In [3]:
```
pip install graphviz
```

```
Requirement already satisfied: graphviz in c:\users\91998\anaconda3\lib\site-packages (0.19.1)
Note: you may need to restart the kernel to use updated packages.
```

Import Necessary Libraries

In [58]:
```
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier as RF
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from io import StringIO
import pydotplus
from sklearn.tree import export_graphviz
import matplotlib.image as mpimg
```

Data collection

In [59]:
```
df = pd.read_csv('Fraud_check (1).csv')
```

In [60]:
```
fraud=df.copy()
```

In [61]:
```
fraud.head()
```

Out[61]:

	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 [62]:
```
fraud.describe().T
```

Out[62]:

	count	mean	std	min	25%	50%	75%	max
Taxable.Income	600.0	55208.375000	26204.827597	10003.0	32871.50	55074.5	78611.75	99619.0
City.Population	600.0	108747.368333	49850.075134	25779.0	66966.75	106493.5	150114.25	199778.0
Work.Experience	600.0	15.558333	8.842147	0.0	8.00	15.0	24.00	30.0

In [63]:
```
fraud.isnull().sum()
```

Out[63]:
```
Undergrad          0
Marital.Status     0
Taxable.Income     0
City.Population    0
Work.Experience    0
Urban              0
dtype: int64
```

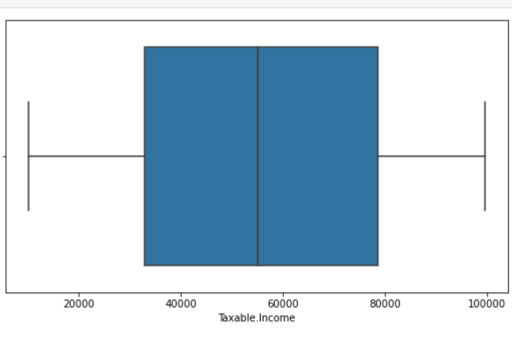
In [64]:
```
fraud.dtypes
```

Out[64]:
```
Undergrad          object
Marital.Status     object
Taxable.Income      int64
City.Population     int64
Work.Experience     int64
Urban              object
dtype: object
```

In [65]:
```
import warnings
warnings.filterwarnings('ignore')
```

Outlier Check

In [66]:
```
ax = sns.boxplot(fraud['Taxable.Income'])
```

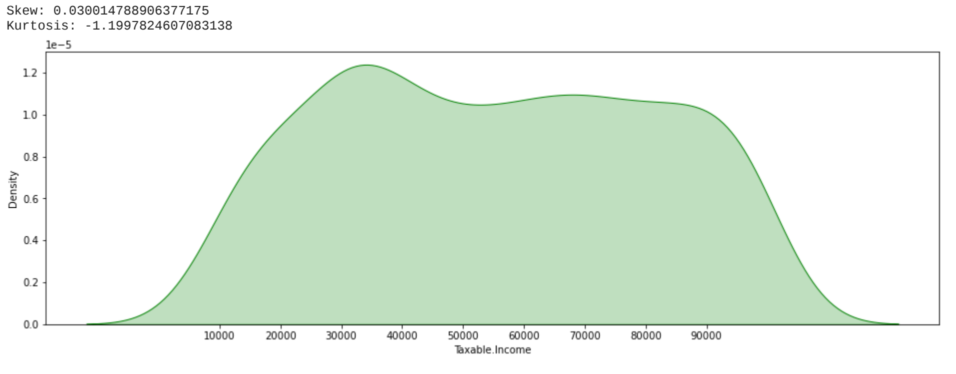


There are no outliers in the data

In [67]:
```
plt.rcParams["figure.figsize"] = 9,5
```

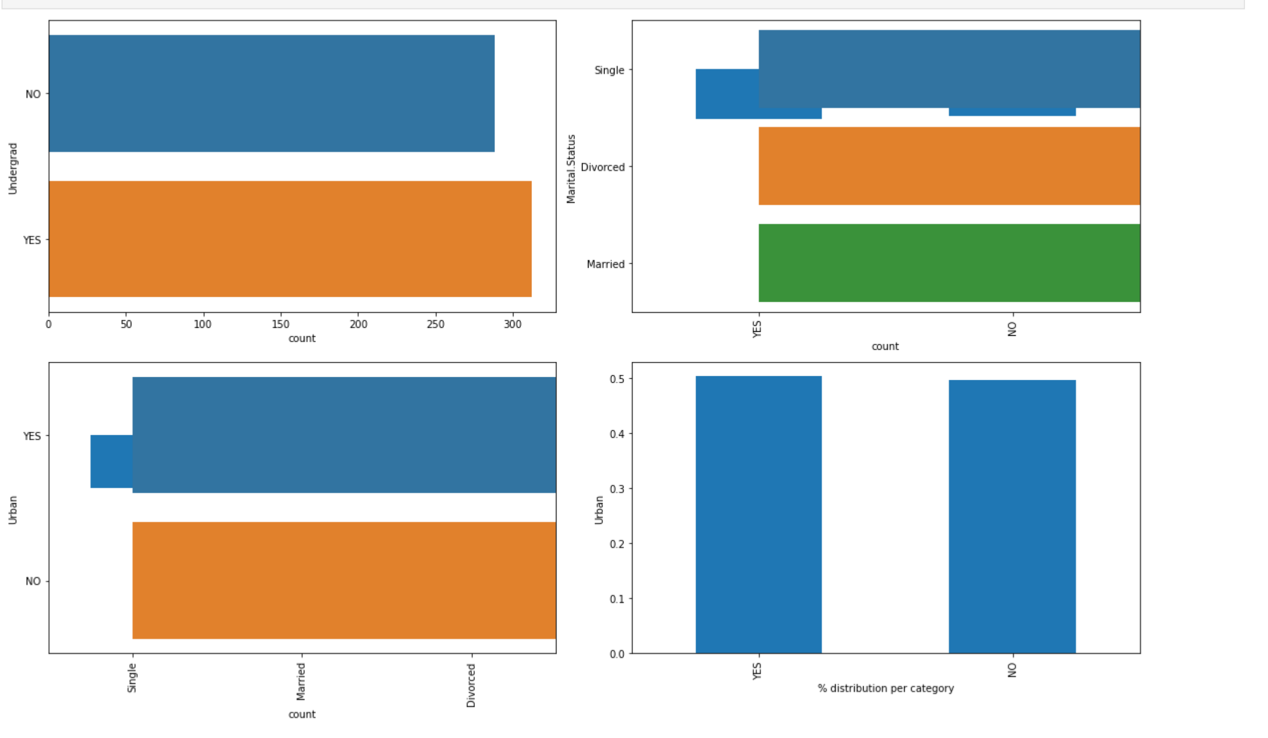
In [68]:
```
plt.figure(figsize=(16,5))
print("Skew: {}".format(fraud['Taxable.Income'].skew()))
print("Kurtosis: {}".format(fraud['Taxable.Income'].kurtosis()))
ax = sns.kdeplot(fraud['Taxable.Income'],shade=True,color='g')
plt.xticks([i for i in range(10000,100000,10000)])
plt.show()
```

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



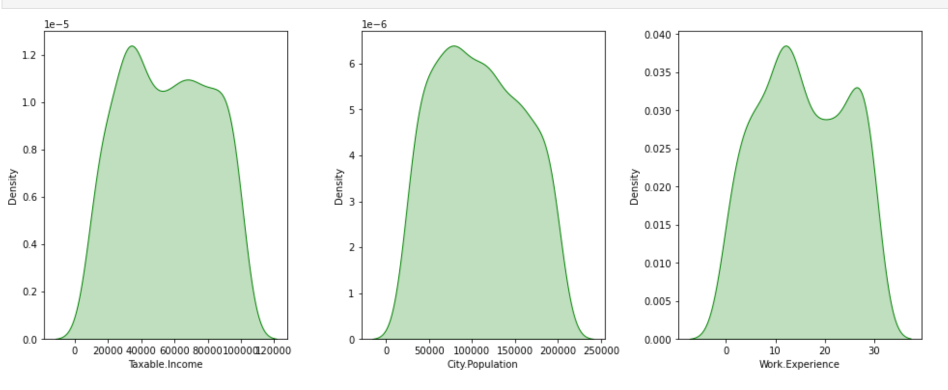
In [69]:
```
obj_colum = fraud.select_dtypes(include='object').columns.tolist()
```

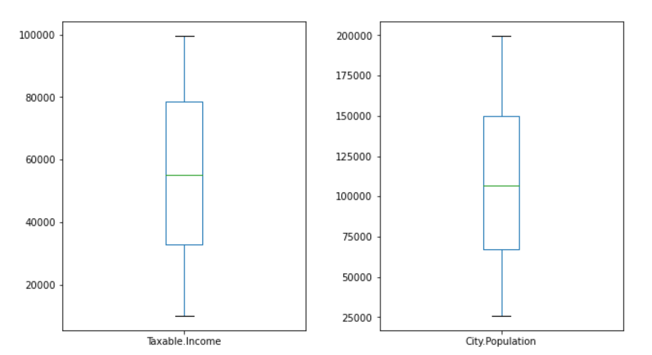
In [70]:
```
plt.figure(figsize=(16,10))
for i,col in enumerate(obj_colum,1):
    plt.subplot(2,2,i)
    sns.countplot(data=fraud,y=col)
    plt.subplot(2,2,i+1)
    fraud[col].value_counts(normalize=True).plot.bar()
    plt.ylabel(col)
    plt.xlabel('% distribution per category')
plt.tight_layout()
plt.show()
```

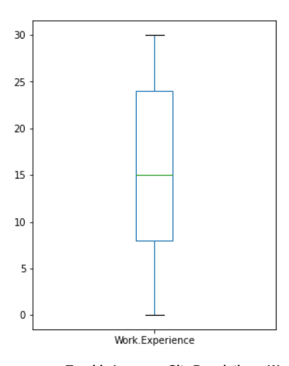


In [71]:
```
num_columns = fraud.select_dtypes(exclude='object').columns.tolist()
```

In [72]:
```
plt.figure(figsize=(18,40))
for i,col in enumerate(num_columns,1):
    plt.subplot(8,4,i)
    sns.kdeplot(df[col],color='g',shade=True)
    plt.subplot(8,4,i+10)
    df[col].plot.box()
plt.tight_layout()
plt.show()
num_data = df[num_columns]
pd.DataFrame(data=[num_data.skew(),num_data.kurtosis()],index=['skewness','kurtosis'])
```







Out[72]:

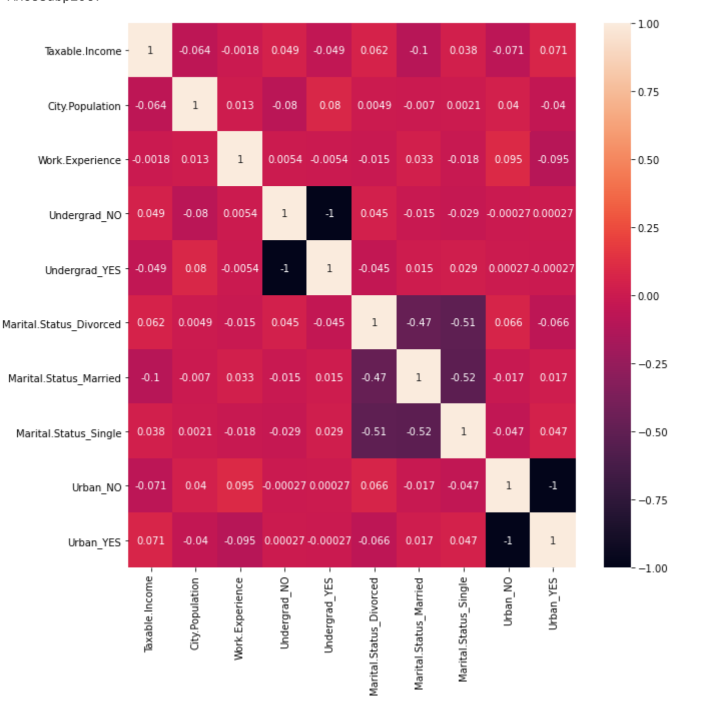
	Taxable.Income	City.Population	Work.Experience
skewness	0.030015	0.125009	0.018529
kurtosis	-1.199782	-1.120154	-1.167524

In [73]:
```
fraud = pd.get_dummies(fraud, columns = ['Undergrad','Marital.Status','Urban'])
```

In [74]:
```
corr = fraud.corr()
```

In [75]:
```
plt.figure(figsize=(10,10))
sns.heatmap(corr,annot=True)
```

Out[75]: <AxesSubplot:>



Randome Forest Model

In [76]:
```
fraud['Taxable.Income']=pd.cut(fraud['Taxable.Income'],bins=[0,30000,100000],labels=['risky','good'])
```

In [77]:
```
list(fraud.columns)
```

Out[77]:
```
['Taxable.Income',
 'City.Population',
 'Work.Experience',
 'Undergrad_NO',
 'Undergrad_YES',
 'Marital.Status_Divorced',
 'Marital.Status_Married',
 'Marital.Status_Single',
 'Urban_NO',
 'Urban_YES']
```

In [78]:
```
X = fraud.iloc[:,1:10]
y = fraud.iloc[:,0]
```

In [79]:
```
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size = 0.2)
```

In [80]:
```
y_train.value_counts()
```

Out[80]:
```
good     376
risky    104
Name: Taxable.Income, dtype: int64
```

In [81]:
```
model =RF(n_jobs=4,n_estimators = 150, oob_score =True,criterion ='entropy')
model.fit(x_train,y_train)
model.oob_score_
```

Out[81]: 0.7291666666666666

In [82]:
```
pred_train = model.predict(x_train)
```

In [83]:
```
accuracy_score(y_train,pred_train)
```

Out[83]: 1.0

In [84]:
```
confusion_matrix(y_train,pred_train)
```

Out[84]:
```
array([[376,   0],
       [  0, 104]], dtype=int64)
```

In [85]:
```
pred_test = model.predict(x_test)
```

In [86]:
```
accuracy_score(y_test,pred_test)
```

Out[86]: 0.7416666666666667

In [87]:
```
confusion_matrix(y_test,pred_test)
```

Out[87]:
```
array([[85, 15],
       [16,  4]], dtype=int64)
```

In [88]:
```
df_t=pd.DataFrame({'Actual':y_test, 'Predicted':pred_test})
```

In [89]:
```
df_t
```

Out[89]:

	Actual	Predicted
473	good	good
161	good	good
133	good	good
599	good	good
545	good	good
...	...	...
519	good	good
281	good	good
63	good	good
556	good	good
553	good	good

120 rows × 2 columns

In [90]:
```
cols = list(fraud.columns)
```

In [91]:
```
predictors = cols[1:10]
target = cols[0]
```

In [92]:
```
tree1 = model.estimators_[20]
```

In [93]:
```
dot_data = StringIO()
```

In [94]:
```
export_graphviz(tree1, out_file = dot_data, feature_names =predictors, class_names = target, filled =True,rounded=True,impurity =False,proportion=False)
```

In [95]:
```
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
```

In [96]:
```
graph.write_png('fraud_full.png')
```

Out[96]: True

Conclusion

In [42]:
```
rf_small = RF(n_estimators=10, max_depth = 3)
```

In [43]:
```
rf_small.fit(x_train,y_train)
```

Out[43]: RandomForestClassifier(max_depth=3, n_estimators=10)

In [44]:
```
tree_small = rf_small.estimators_[5]
```

In [45]:
```
export_graphviz(tree_small, out_file = dot_data, feature_names = predictors, rounded = True, precision = 1)
```

In [46]:
```
graph_small = pydotplus.graph_from_dot_data(dot_data.getvalue())
```

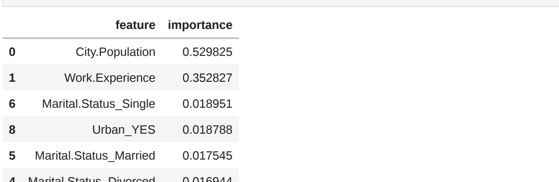
In [47]:
```
graph.write_png('fraud_small.png')
```

Out[47]: True

In [48]:
```
img = mpimg.imread('fraud_full.png')
```

In [49]:
```
plt.imshow(img)
```

Out[49]: <matplotlib.image.AxesImage at 0x1a83eca8ee0>



In [50]:
```
model.feature_importances_
```

Out[50]:
```
array([0.52982467, 0.35282662, 0.01511686, 0.01462354, 0.01694437,
       0.01754481, 0.01895085, 0.01538051, 0.01878779])
```

In [52]:
```
fi = pd.DataFrame({'feature': list(x_train.columns),
                   'importance': model.feature_importances_}).\
                    sort_values('importance', ascending = False)
```

In [53]:
```
fi
```

Out[53]:

	feature	importance
0	City.Population	0.529825
1	Work.Experience	0.352827
6	Marital.Status_Single	0.018951
8	Urban_YES	0.018788
5	Marital.Status_Married	0.017545
4	Marital.Status_Divorced	0.016944
7	Urban_NO	0.015381
2	Undergrad_NO	0.015117
3	Undergrad_YES	0.014624

In [ ]:

In [ ]: