Import neccessery libraries

In [58]:

```
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

A cloth manufacturing company is interested to know about the segment or attributes causes high sale

Import data

In [2]:

```
company_data = pd.read_csv('Company_Data.csv')
company_data
```

Out[2]:

	Sales	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urba
0	9.50	138	73	11	276	120	Bad	42	17	Y
1	11.22	111	48	16	260	83	Good	65	10	Y
2	10.06	113	35	10	269	80	Medium	59	12	Y
3	7.40	117	100	4	466	97	Medium	55	14	Y
4	4.15	141	64	3	340	128	Bad	38	13	Y
...	...	...	...	...	...	...	...	...	...	
395	12.57	138	108	17	203	128	Good	33	14	Y
396	6.14	139	23	3	37	120	Medium	55	11	N
397	7.41	162	26	12	368	159	Medium	40	18	Y
398	5.94	100	79	7	284	95	Bad	50	12	Y
399	9.71	134	37	0	27	120	Good	49	16	Y

400 rows × 11 columns

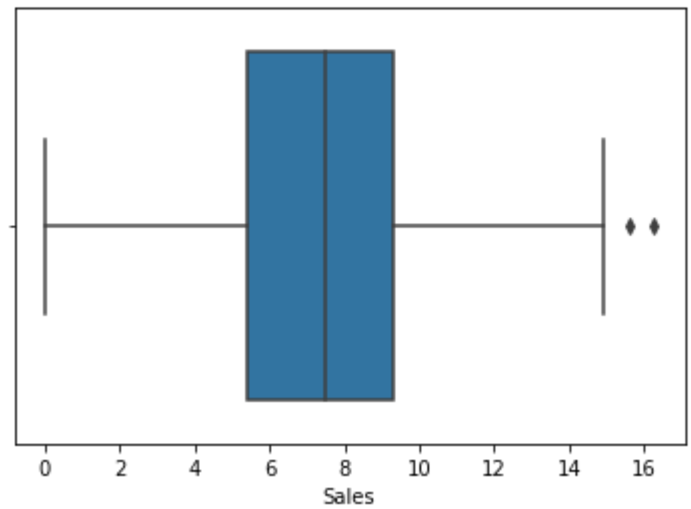
Data understanding

In [3]:
```
company_data.shape
```
Out[3]:
```
(400, 11)
```
In [4]:
```
company_data.isnull().sum()
```
Out[4]:
```
Sales          0
CompPrice      0
Income         0
Advertising    0
Population     0
Price          0
ShelveLoc      0
Age            0
Education      0
Urban          0
US             0
dtype: int64
```
In [5]:
```
company_data.dtypes
```
Out[5]:
```
Sales          float64
CompPrice        int64
Income           int64
Advertising      int64
Population       int64
Price            int64
ShelveLoc       object
Age              int64
Education        int64
Urban           object
US              object
dtype: object
```
In [6]:
```
company_data.describe().T
```
Out[6]:

	count	mean	std	min	25%	50%	75%	max
Sales	400.0	7.496325	2.824115	0.0	5.39	7.49	9.32	16.27
CompPrice	400.0	124.975000	15.334512	77.0	115.00	125.00	135.00	175.00
Income	400.0	68.657500	27.986037	21.0	42.75	69.00	91.00	120.00
Advertising	400.0	6.635000	6.650364	0.0	0.00	5.00	12.00	29.00
Population	400.0	264.840000	147.376436	10.0	139.00	272.00	398.50	509.00
Price	400.0	115.795000	23.676664	24.0	100.00	117.00	131.00	191.00
Age	400.0	53.322500	16.200297	25.0	39.75	54.50	66.00	80.00
Education	400.0	13.900000	2.620528	10.0	12.00	14.00	16.00	18.00

Outlier Check

In [8]:
```
ax = sns.boxplot(company_data['Sales'])
```



Data has 2 outlier instances

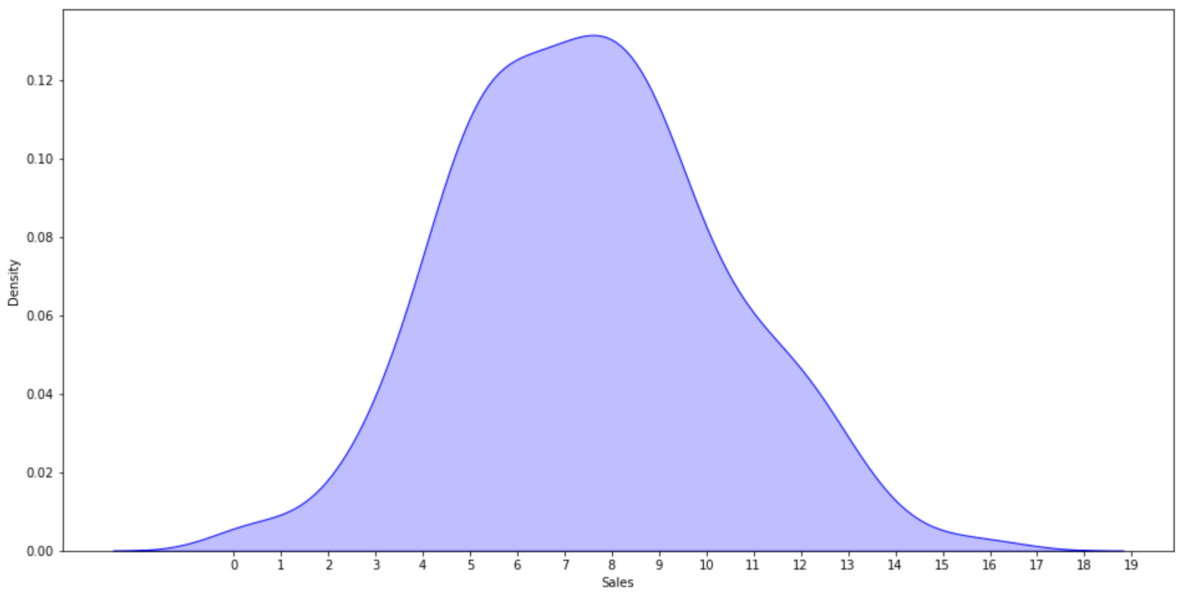
In [9]:

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

In [10]:

```
plt.figure(figsize=(16,8))
print("Skew: {}".format(company_data['Sales'].skew()))
print("Kurtosis: {}".format(company_data['Sales'].kurtosis()))
ax = sns.kdeplot(company_data['Sales'],shade=True,color='b')
plt.xticks([i for i in range(0,20,1)])
plt.show()
```

```
Skew: 0.18556036318721578
Kurtosis: -0.08087736743346197
```



The data is Skwed on the right

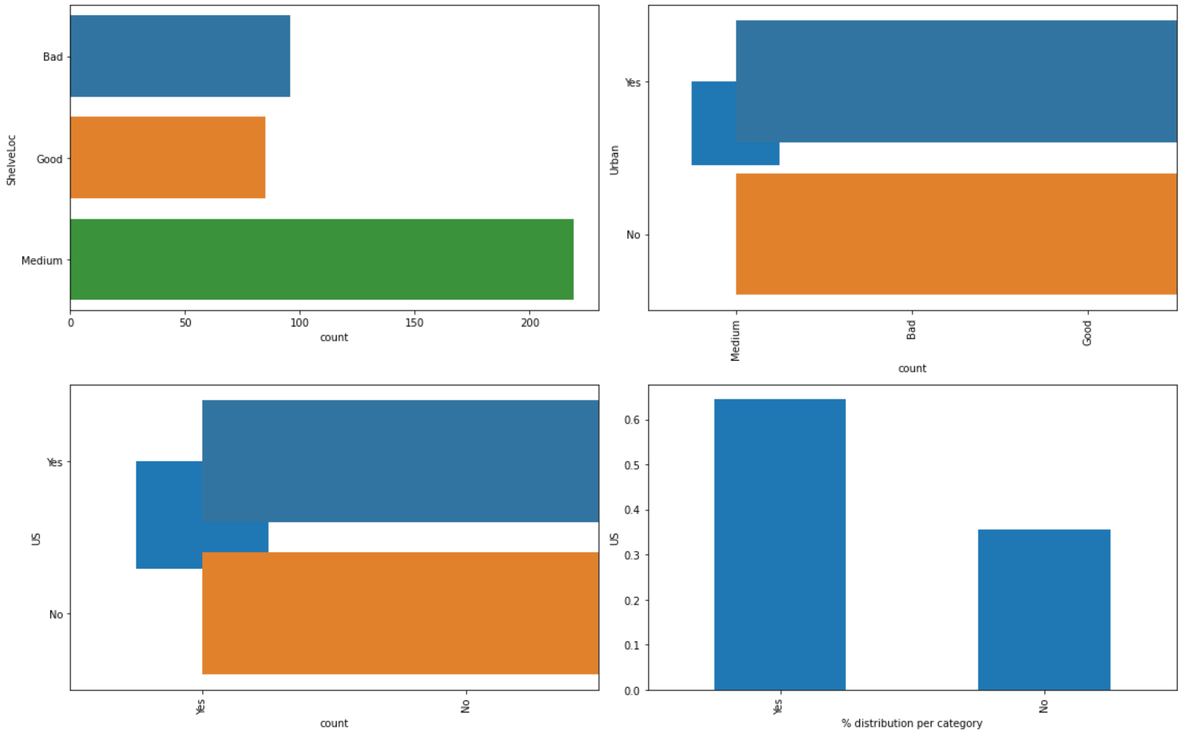
The data has negative Kurtosis

In [11]:

```
obj_colum = company_data.select_dtypes(include='object').columns.tolist()
```

In [13]:

```
plt.figure(figsize=(16,10))
for i,col in enumerate(obj_colum,1):
    plt.subplot(2,2,i)
    sns.countplot(data=company_data,y=col)
    plt.subplot(2,2,i+1)
    company_data[col].value_counts(normalize=True).plot.bar()
    plt.ylabel(col)
    plt.xlabel('% distribution per category')
plt.tight_layout()
plt.show()
```

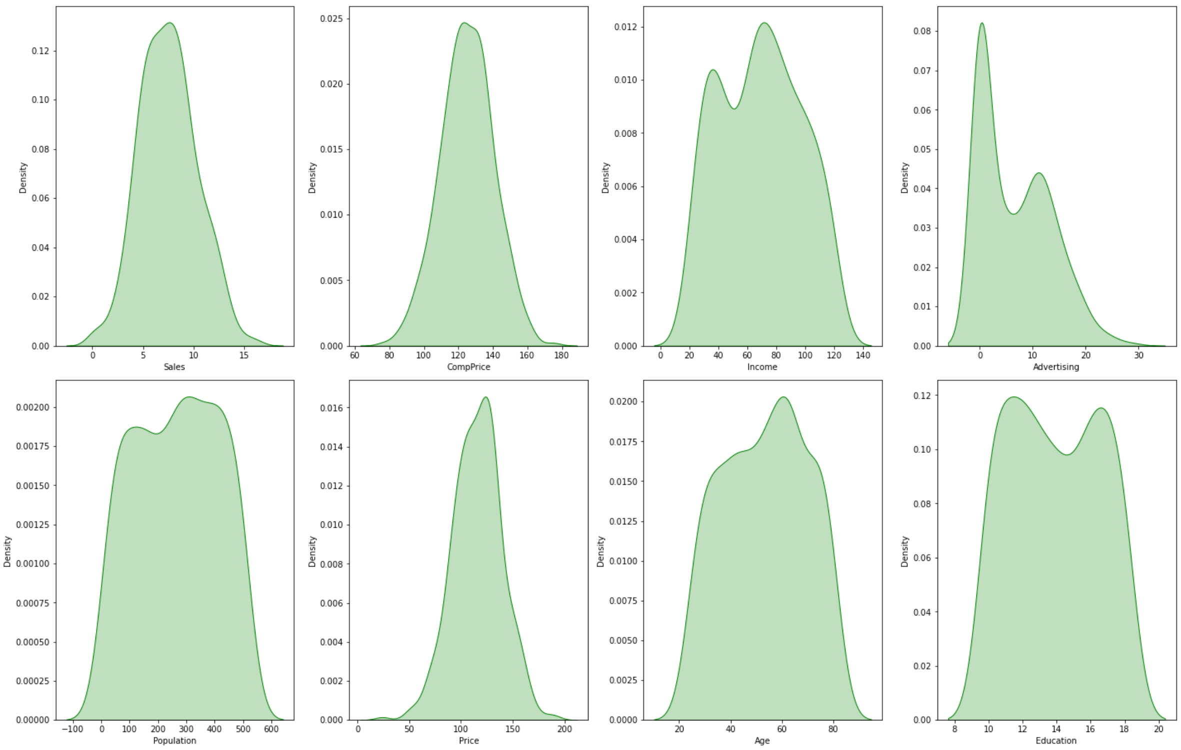


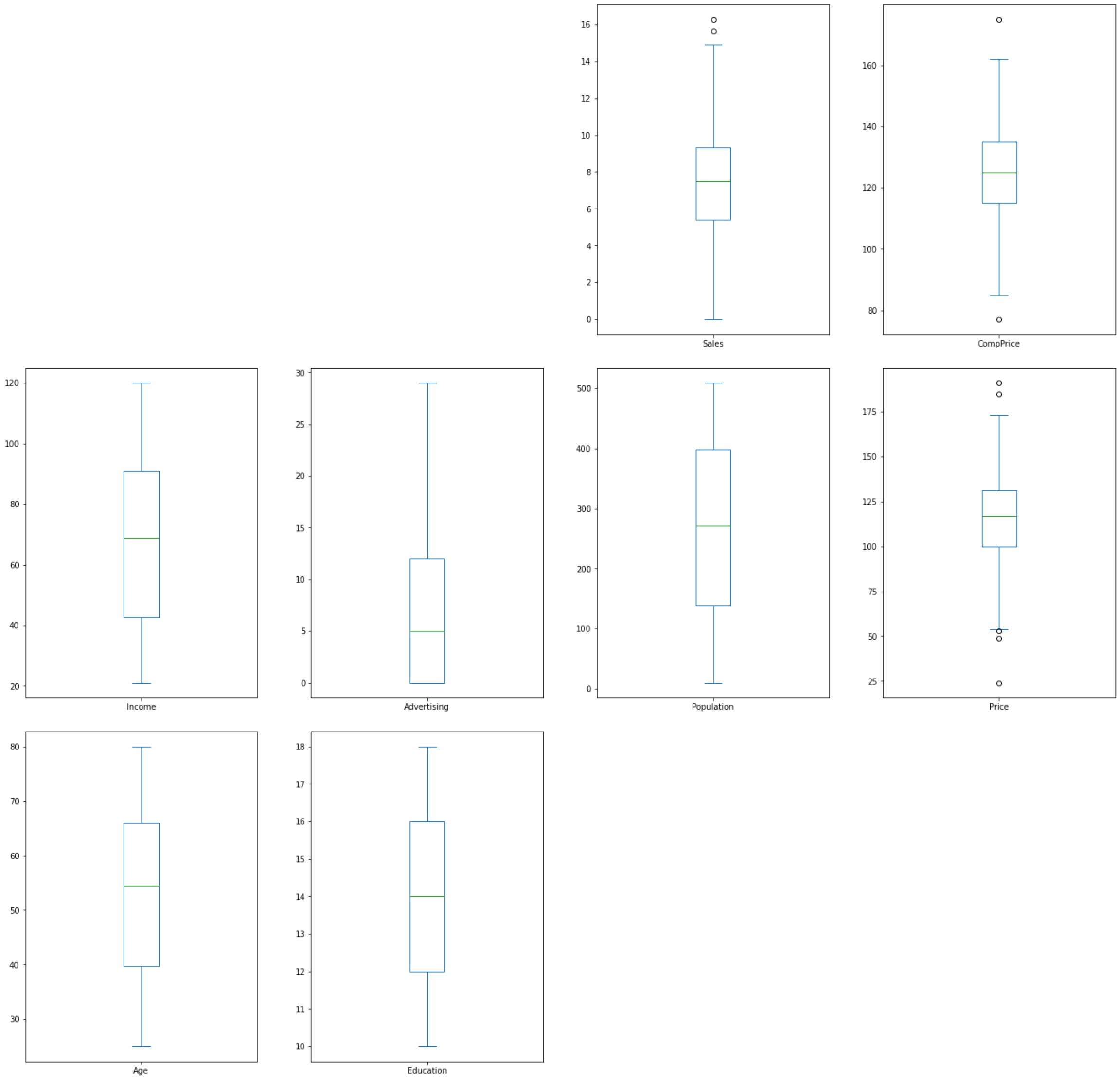
In [14]:

```
number_columns = company_data.select_dtypes(exclude='object').columns.tolis
```

In [17]:

```
plt.figure(figsize=(20,50))
for i,col in enumerate(number_columns,1):
    plt.subplot(8,4,i)
    sns.kdeplot(company_data[col],color='g',shade=True)
    plt.subplot(8,4,i+10)
    company_data[col].plot.box()
plt.tight_layout()
plt.show()
number_data = company_data[number_columns]
pd.DataFrame(data=[number_data.skew(),number_data.kurtosis()],index=['skewn
```





Out[17]:

	Sales	CompPrice	Income	Advertising	Population	Price	Age	Educat
skewness	0.185560	-0.042755	0.049444	0.639586	-0.051227	-0.125286	-0.077182	0.044(
kurtosis	-0.080877	0.041666	-1.085289	-0.545118	-1.202318	0.451885	-1.134392	-1.298:

In [18]:

```
corr = company_data.corr()
```

In [19]:

```
com_data2 = pd.get_dummies(company_data, columns = ['ShelveLoc','Urban','US
```

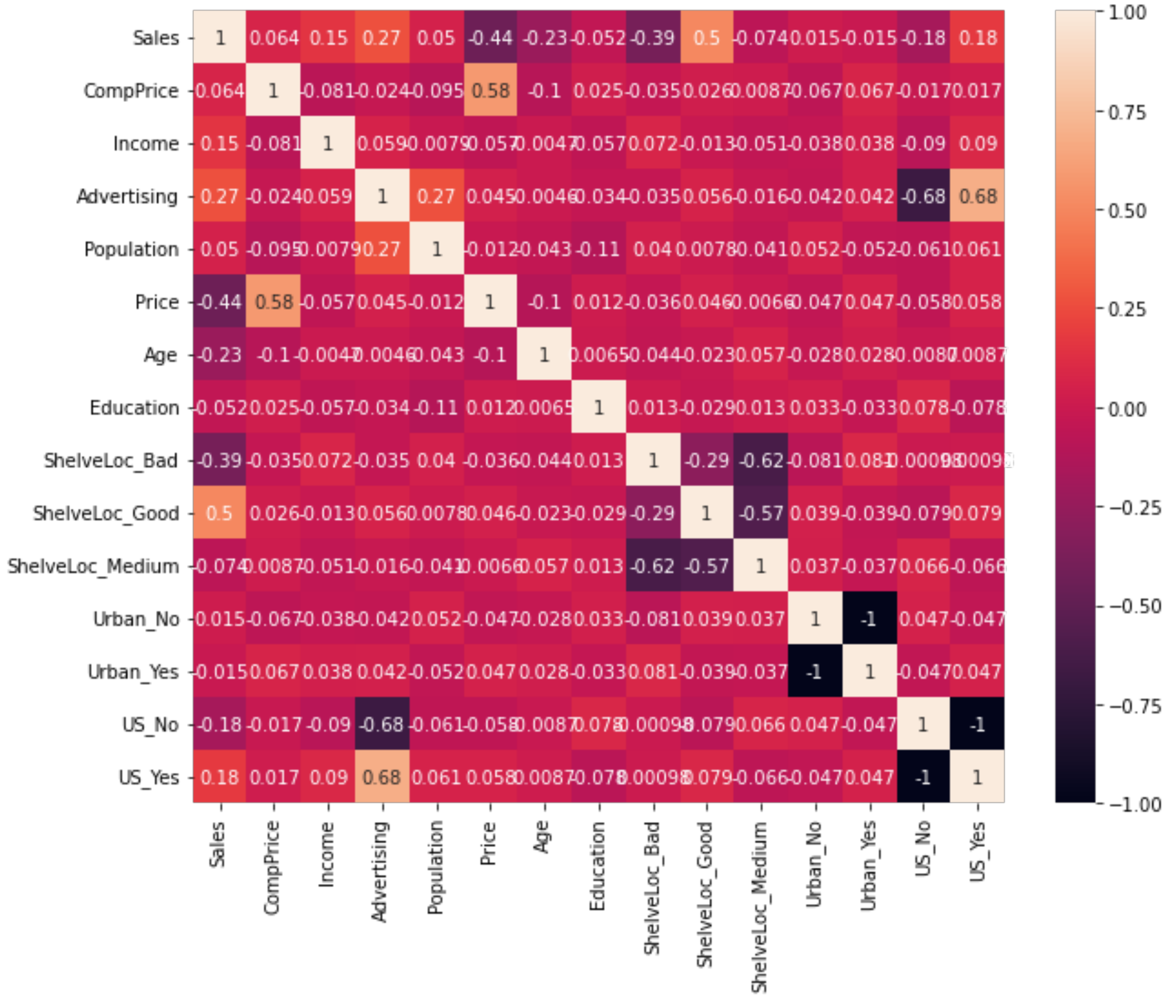
In [20]:

```
corr = com_data2.corr()
```

In [21]:

```
plt.figure(figsize=(10,8))
sns.heatmap(corr,annot=True)
```

Out[21]: <AxesSubplot:>



Random Forest Model

Since the target variable is continious, we create a class of the value based on the mean

<= 7.49 == "Small" and > 7.49 == "large"

In [22]:

```
com_data2["sales"]="small"
com_data2.loc[com_data2["Sales"]>7.49,"sales"]="large"
com_data2.drop(["Sales"],axis=1,inplace=True)
```

In [23]:
```
X = com_data2.iloc[:,0:14]
y = com_data2.iloc[:,14]
```

In [29]:
```
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size = 0.2)
```

In [30]:
```
y_train.value_counts()
```

Out[30]:
```
large    160
small    160
Name: sales, dtype: int64
```

In [46]:
```
model =RandomForestClassifier(n_jobs=4,n_estimators = 150, oob_score =True,
model.fit(x_train,y_train)
```

Out[46]:
```
RandomForestClassifier(criterion='entropy', n_estimators=150, n_jobs=4,
                       oob_score=True)
```

In [44]:
```
%%time
model.fit(x_train, y_train)
```

```
Wall time: 200 ms
```

Out[44]:
```
RandomForestClassifier(criterion='entropy', n_estimators=150, n_jobs=4,
                       oob_score=True)
```

In [47]:
```
# checking the oob score
model.oob_score_
```

Out[47]:
```
0.8
```

In [48]:
```
pred_train = model.predict(x_train)
```

In [73]:
```
accuracy_score(y_train,pred_train)
```

Out[73]:
```
1.0
```

In [74]:
```
confusion_matrix(y_train,pred_train)
```

Out[74]:
```
array([[160,   0],
       [  0, 160]], dtype=int64)
```

In [75]:
```
pred_test = model.predict(x_test)
```

In [76]:
```
accuracy_score(y_test,pred_test)
```

Out[76]:
```
0.8
```

In [77]:
```
confusion_matrix(y_test,pred_test)
```

Out[77]:
```
array([[31,  8],
       [ 8, 33]], dtype=int64)
```

In [54]:

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

In [55]:

```
df_t
```

Out[55]:

	Actual	Predicted
214	small	small
275	small	small
45	small	small
136	small	small
190	large	large
...	...	...
74	small	small
158	large	large
145	large	large
325	large	large
245	large	large

80 rows × 2 columns

Conclusion

**Since the accuracy of the Training set is 100% we test the accurancy on the test data which is 0.8%*

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

In [56]:

```
rf = RandomForestClassifier(random_state=42, n_jobs=-1)
```

In [57]:

```
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 [59]:

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

In [61]:

```
%%time
grid_search.fit(x_train, y_train)
```

```
Fitting 4 folds for each of 180 candidates, totalling 720 fits
Wall time: 16.7 s
```

Out[61]:

```
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 [62]:

```
grid_search.best_score_
```

Out[62]:

```
0.8093750000000001
```

In [63]:

```
rf_best = grid_search.best_estimator_
rf_best
```

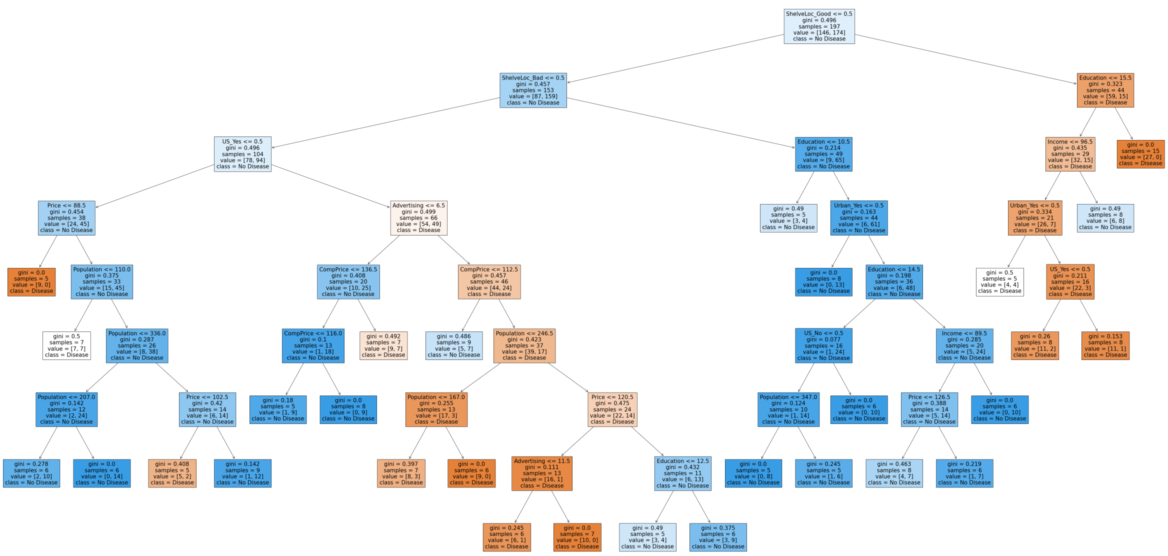
Out[63]:

```
RandomForestClassifier(max_depth=20, min_samples_leaf=5, n_estimators=200,
                       n_jobs=-1, random_state=42)
```

Now let's visualize

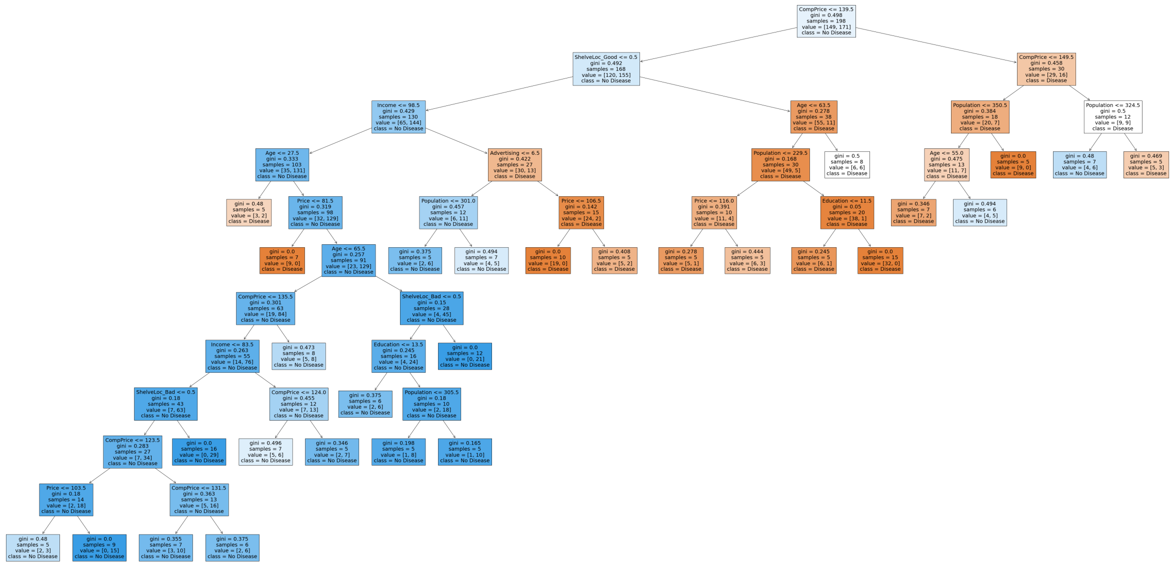
In [64]:

```
from sklearn.tree import plot_tree
plt.figure(figsize=(80,40))
plot_tree(rf_best.estimators_[5], feature_names = X.columns,class_names=['I
```



In [65]:

```
from sklearn.tree import plot_tree
plt.figure(figsize=(80,40))
plot_tree(rf_best.estimators_[7], feature_names = X.columns,class_names=['I
```



The trees created by estimators*[5] and estimators*[7] 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 [66]:

```
rf_best.feature_importances_
```

Out[66]:

```
array([0.08918495, 0.09284378, 0.11257782, 0.06233209, 0.2696204 ,
       0.10678479, 0.03506879, 0.07788308, 0.09125975, 0.02628124,
       0.00810564, 0.00670606, 0.01071035, 0.01064127])
```

In [70]:

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

In [71]:

```
imp_df.sort_values(by="Importance", ascending=False)
```

Out[71]:

	features	Importance
4	Price	0.269620
2	Advertising	0.112578
5	Age	0.106785
1	Income	0.092844
8	ShelveLoc_Good	0.091260
0	CompPrice	0.089185
7	ShelveLoc_Bad	0.077883
3	Population	0.062332
6	Education	0.035069
9	ShelveLoc_Medium	0.026281

	features	Importance
12	US_No	0.010710
13	US_Yes	0.010641
10	Urban_No	0.008106

As seen in the above table Price is most important feature

In [ ]: