Import neccessery librarires

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
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
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings('ignore')
```

Problem

Use Random Forest to prepare a model on fraud data

Import data

In [2]:

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

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

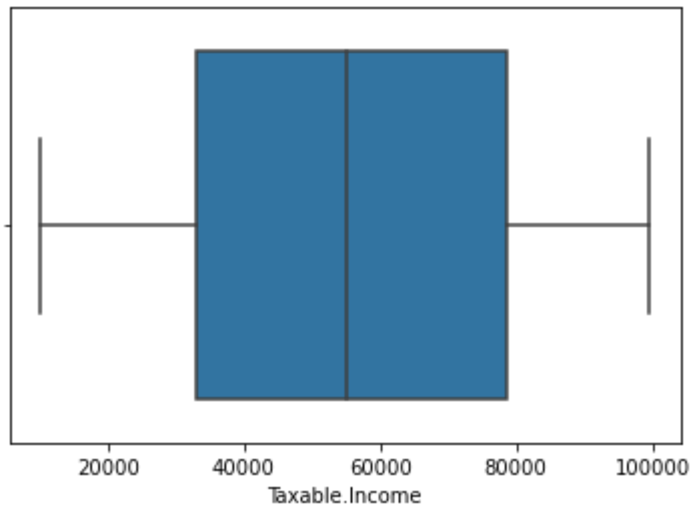
600 rows × 6 columns

Data understanding

In [3]:
```
fraud_data.shape
```
Out[3]:
```
(600, 6)
```
In [4]:
```
fraud_data.isnull().sum()
```
Out[4]:
```
Undergrad          0
Marital.Status     0
Taxable.Income     0
City.Population    0
Work.Experience    0
Urban              0
dtype: int64
```
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.describe().T
```
Out[6]:

	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

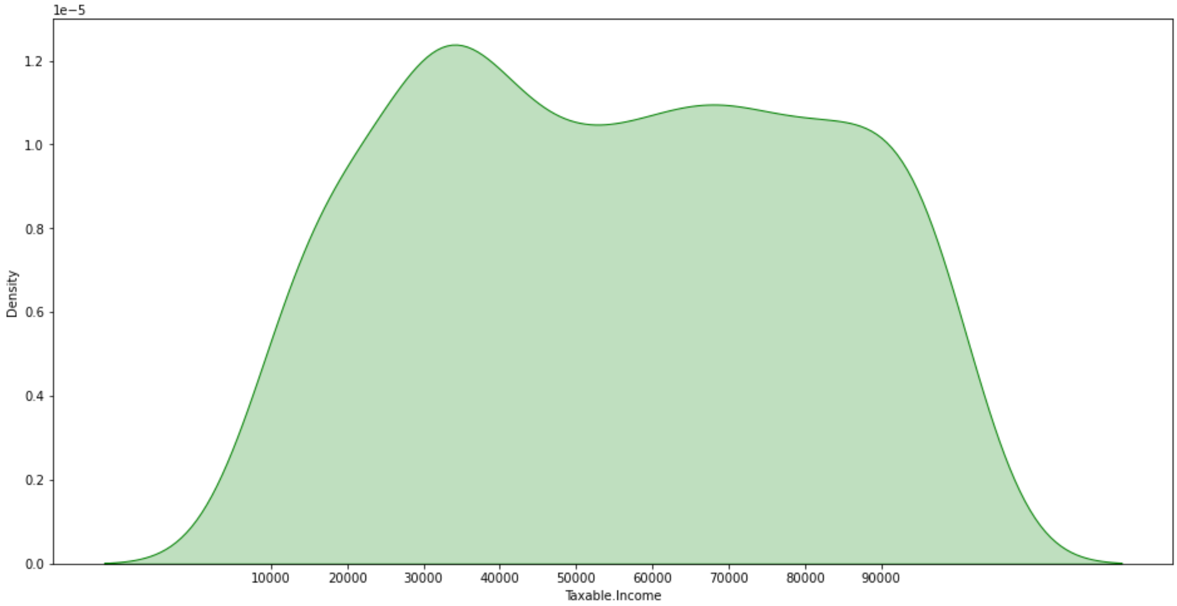
In [8]:
```
ax = sns.boxplot(fraud_data['Taxable.Income'])
```


There are no outliers in the data

In [9]:
```python
plt.rcParams["figure.figsize"] = 9,5
```

In [10]:
```python
plt.figure(figsize=(16,8))
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='g')
plt.xticks([i for i in range(10000,100000,10000)])
plt.show()
```

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

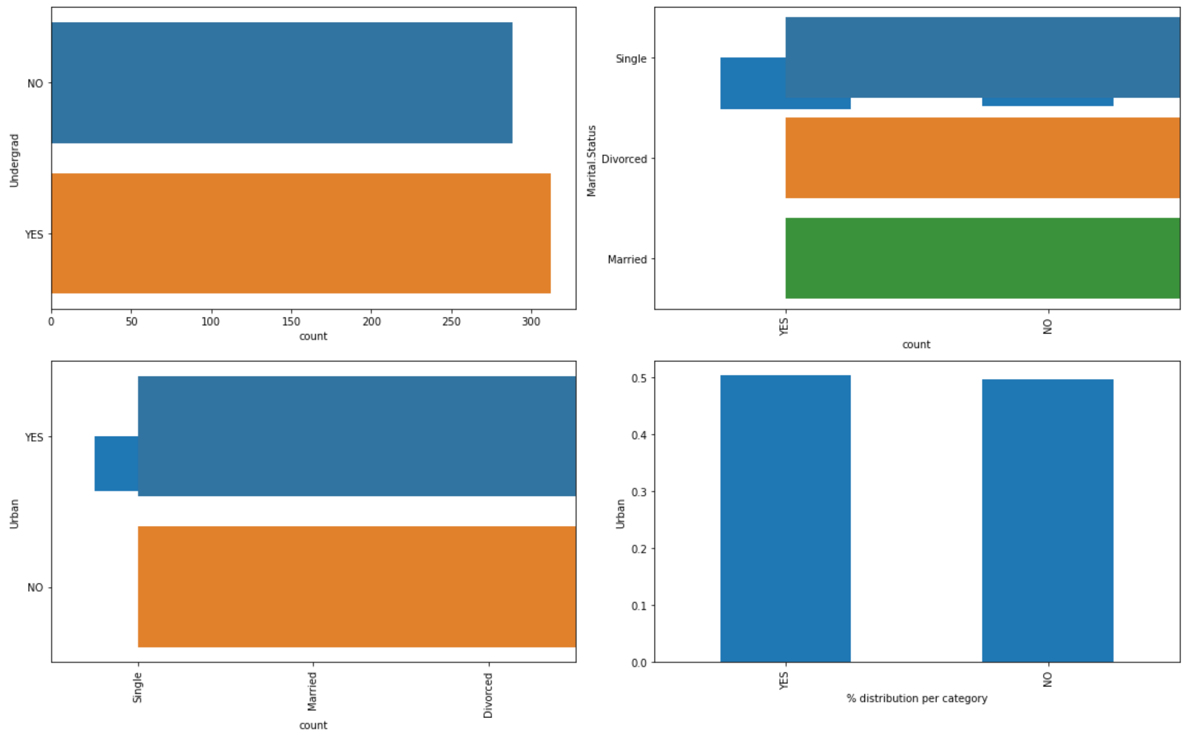


The data is Skwed on the right

The data has negative Kurtosis

In [11]:
```python
obj_colum = fraud_data.select_dtypes(include='object').columns.tolist()
```

In [13]:
```python
plt.figure(figsize=(16,10))
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()
```

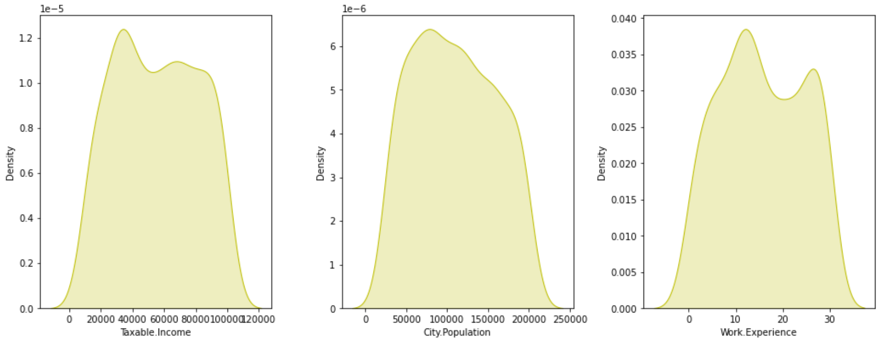


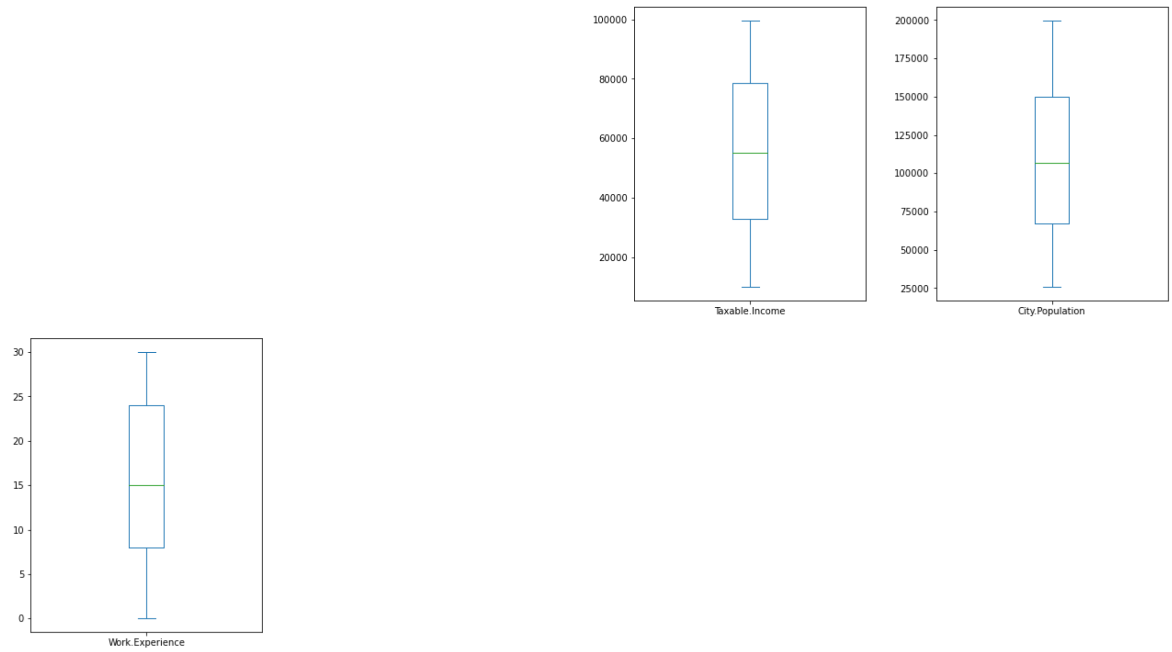
In [15]:

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

In [16]:

```
plt.figure(figsize=(18,40))
for i,col in enumerate(num_columns,1):
    plt.subplot(8,4,i)
    sns.kdeplot(fraud_data[col],color='y',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[16]:

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

In [18]:

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

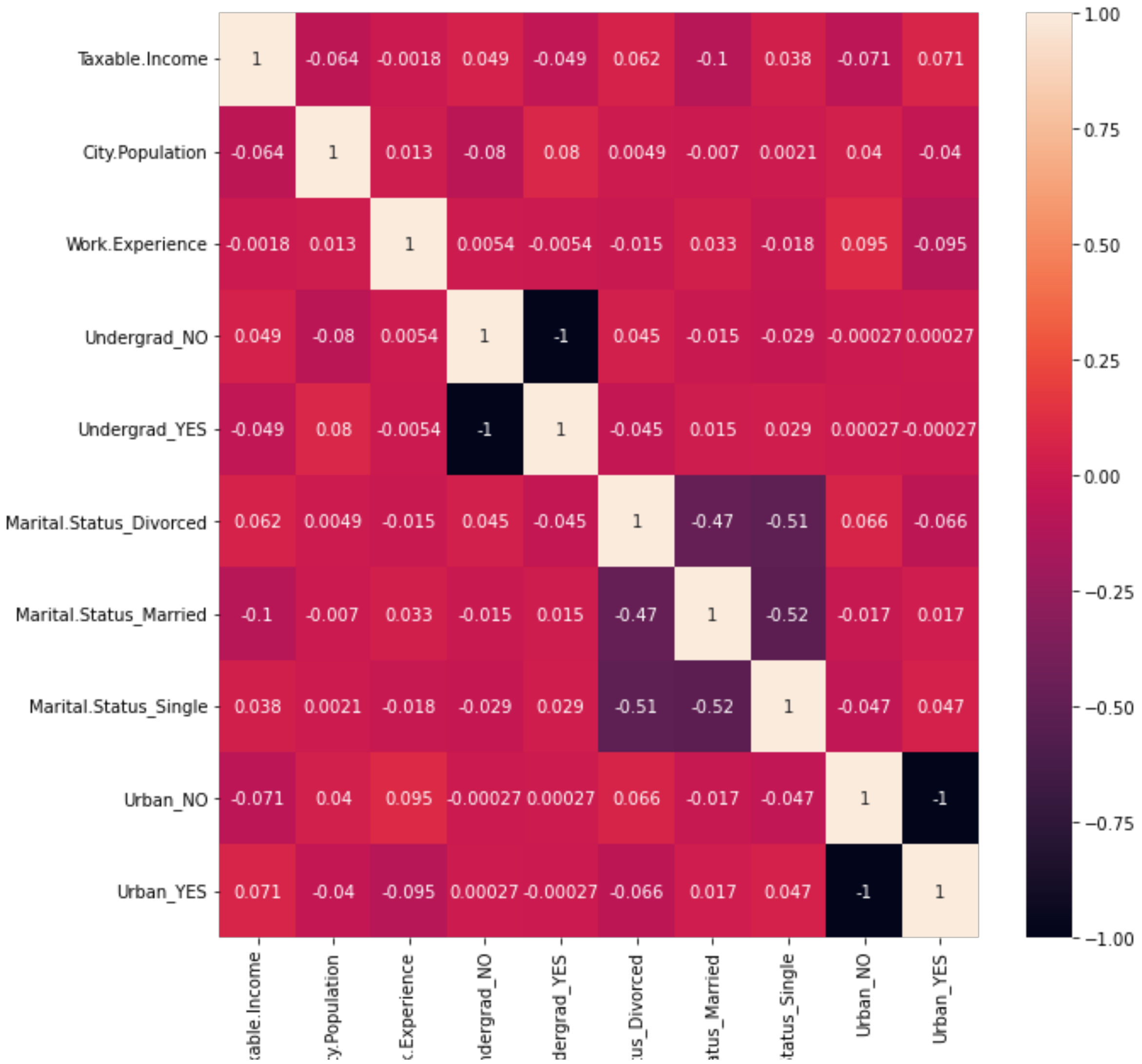
In [19]:

```
corr = df1.corr()
```

In [20]:

```
plt.figure(figsize=(10,10))
sns.heatmap(corr,annot=True)
```

Out[20]: <AxesSubplot:>



Random Forest Model

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

In [21]:

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

In [22]:

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

Out[22]:

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

In [23]:

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

In [24]:
```
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size = 0.2)
```

In [25]:
```
y_train.value_counts()
```

Out[25]:
```
good     386
risky     94
Name: Taxable.Income, dtype: int64
```

In [27]:
```
model =RandomForestClassifier(n_jobs=4,n_estimators = 150, oob_score =True,
model.fit(x_train,y_train)
model.oob_score_
```

Out[27]:
```
0.7645833333333333
```

In [28]:
```
pred_train = model.predict(x_train)
```

In [29]:
```
accuracy_score(y_train,pred_train)
```

Out[29]:
```
1.0
```

In [30]:
```
confusion_matrix(y_train,pred_train)
```

Out[30]:
```
array([[386,   0],
       [  0,  94]], dtype=int64)
```

In [31]:
```
pred_test = model.predict(x_test)
```

In [32]:
```
accuracy_score(y_test,pred_test)
```

Out[32]:
```
0.7083333333333334
```

In [33]:
```
confusion_matrix(y_test,pred_test)
```

Out[33]:
```
array([[84,  6],
       [29,  1]], dtype=int64)
```

In [34]:
```
df_t=pd.DataFrame({'Actual':y_test, 'Predicted':pred_test})
```

In [35]:
```
df_t
```

Out[35]:

	Actual	Predicted
122	risky	good
147	good	good
199	risky	good
399	good	good
458	risky	good

	Actual	Predicted
...	...	...
326	risky	good
58	risky	good
375	good	good
316	good	good
116	good	good

Conclusion

Since the accuracy of the Training set is 100% we test the accurancy on the test data which is 71% As seen in the confusion matrix of Test data 94 instances are presdected correctly and 26 instances are not

Let’s do hyperparameter tuning for Random Forest using GridSearchCV and fit the data.

In [36]:
```
rf = RandomForestClassifier(random_state=42, n_jobs=-1)
```

In [37]:
```
params = {
    'max_depth': [2,3,5,10,20],
    'min_samples_leaf': [5,10,20,50,100,200],
    'n_estimators': [10,25,30,50,100,200]
}
```

In [38]:
```
# Instantiate the grid search model
grid_search = GridSearchCV(estimator=rf,
                           param_grid=params,
                           cv = 4,
                           n_jobs=-1, verbose=1, scoring="accuracy")
```

In [39]:
```
%%time
grid_search.fit(x_train, y_train)
```

```
Fitting 4 folds for each of 180 candidates, totalling 720 fits
Wall time: 14.2 s
```

Out[39]:
```
GridSearchCV(cv=4, estimator=RandomForestClassifier(n_jobs=-1, random_state
=42),
             n_jobs=-1,
             param_grid={'max_depth': [2, 3, 5, 10, 20],
                         'min_samples_leaf': [5, 10, 20, 50, 100, 200],
                         'n_estimators': [10, 25, 30, 50, 100, 200]},
             scoring='accuracy', verbose=1)
```

In [40]:
```
grid_search.best_score_
```

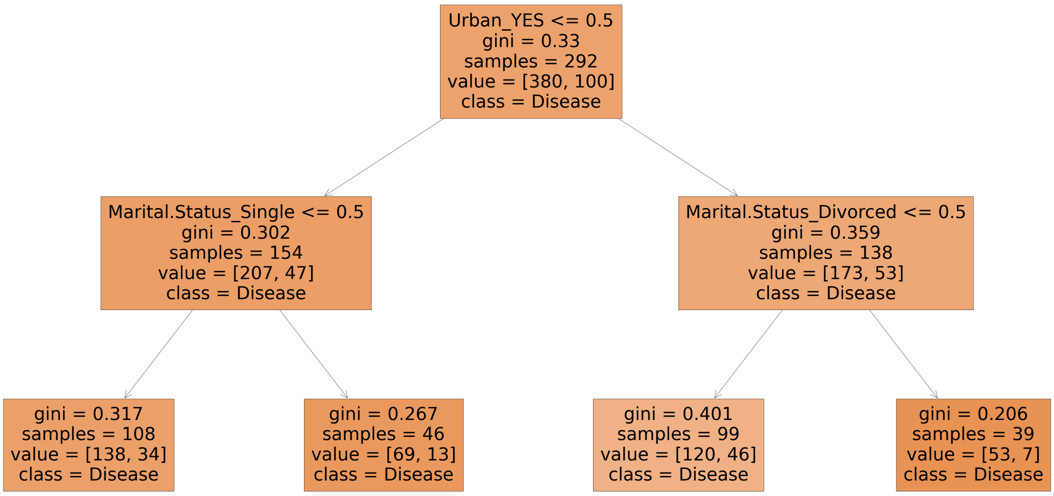
Out[40]:
```
0.8041666666666667
```

In [41]:
```python
rf_best = grid_search.best_estimator_
rf_best
```

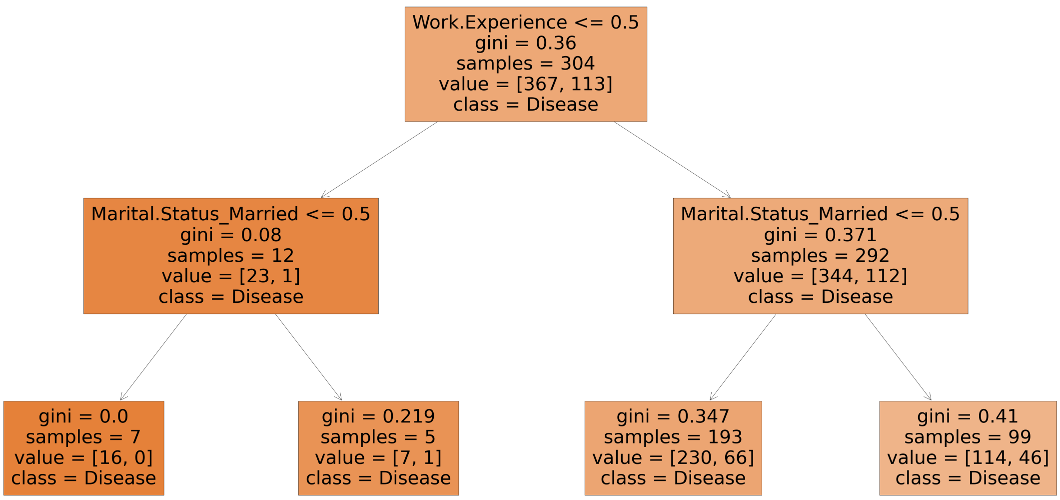
Out[41]:
```
RandomForestClassifier(max_depth=2, min_samples_leaf=5, n_estimators=10,
                       n_jobs=-1, random_state=42)
```

Now let's visualize

In [45]:
```python
from sklearn.tree import plot_tree
plt.figure(figsize=(80,40))
plot_tree(rf_best.estimators_[4], feature_names = X.columns,class_names=['I
```



In [44]:
```python
from sklearn.tree import plot_tree
plt.figure(figsize=(80,40))
plot_tree(rf_best.estimators_[8], feature_names = X.columns,class_names=['I
```



The trees created by estimators*[4] and estimators*[8] are different. Thus we can say that each tree is independent of the other.

Now let's sort the data with the help of feature importance

In [46]:
```
rf_best.feature_importances_
```

Out[46]:
```
array([0.28939711, 0.37473903, 0.00831241, 0.06132809, 0.11047169,
       0.06155576, 0.05380403, 0.01213875, 0.02825312])
```

In [47]:
```
imp_df = pd.DataFrame({
    "features": x_train.columns,
    "Importance": rf_best.feature_importances_
})
```

In [48]:
```
imp_df.sort_values(by="Importance", ascending=False)
```

Out[48]:

	features	Importance
1	Work.Experience	0.374739
0	City.Population	0.289397
4	Marital.Status_Divorced	0.110472
5	Marital.Status_Married	0.061556
3	Undergrad_YES	0.061328
6	Marital.Status_Single	0.053804
8	Urban_YES	0.028253
7	Urban_NO	0.012139
2	Undergrad_NO	0.008312

As seen in the above table city population is most important feature

In [ ]:
```

```