Fraud Check

Use Random Forest 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

In [1]:
```
import warnings
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns



sns.set_style('darkgrid')

import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import  DecisionTreeClassifier
from sklearn import tree
from sklearn.metrics import classification_report
from sklearn import preprocessing
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier



from sklearn.metrics import classification_report, accuracy_score,precision_score,recall_score,f1_score,matt
from sklearn.metrics import confusion_matrix
```

In [3]:
```
fraud_check = pd.read_csv("fraud_Check.csv")
fraud_check.head()
```

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

In [4]:
```
fraud_check.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 600 entries, 0 to 599
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Undergrad        600 non-null    object
 1   Marital.Status   600 non-null    object
 2   Taxable.Income   600 non-null    int64 
 3   City.Population  600 non-null    int64 
 4   Work.Experience  600 non-null    int64 
 5   Urban            600 non-null    object
dtypes: int64(3), object(3)
memory usage: 28.2+ KB
```

In [5]:
```
categorical_features = fraud_check.describe(include=["object"]).columns
categorical_features
```

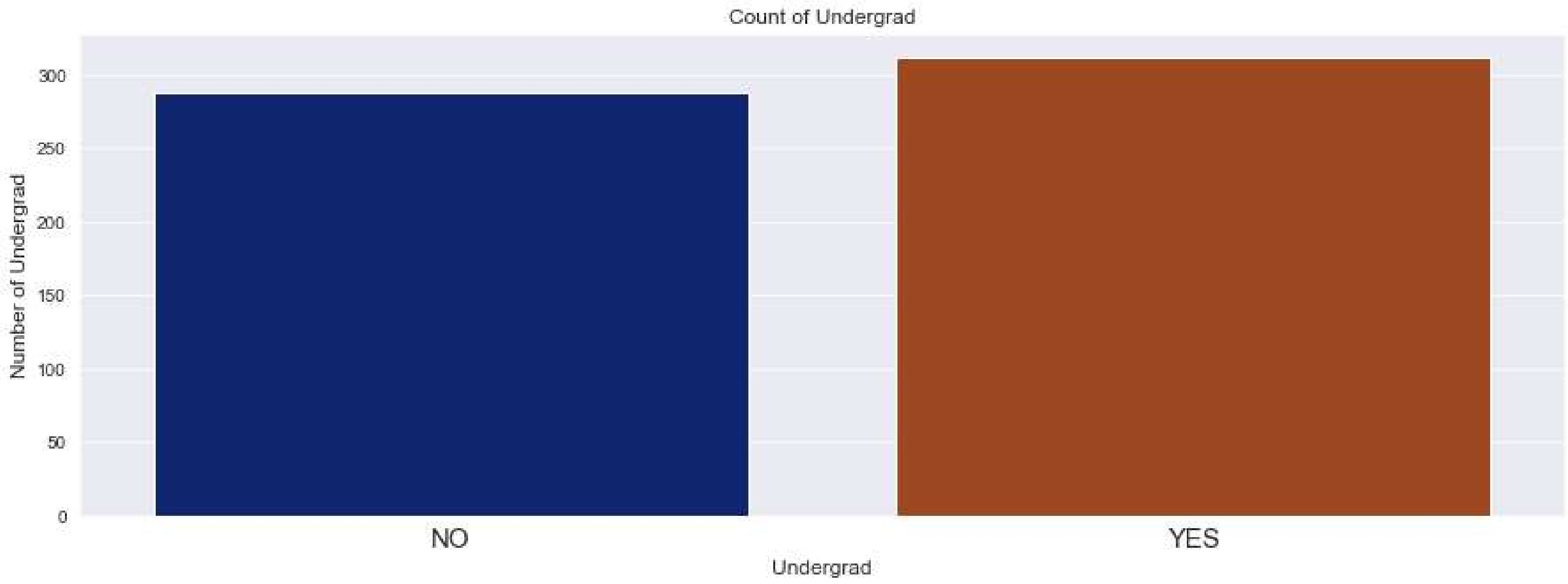
Out[5]: `Index(['Undergrad', 'Marital.Status', 'Urban'], dtype='object')`

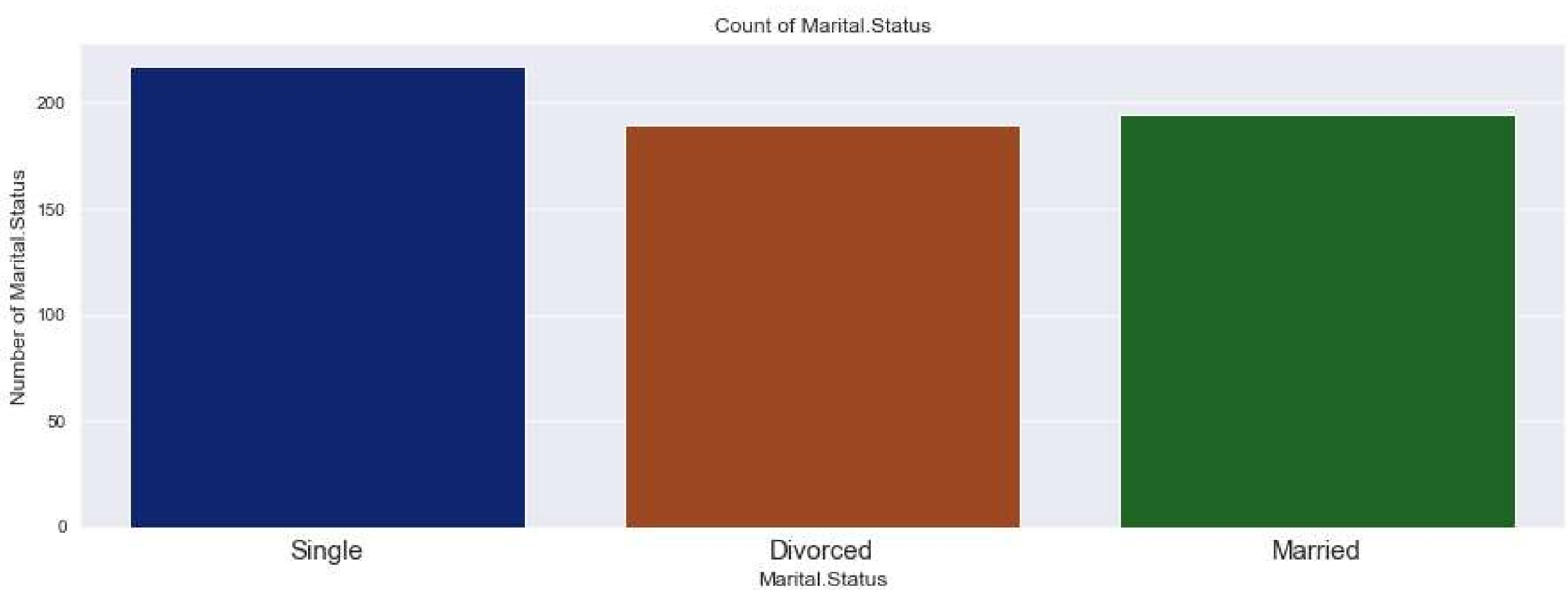
In [6]:
```
numerical_features = fraud_check.describe(include=["int64"]).columns
numerical_features
```

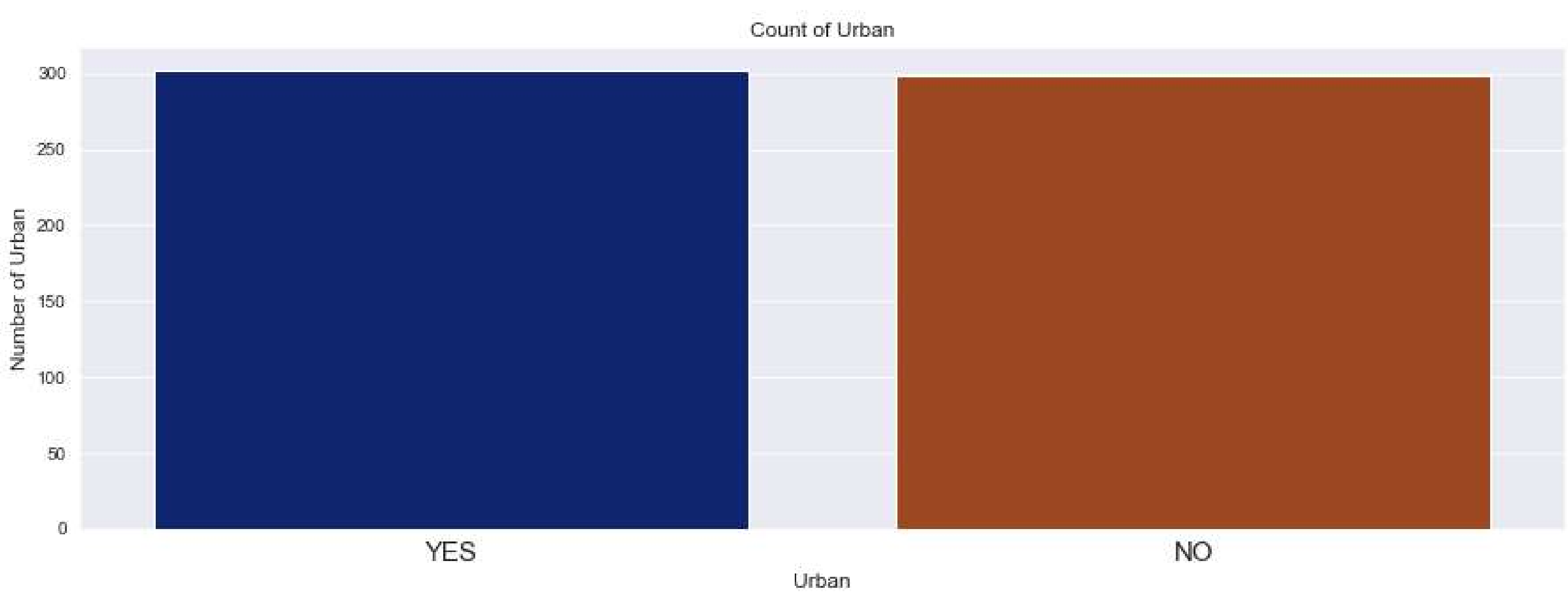
Out[6]: `Index(['Taxable.Income', 'City.Population', 'Work.Experience'], dtype='object')`

In [7]:
```
print(categorical_features)



for idx, column in enumerate(categorical_features):
    plt.figure(figsize=(15, 5))
    df = fraud_check.copy()
    unique = df[column].value_counts(ascending=True);

    #plt.subplot(1, len(categorical_features), idx+1)
    plt.title("Count of "+ column)
    sns.countplot(data=fraud_check, x=column,palette = "dark")
    #plt.bar(unique.index, unique.values);
    plt.xticks(rotation = 0, size = 15)

    plt.xlabel(column, fontsize=12)
    plt.ylabel("Number of "+ column, fontsize=12)
```

```
Index(['Undergrad', 'Marital.Status', 'Urban'], dtype='object')
```

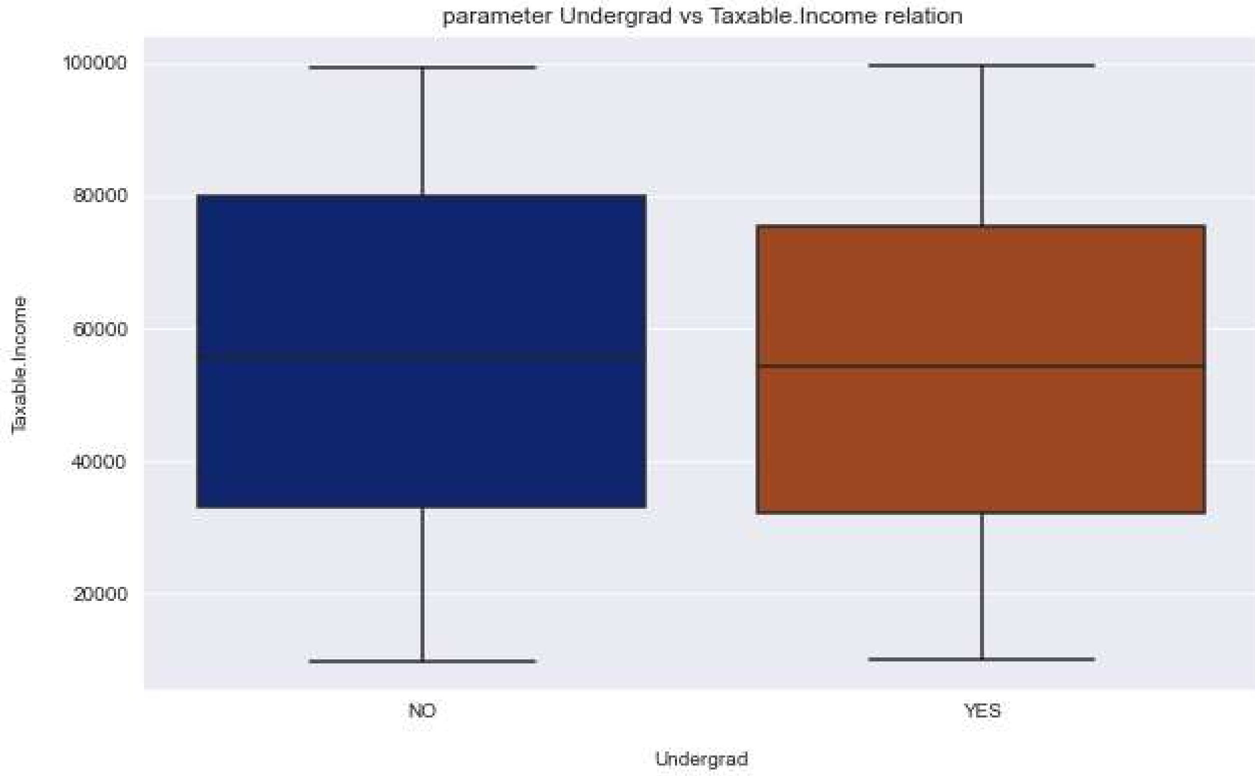




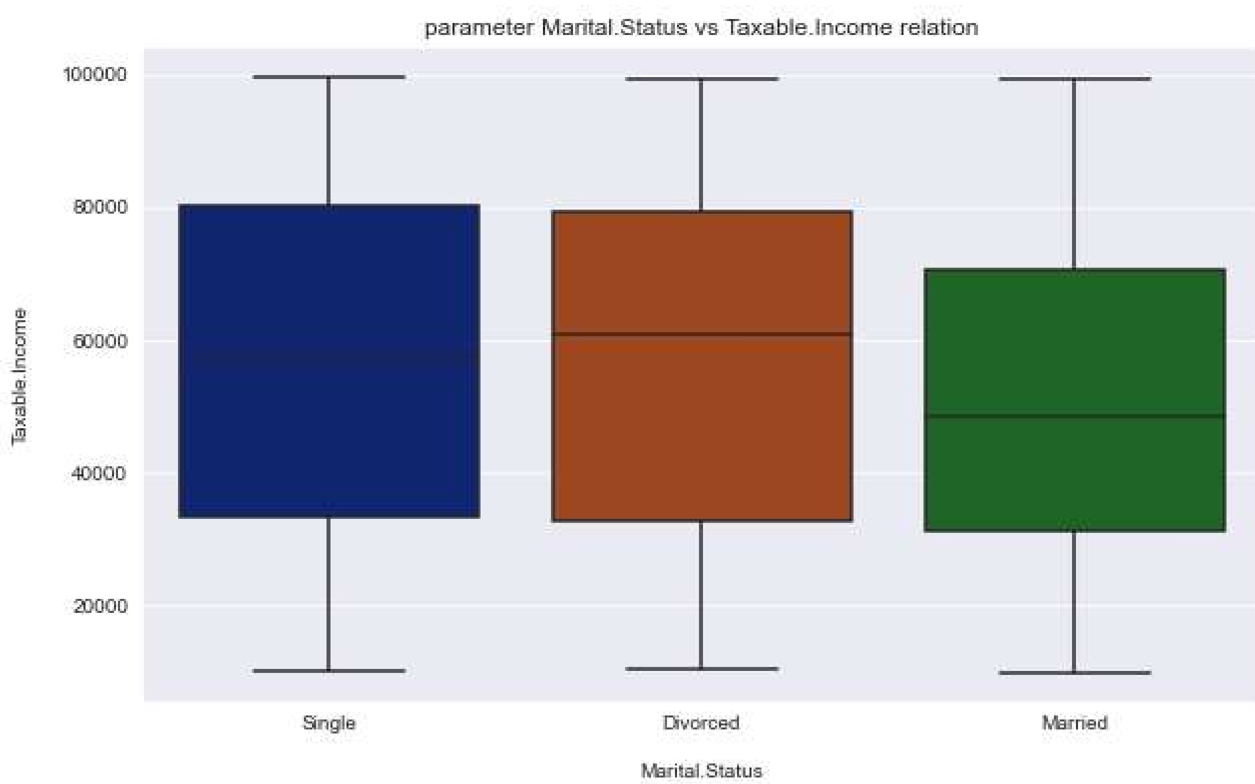


In [8]:
```
def boxplot(x_param, y_param):
    plt.figure(figsize=(10,6))
    sns.boxplot(x=x_param, data=fraud_check,y=y_param, palette = "dark")
    plt.xlabel('\n'+ x_param)
    plt.ylabel(y_param + '\n')
    plt.title("parameter " + x_param + " vs " + y_param + " relation")
    plt.show()
```

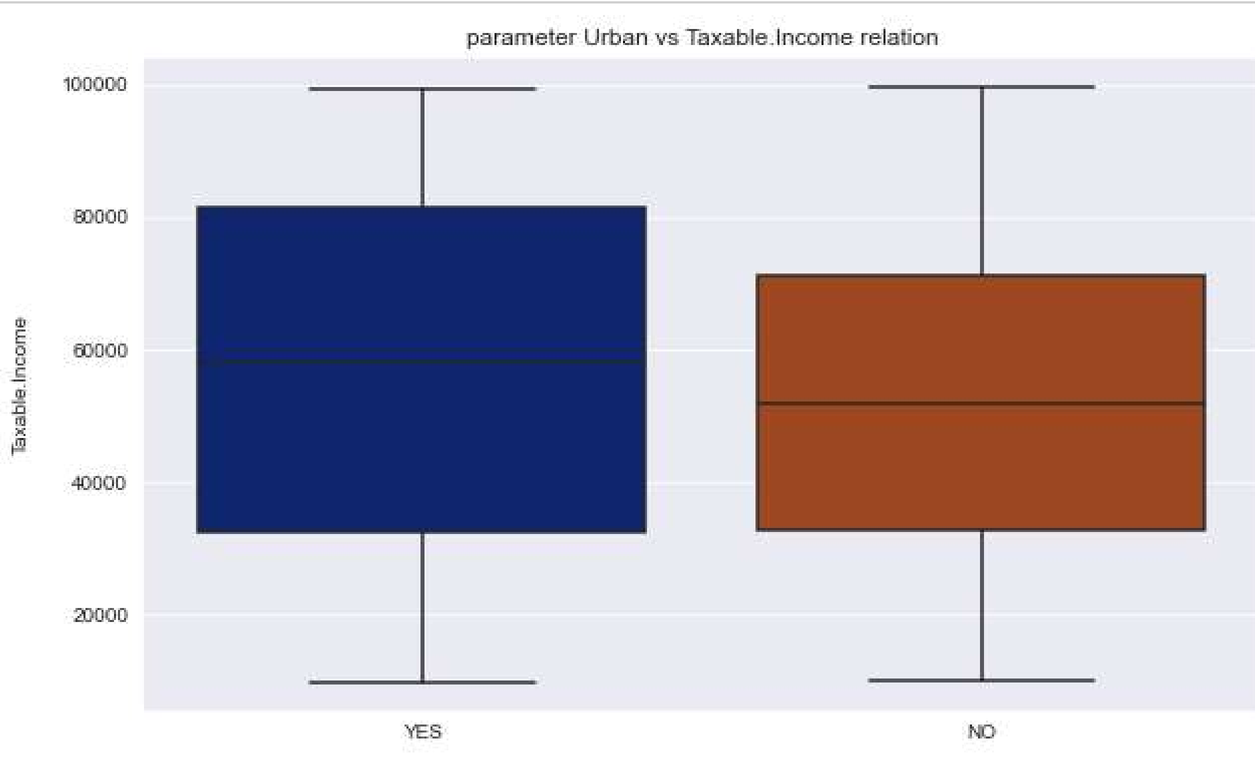
In [9]:
```
boxplot('Undergrad','Taxable.Income')
```



In [10]:
```
boxplot('Marital.Status', 'Taxable.Income')
```

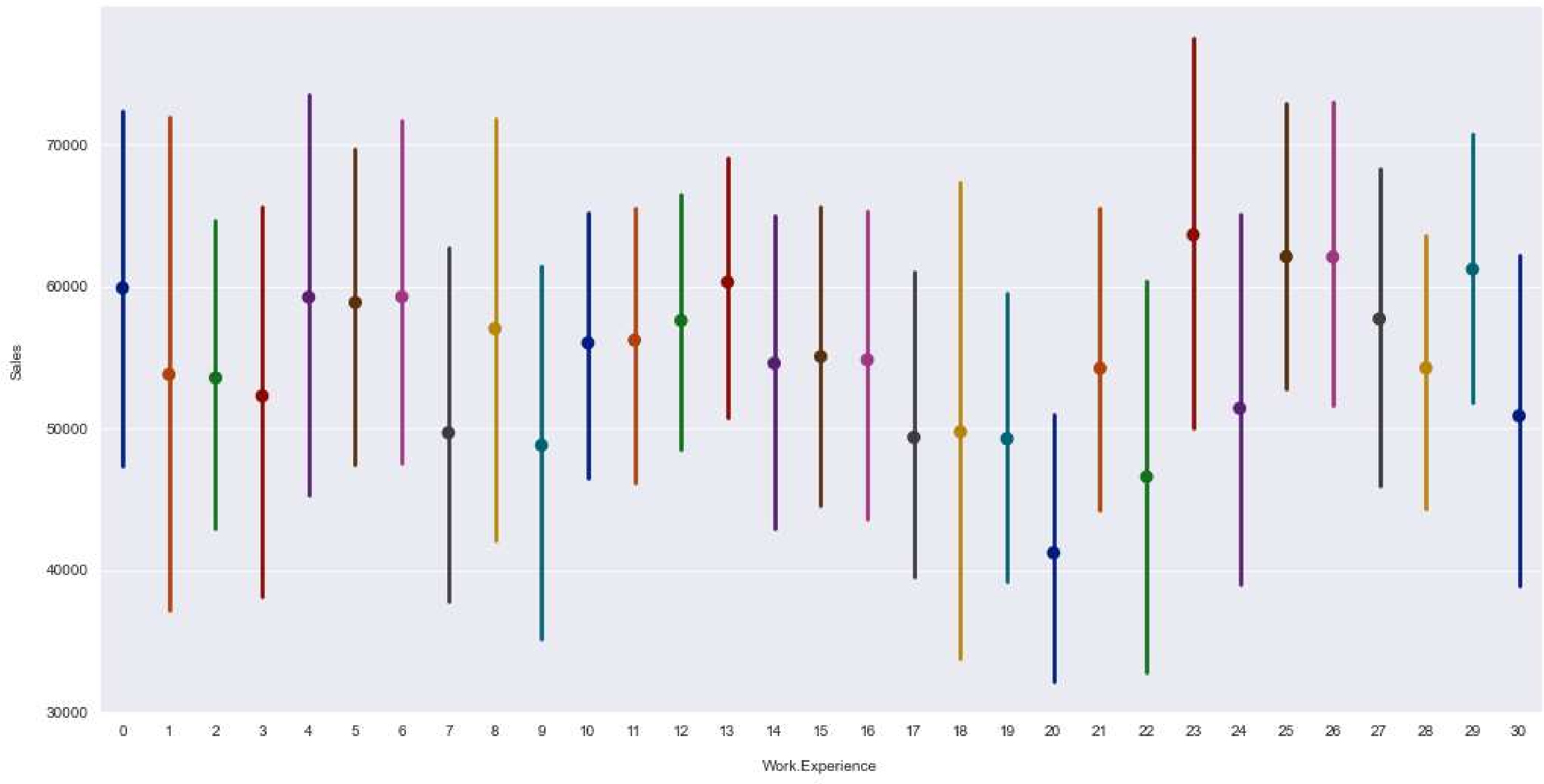


In [11]:
```
boxplot('Urban','Taxable.Income')
```



In [12]:
```
def factorplot(param):
    sns.factorplot(x =param, size = 7, aspect = 2, data = fraud_check, y= "Taxable.Income", palette = "dark"
    plt.xlabel("\n" + param)
    plt.ylabel("Sales\n")
    plt.show()
```

In [13]:
```
factorplot("Work.Experience")
```



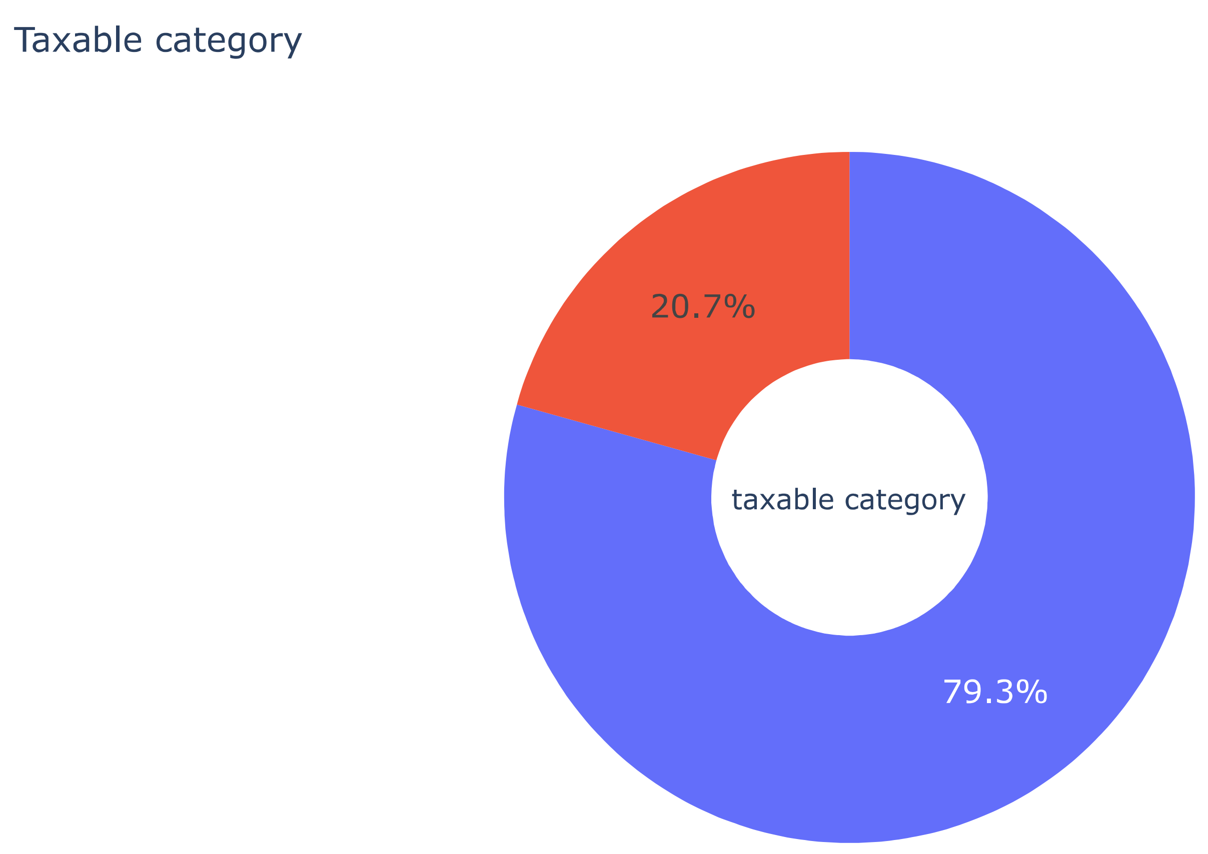
In [14]:
```
fraud_check["Taxable.Income"].min()
```

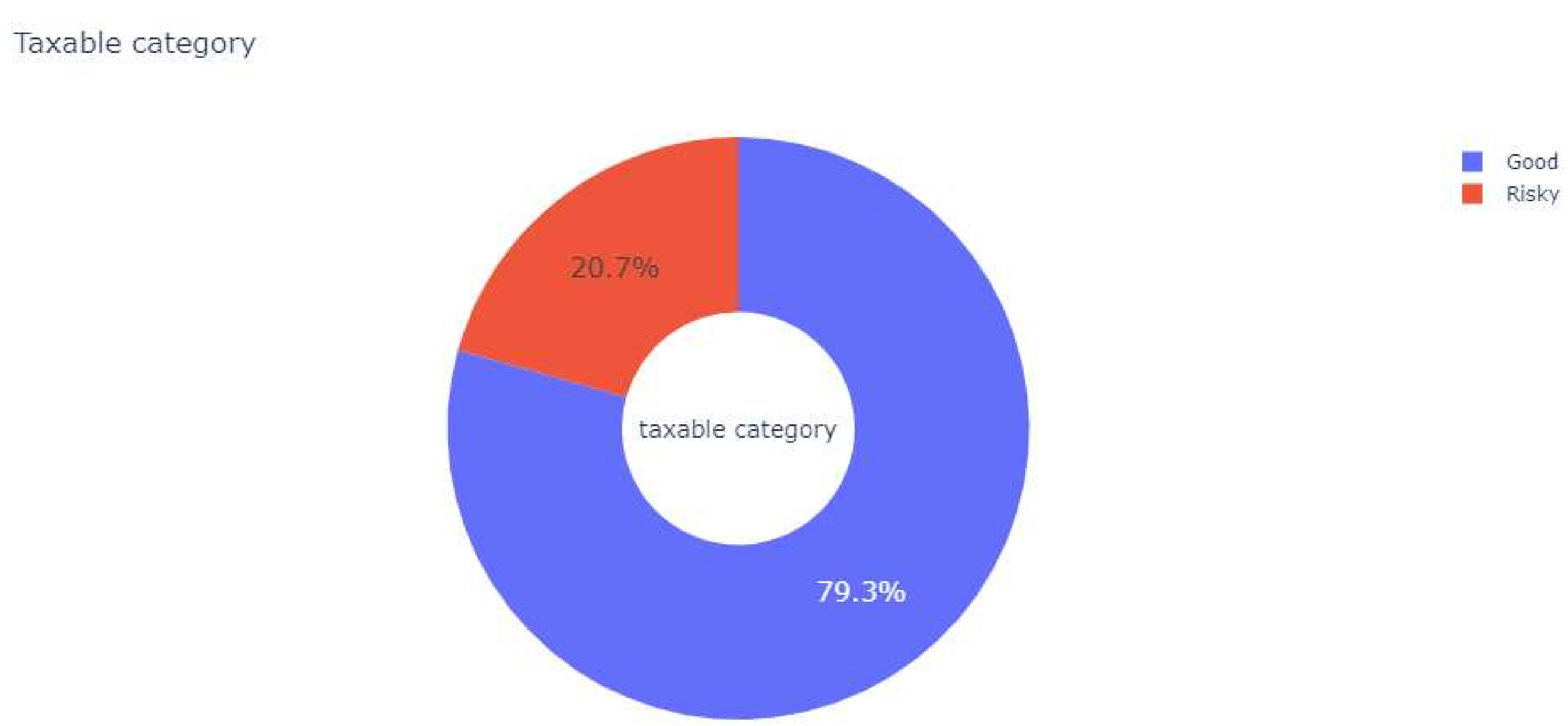
Out[14]:
```
10003
```

In [57]:
```
# Converting taxable_income <= 30000 as "Risky" and others are "Good"
fraud_check['taxable_category'] = pd.cut(x = fraud_check['Taxable.Income'], bins = [10002,30000,99620], labe
fraud_check.head()
```

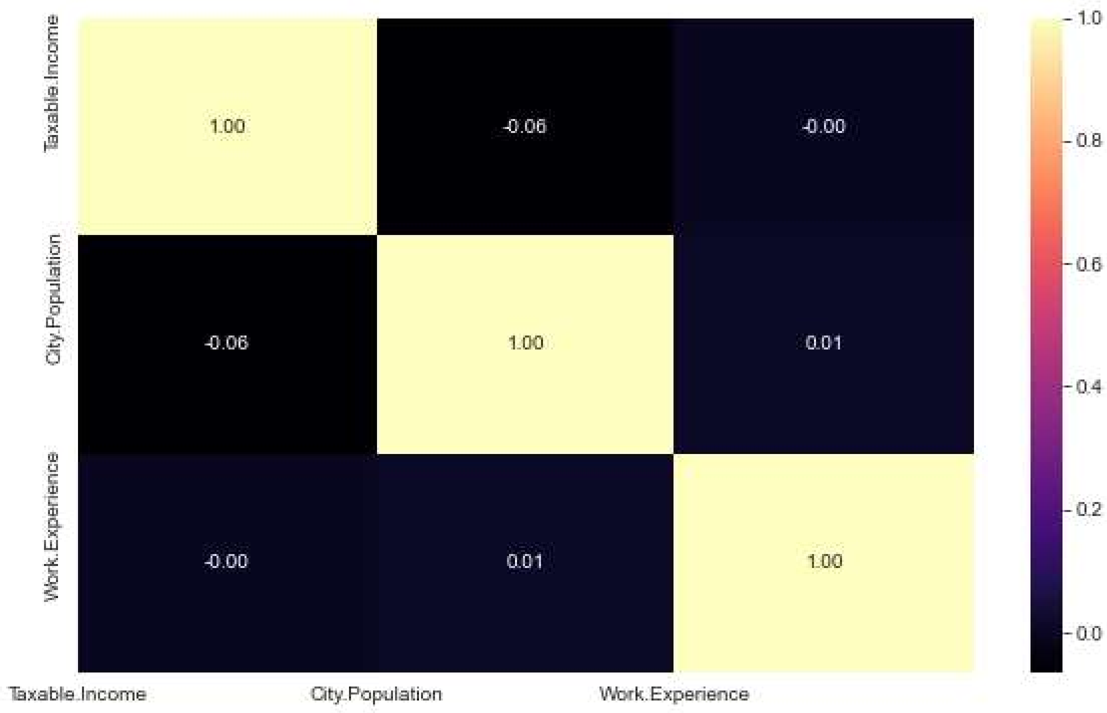
Out[57]:
```
Index(['Undergrad', 'Marital.Status', 'Taxable.Income', 'City.Population',
       'Work.Experience', 'Urban'],
      dtype='object')
```

In [16]:
```
type_ = ['Good', 'Risky']
fig = make_subplots(rows=1, cols=1)

fig.add_trace(go.Pie(labels=type_, values=fraud_check['taxable_category'].value_counts(), name="taxable_cate

# Use `hole` to create a donut-like pie chart
fig.update_traces(hole=.4, hoverinfo="label+percent+name", textfont_size=16)

fig.update_layout(
    title_text="Taxable category",
    # Add annotations in the center of the donut pies.
    annotations=[dict(text='taxable category', x=0.5, y=0.5, font_size=14, showarrow=False)])
fig.show()
```





In [17]:
```
corr = fraud_check.corr()

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(corr, cmap='magma', annot=True, fmt=".2f")

plt.xticks(range(len(corr.columns)), corr.columns);

plt.yticks(range(len(corr.columns)), corr.columns)

plt.show()
```



In [18]:
```
fraud_check["taxable_category"][fraud_check["taxable_category"] == 'Risky'].groupby(by = fraud_check.Undergr
```

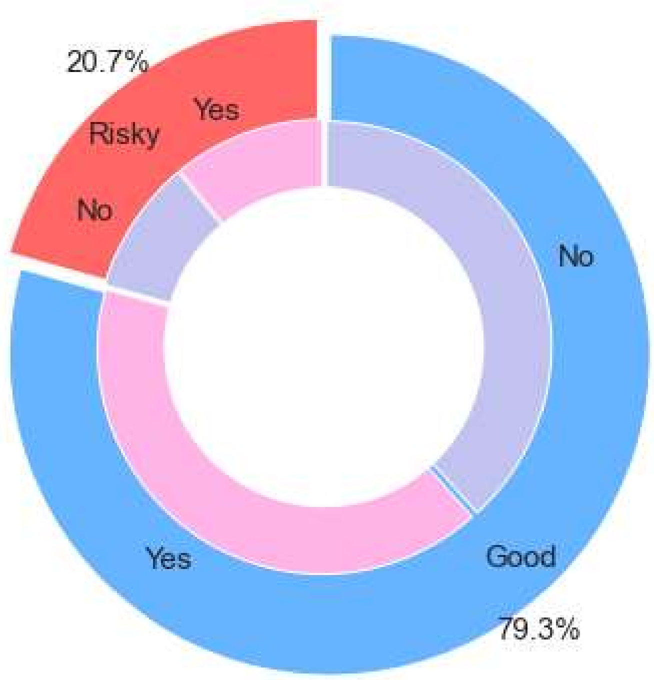
Out[18]:
```
Undergrad
NO     58
YES    66
Name: taxable_category, dtype: int64
```

In [19]:
```
fraud_check["taxable_category"][fraud_check["taxable_category"] == 'Good'].groupby(by = fraud_check.Undergra
```

Out[19]:
```
Undergrad
NO     230
YES    246
Name: taxable_category, dtype: int64
```

In [20]:
```
plt.figure(figsize=(6, 6))
labels =["Risky", "Good"]
values = [fraud_check["taxable_category"][fraud_check["taxable_category"] == 'Risky'].groupby(by = fraud_che
         fraud_check["taxable_category"][fraud_check["taxable_category"] == 'Good'].groupby(by = fraud_check
labels_gender = ["Yes","No","Yes","No"]
sizes_gender = [66,58 , 246,230]
colors = ['#ff6666', '#66b3ff']
colors_gender = ['#ffb3e6','#c2c2f0','#ffb3e6', '#c2c2f0']
explode = (0.3,0.3)
explode_gender = (0.1,0.1,0.1,0.1)
textprops = {"fontsize":15}
#Plot
plt.pie(values, labels=labels,autopct='%1.1f%%',pctdistance=1.08, labeldistance=0.8,colors=colors, startangl
plt.pie(sizes_gender,labels=labels_gender,colors=colors_gender,startangle=90, explode=explode_gender,radius=
#Draw circle
centre_circle = plt.Circle((0,0),5,color='black', fc='white',linewidth=0)
fig = plt.gcf()
fig.gca().add_artist(centre_circle)

plt.title('Taxable income distribution w.r.t Graduation status: Yes(Undergrad), No(Grad)', fontsize=15, y=1.

# show plot

plt.axis('equal')
plt.tight_layout()
plt.show()
```

Taxable income distribution w.r.t Graduation status: Yes(Undergrad), No(Grad)



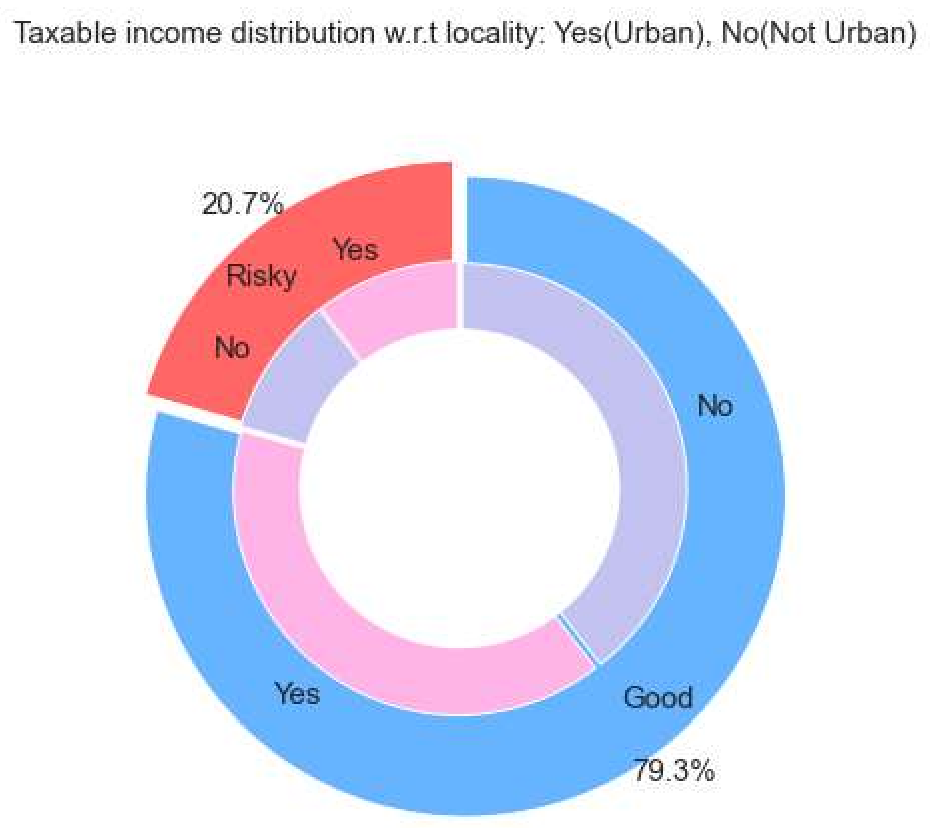
In [21]:
```
fraud_check["taxable_category"][fraud_check["taxable_category"] == 'Risky'].groupby(by = fraud_check.Urban).
```

Out[21]:
```
Urban
NO     61
YES    63
Name: taxable_category, dtype: int64
```

In [22]:
```
fraud_check["taxable_category"][fraud_check["taxable_category"] == 'Good'].groupby(by = fraud_check.Urban).c
```

Out[22]:
```
Urban
NO     237
YES    239
Name: taxable_category, dtype: int64
```

In [23]:
```
plt.figure(figsize=(6, 6))
labels =["Risky", "Good"]
values = [fraud_check["taxable_category"][fraud_check["taxable_category"] == 'Risky'].groupby(by = fraud_che
         fraud_check["taxable_category"][fraud_check["taxable_category"] == 'Good'].groupby(by = fraud_check
labels_gender = ["Yes","No","Yes","No"]
sizes_gender = [63,61 , 239,237]
colors = ['#ff6666', '#66b3ff']
colors_gender = ['#ffb3e6','#c2c2f0','#ffb3e6', '#c2c2f0']
explode = (0.3,0.3)
explode_gender = (0.1,0.1,0.1,0.1)
textprops = {"fontsize":15}
#Plot
plt.pie(values, labels=labels,autopct='%1.1f%%',pctdistance=1.08, labeldistance=0.8,colors=colors, startangl
plt.pie(sizes_gender,labels=labels_gender,colors=colors_gender,startangle=90, explode=explode_gender,radius=
#Draw circle
centre_circle = plt.Circle((0,0),5,color='black', fc='white',linewidth=0)
fig = plt.gcf()
fig.gca().add_artist(centre_circle)

plt.title('Taxable income distribution w.r.t locality: Yes(Urban), No(Not Urban)', fontsize=15, y=1.1)

# show plot

plt.axis('equal')
plt.tight_layout()
plt.show()
```



In [24]:
```
fraud_check["taxable_category"][fraud_check["taxable_category"] == 'Risky'].groupby(by = fraud_check["Marita
```

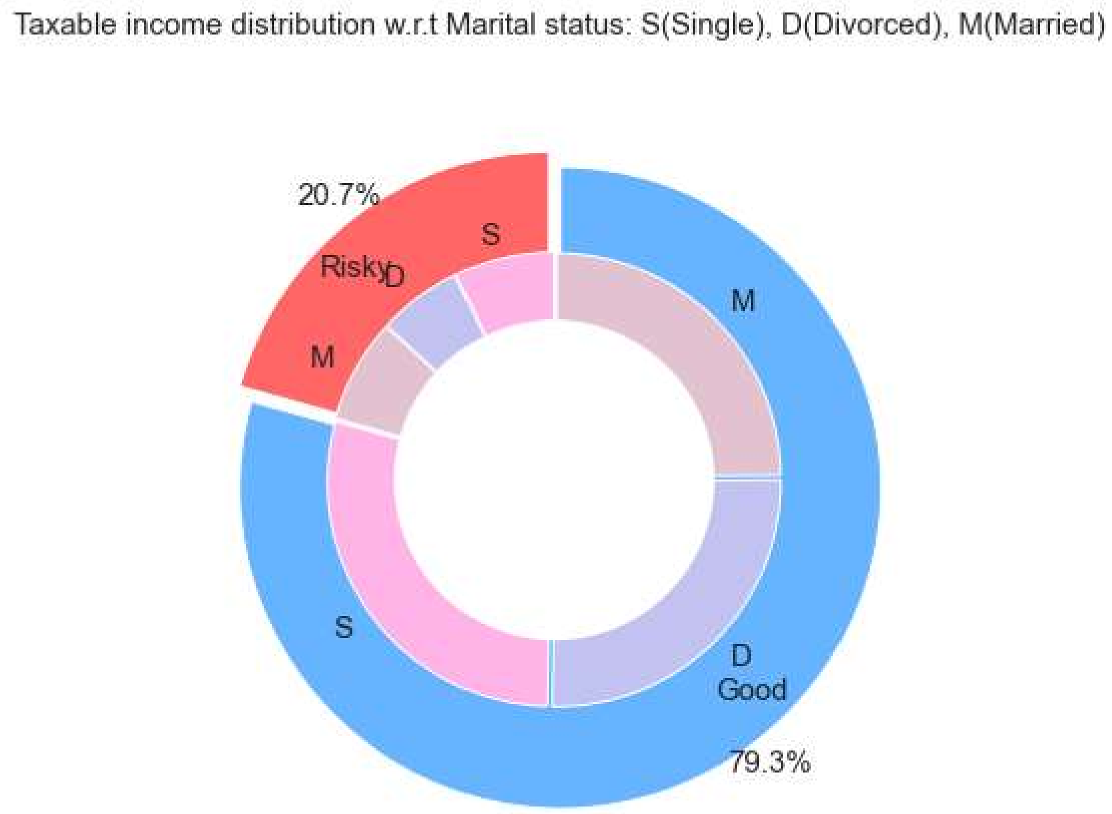
Out[24]:
```
Marital.Status
Divorced    36
Married     45
Single      43
Name: taxable_category, dtype: int64
```

In [25]:
```
fraud_check["taxable_category"][fraud_check["taxable_category"] == 'Good'].groupby(by = fraud_check["Marital
```

Out[25]:
```
Marital.Status
Divorced    153
Married     149
Single      174
Name: taxable_category, dtype: int64
```

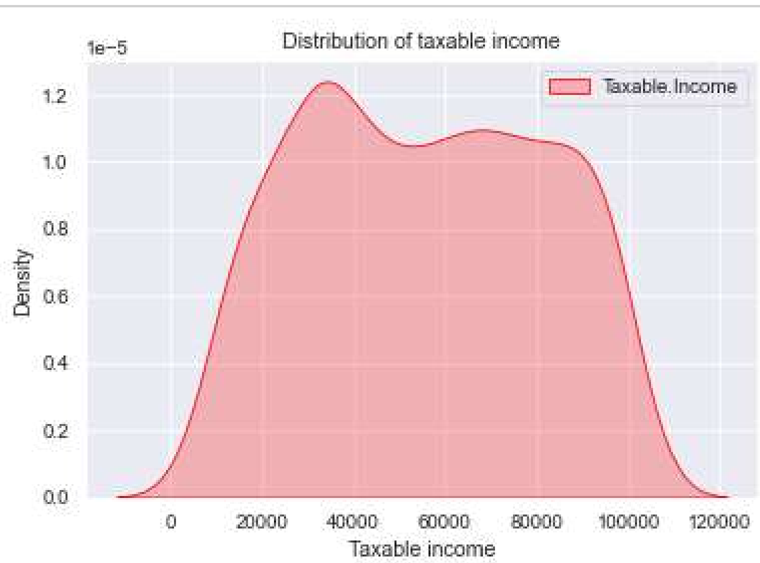
In [26]:

```
plt.figure(figsize=(6, 6))
labels =["Risky", "Good"]
values = [fraud_check["taxable_category"][fraud_check["taxable_category"] == 'Risky'].groupby(by = fraud_che
         fraud_check["taxable_category"][fraud_check["taxable_category"] == 'Good'].groupby(by = fraud_check
labels_gender = ["S","D","M","S","D", "M"]
sizes_gender = [43,36,45,174,153,149]
colors = ['#ff6666', '#66b3ff']
colors_gender = ['#ffb3e6','#c2c2f0','#e2c2d0','#ffb3e6', '#c2c2f0', '#e2c2d0']
explode = (0.3,0.3)
explode_gender = (0.1,0.1,0.1,0.1,0.1,0.1)
textprops = {"fontsize":15}
#Plot
plt.pie(values, labels=labels,autopct='%1.1f%%',pctdistance=1.08, labeldistance=0.8,colors=colors, startangl
plt.pie(sizes_gender,labels=labels_gender,colors=colors_gender,startangle=90, explode=explode_gender,radius=
#Draw circle
centre_circle = plt.Circle((0,0),5,color='black', fc='white',linewidth=0)
fig = plt.gcf()
fig.gca().add_artist(centre_circle)

plt.title('Taxable income distribution w.r.t Marital status: S(Single), D(Divorced), M(Married)', fontsize=1

# show plot

plt.axis('equal')
plt.tight_layout()
plt.show()
```

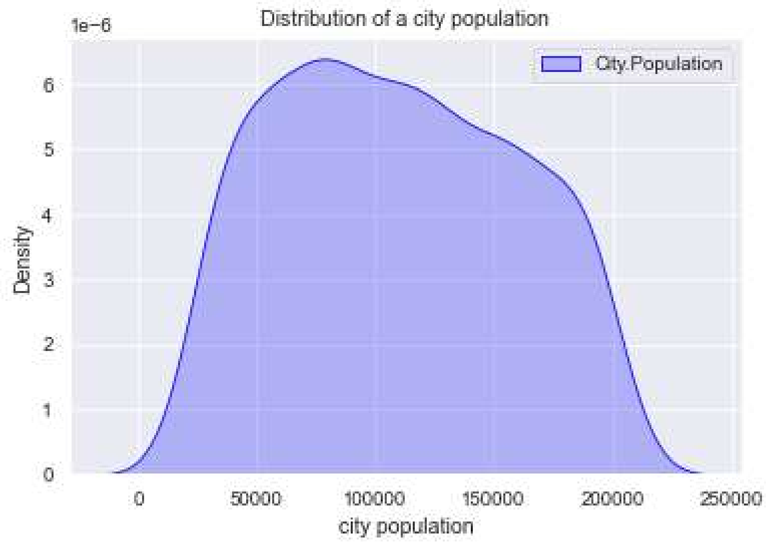


In [27]:

```
sns.set_context("paper",font_scale=1.1)
ax = sns.kdeplot(fraud_check["Taxable.Income"],
                color="Red", shade = True);
ax.legend(["Taxable.Income"],loc='upper right');
ax.set_ylabel('Density');
ax.set_xlabel('Taxable income');
ax.set_title('Distribution of taxable income');
```



In [28]:
```
sns.set_context("paper",font_scale=1.1)

ax = sns.kdeplot(fraud_check["City.Population"],
                 color="Blue", shade= True);
ax.legend(["City.Population"],loc='upper right');
ax.set_ylabel('Density');
ax.set_xlabel('city population');
ax.set_title('Distribution of a city population');
```



In [29]:
```
"""# Label encoding

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
data_copy= fraud_check.copy()
le = LabelEncoder()
# Label Encoding will be used for columns with 2 or less unique values
le_count = 0
for col in data_copy.columns[0:]:
    if len(list(data_copy[col].unique())) <= 3:
        le.fit(data_copy[col])
        data_copy[col] = le.transform(data_copy[col])
        le_count += 1
print('{} columns were label encoded.'.format(le_count))"""


# Converting categorical variables into dummy variables
data_= fraud_check.copy()
data_copy = pd.get_dummies(data_.iloc[:,:-1])
```

In [30]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
data_copy["taxable_category"] =fraud_check.taxable_category
le = LabelEncoder()
le.fit(data_copy["taxable_category"])
data_copy["taxable_category"]=le.transform(data_copy["taxable_category"])
data_copy.head()
```

Out[30]:

	Taxable.Income	City.Population	Work.Experience	Undergrad_NO	Undergrad_YES	Marital.Status_Divorced	Marital.Status_Married
0	68833	50047	10	1	0	0	0
1	33700	134075	18	0	1	1	0
2	36925	160205	30	1	0	0	1
3	50190	193264	15	0	1	0	0
4	81002	27533	28	1	0	0	1

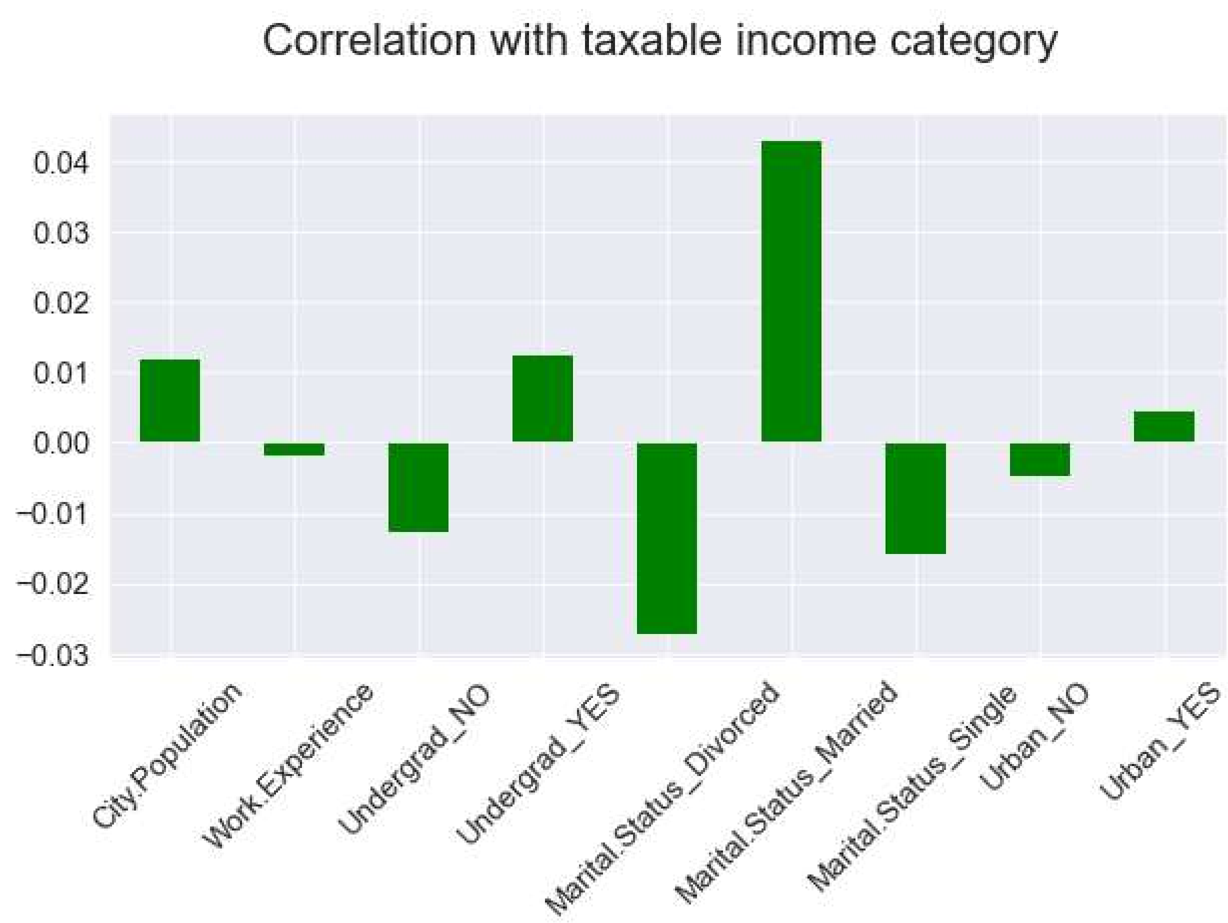
In [31]:
```
fraudCheck_data = data_copy.drop('Taxable.Income', axis = 1)
fraudCheck_data.head()
```

Out[31]:

	City.Population	Work.Experience	Undergrad_NO	Undergrad_YES	Marital.Status_Divorced	Marital.Status_Married	Marital.Status_Sin
0	50047	10	1	0	0	0	
1	134075	18	0	1	1	0	
2	160205	30	1	0	0	1	
3	193264	15	0	1	0	0	
4	27533	28	1	0	0	1	

In [32]:
```
data2 = fraudCheck_data.iloc[:,:-1]

correlations = data2.corrwith(fraudCheck_data.taxable_category)
correlations = correlations[correlations!=1]
positive_correlations = correlations[correlations >0].sort_values(ascending = False)
negative_correlations =correlations[correlations<0].sort_values(ascending = False)

correlations.plot.bar(
        figsize = (10, 5),
        fontsize = 15,
        color = 'green',
        rot = 45, grid = True)
plt.title('Correlation with taxable income category \n',
horizontalalignment="center", fontstyle = "normal",
fontsize = "22", fontfamily = "sans-serif")
```

Out[32]:
```
Text(0.5, 1.0, 'Correlation with taxable income category \n')
```



In [33]:
```
y = fraudCheck_data['taxable_category']
X = fraudCheck_data.drop('taxable_category', axis = 1)
```

In [34]:
```
def norm_func(i):
    x= (i-i.min())/(i.max()-i.min())
    return (x)

X_ =norm_func(X)
X_.head()
```

Out[34]:

	City.Population	Work.Experience	Undergrad_NO	Undergrad_YES	Marital.Status_Divorced	Marital.Status_Married	Marital.Status_Sin
0	0.139472	0.333333	1.0	0.0	0.0	0.0	
1	0.622394	0.600000	0.0	1.0	1.0	0.0	
2	0.772568	1.000000	1.0	0.0	0.0	1.0	
3	0.962563	0.500000	0.0	1.0	0.0	0.0	
4	0.010081	0.933333	1.0	0.0	0.0	1.0	

In [35]:
```
X_train, X_test, y_train, y_test = train_test_split(X_, y, test_size=0.33, random_state=42)
```

In [36]:
```
print('Shape of x_train: ', X_train.shape)
print('Shape of x_test: ', X_test.shape)
print('Shape of y_train: ', y_train.shape)
print('Shape of y_test: ', y_test.shape)
```

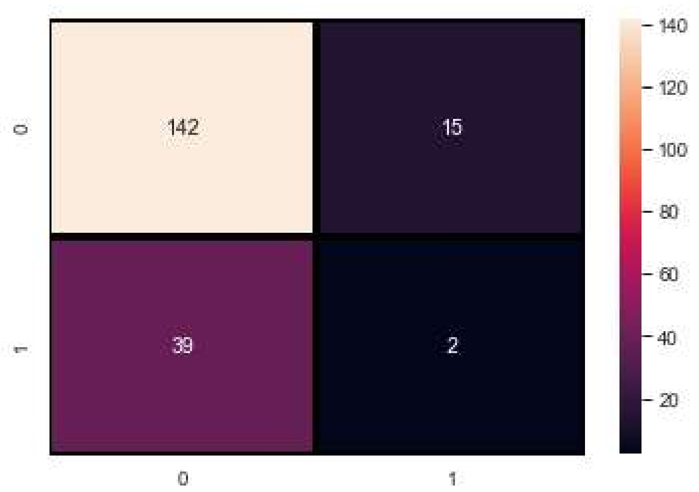
```
Shape of x_train:  (402, 9)
Shape of x_test:  (198, 9)
Shape of y_train:  (402,)
Shape of y_test:  (198,)
```

In [38]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier


decision_tree = DecisionTreeClassifier(criterion = 'entropy', max_depth= 5)
decision_tree.fit(X_train,y_train)

pred1 = decision_tree.predict(X_test)
accuracy_test1 = accuracy_score(y_test,pred1)
accuracy_test1
```

Out[38]: 0.7272727272727273

In [39]:
```
sns.heatmap(confusion_matrix(y_test, pred1),annot=True,fmt = "d",linecolor="k",linewidths=3)
```

Out[39]: <AxesSubplot:>



In [46]:
```
kfold = KFold(n_splits=9, random_state=42)

results = cross_val_score(decision_tree, X_train, y_train, cv=kfold)
print(results.mean())
```

```
0.7834455667789001
```

In [47]:
```
param_dict = {
    "criterion":["gini","entropy"],
    "max_depth":range(1,10),
    "min_samples_split":range(1,10),
    "min_samples_leaf":range(1,10)
}

grid = GridSearchCV(decision_tree,
                   param_grid = param_dict,
                   cv=kfold,
                   verbose=1,
                   n_jobs=6)

grid.fit(X_train,y_train)

model1 = grid.best_estimator_
```

```
Fitting 9 folds for each of 1458 candidates, totalling 13122 fits

[Parallel(n_jobs=6)]: Using backend LokyBackend with 6 concurrent workers.
[Parallel(n_jobs=6)]: Done  39 tasks      | elapsed:   11.9s
[Parallel(n_jobs=6)]: Done 605 tasks      | elapsed:   17.6s
[Parallel(n_jobs=6)]: Done 2586 tasks      | elapsed:   23.5s
[Parallel(n_jobs=6)]: Done 5386 tasks      | elapsed:   31.5s
[Parallel(n_jobs=6)]: Done 10634 tasks      | elapsed:   45.9s
[Parallel(n_jobs=6)]: Done 13122 out of 13122 | elapsed:   52.4s finished
```

In [48]:
```
grid.best_score_
```

Out[48]: 0.7984287317620651

In [49]:
```
predict_output = model1.predict(X_test)
accuracy_test_1 = accuracy_score(y_test,predict_output)
accuracy_test_1
```

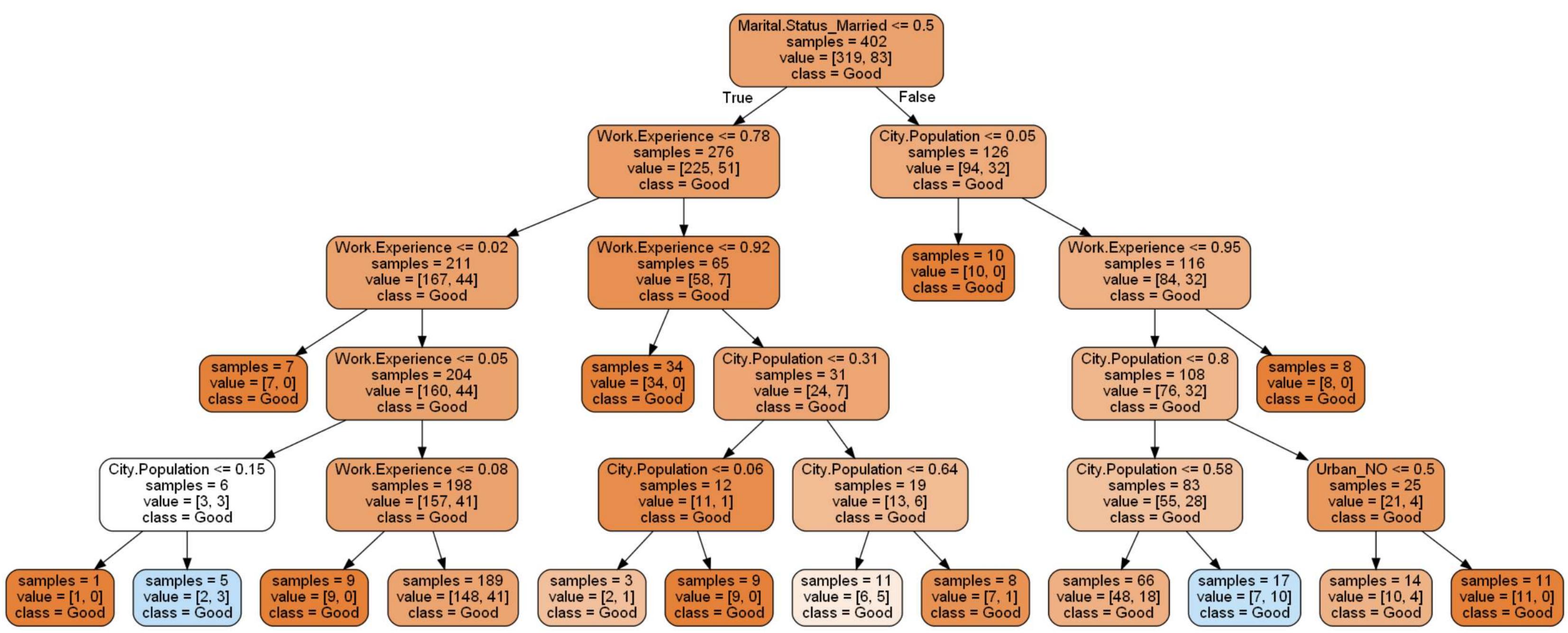
Out[49]: 0.7929292929292929

Graphviz

In [64]:

```
from sklearn.tree import export_graphviz
from six import StringIO
import pydotplus
from IPython.display import Image

colnames = list(X_.columns)
predictors = colnames[0:9]
target = fraud_check.taxable_category
tree1 = grid.estimator
dot_data = StringIO()


import os
os.environ["PATH"] += os.pathsep + 'C:\Program Files\Graphviz/bin/'
export_graphviz(tree1,out_file = dot_data,
                feature_names =predictors,
                class_names = target, filled =True,
                rounded=True,impurity =False,proportion=False,precision =2)


graph = pydotplus.graph_from_dot_data(dot_data.getvalue())


Image(graph.create_png())
```

Out[64]:



In [65]:

```
##Creating pdf file
graph.write_pdf('FraudCheck_DT.pdf')

##Creating png file
graph.write_png('FraudCheck_DT.png')
```

Out[65]: True