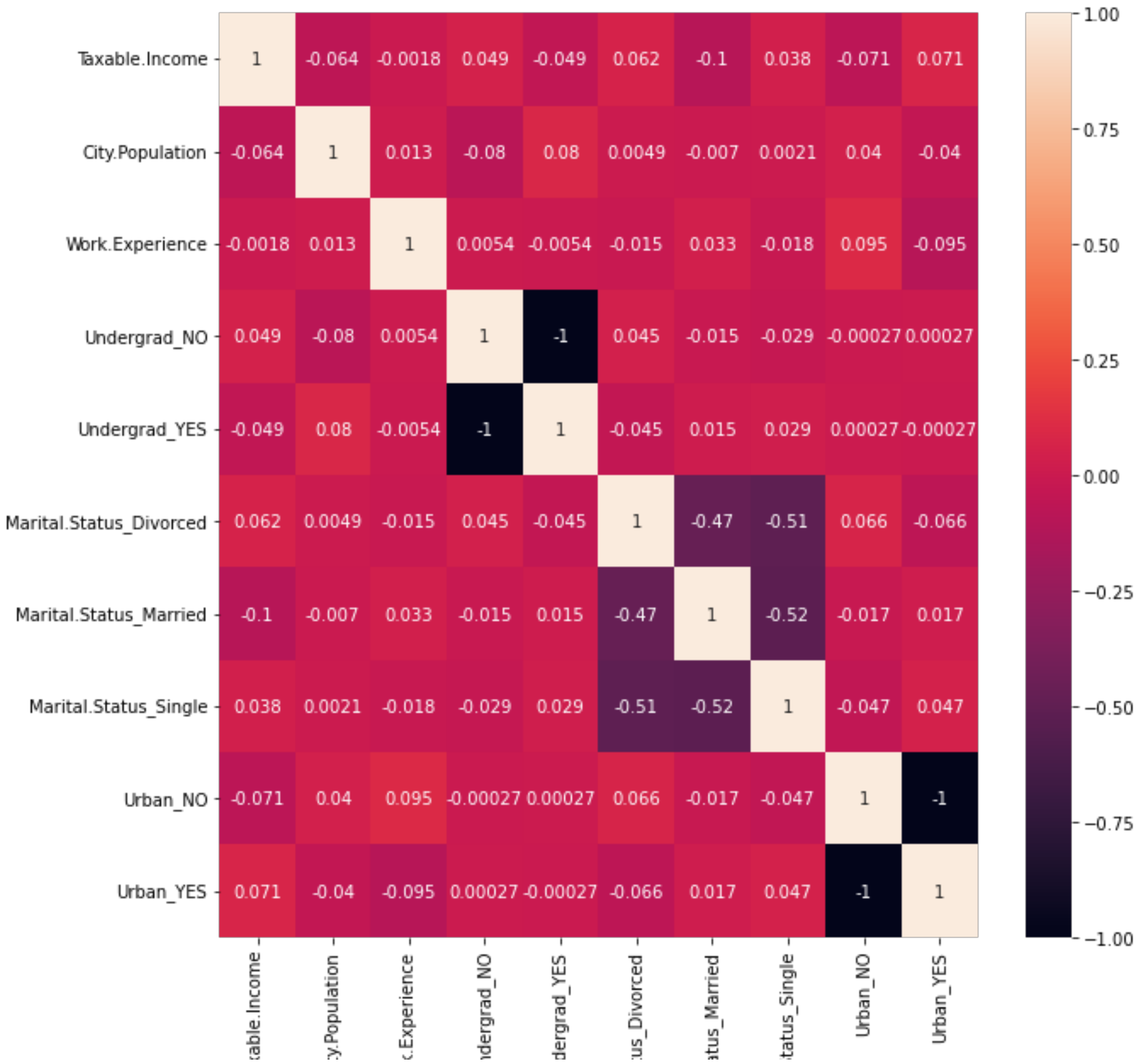
	Taxable.Income	City.Population	Work.Experience
skewness	0.030015	0.125009	0.018529
kurtosis	-1.199782	-1.120154	-1.167524

In [22]:
```
df1 = pd.get_dummies(fraud_data, columns = ['Undergrad','Marital.Status','U
```

In [23]:
```
corr = df1.corr()
```

In [24]:
```
plt.figure(figsize=(10,10))
sns.heatmap(corr,annot=True)
```

Out[24]: <AxesSubplot:>



3 - Decision Tree

Since the target variable is continious, we create a class of taxable_income <= 30000 as "Risky" and others are "Good"

In [25]:

```
df1['Taxable.Income']=pd.cut(df1['Taxable.Income'],bins=[0,30000,100000],la
```

In [26]:

```
list(df1.columns)
```

Out[26]:

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

In [27]:

```
X = df1.iloc[:,1:10]
y = df1.iloc[:,0]
```

In [28]:
```
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size = 0.2)
```

In [29]:
```
y_train.value_counts()
```

Out[29]:
```
good     377
risky    103
Name: Taxable.Income, dtype: int64
```

In [30]:
```
model = DecisionTreeClassifier()
model.fit(x_train,y_train)
```

Out[30]:
```
DecisionTreeClassifier()
```

In [31]:
```
pred_train = model.predict(x_train)
```

In [32]:
```
accuracy_score(y_train,pred_train)
```

Out[32]:
```
1.0
```

In [33]:
```
confusion_matrix(y_train,pred_train)
```

Out[33]:
```
array([[377,   0],
       [  0, 103]], dtype=int64)
```

In [34]:
```
pred_test = model.predict(x_test)
```

In [35]:
```
accuracy_score(y_test,pred_test)
```

Out[35]:
```
0.6416666666666667
```

In [36]:
```
confusion_matrix(y_test,pred_test)
```

Out[36]:
```
array([[75, 24],
       [19,  2]], dtype=int64)
```

In [37]:
```
df_t=pd.DataFrame({'Actual':y_test, 'Predicted':pred_test})
```

In [38]:
```
df_t
```

Out[38]:

	Actual	Predicted
9	good	good
569	good	risky
395	good	good
295	good	risky
481	good	good

	Actual	Predicted
...	...	...
203	good	good
372	good	good
42	good	good
501	good	good
226	good	good

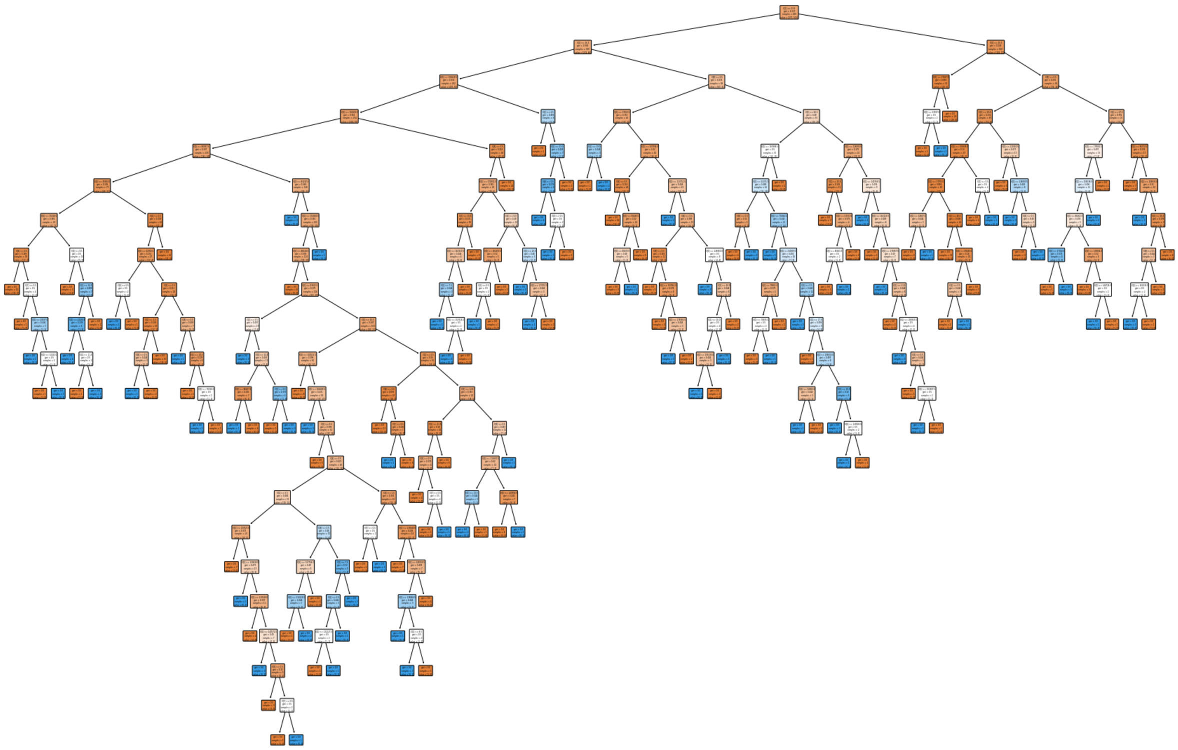
4 - Conclusion

Since the accuracy of the Training set is 100% we test the accurancy on the test data which is 69%

As seen in the confusion matrix of Test data 82 instances are presdected correctly and 38 instances are not

In [44]:

```
from sklearn.tree import plot_tree
plt.figure(figsize=(30,20))
plot_tree(decision_tree=model,filled=True,rounded=True)
plt.show()
```



In [40]:

```
model.feature_importances_
```

Out[40]:

```
array([0.56495418, 0.2248052 , 0.01798588, 0.02712609, 0.05388372,
       0.02299711, 0.01689372, 0.0394383 , 0.0319158 ])
```

In [42]:

```
fig = pd.DataFrame({'feature': list(x_train.columns),
                    'importance': model.feature_importances_}).\
                     sort_values('importance', ascending = False)
```

In [43]:

```
fig
```

Out[43]:

	feature	importance
0	City.Population	0.564954
1	Work.Experience	0.224805
4	Marital.Status_Divorced	0.053884
7	Urban_NO	0.039438
8	Urban_YES	0.031916
3	Undergrad_YES	0.027126
5	Marital.Status_Married	0.022997
2	Undergrad_NO	0.017986
6	Marital.Status_Single	0.016894

In [ ]: