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 [152…
```
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,matthews_corrcoef
from sklearn.metrics import confusion_matrix
```

In [4]:
```
fraud_check = pd.read_csv("fraud_Check.csv")
fraud_check.head()
```

Out[4]:

	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 [6]:
```
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')

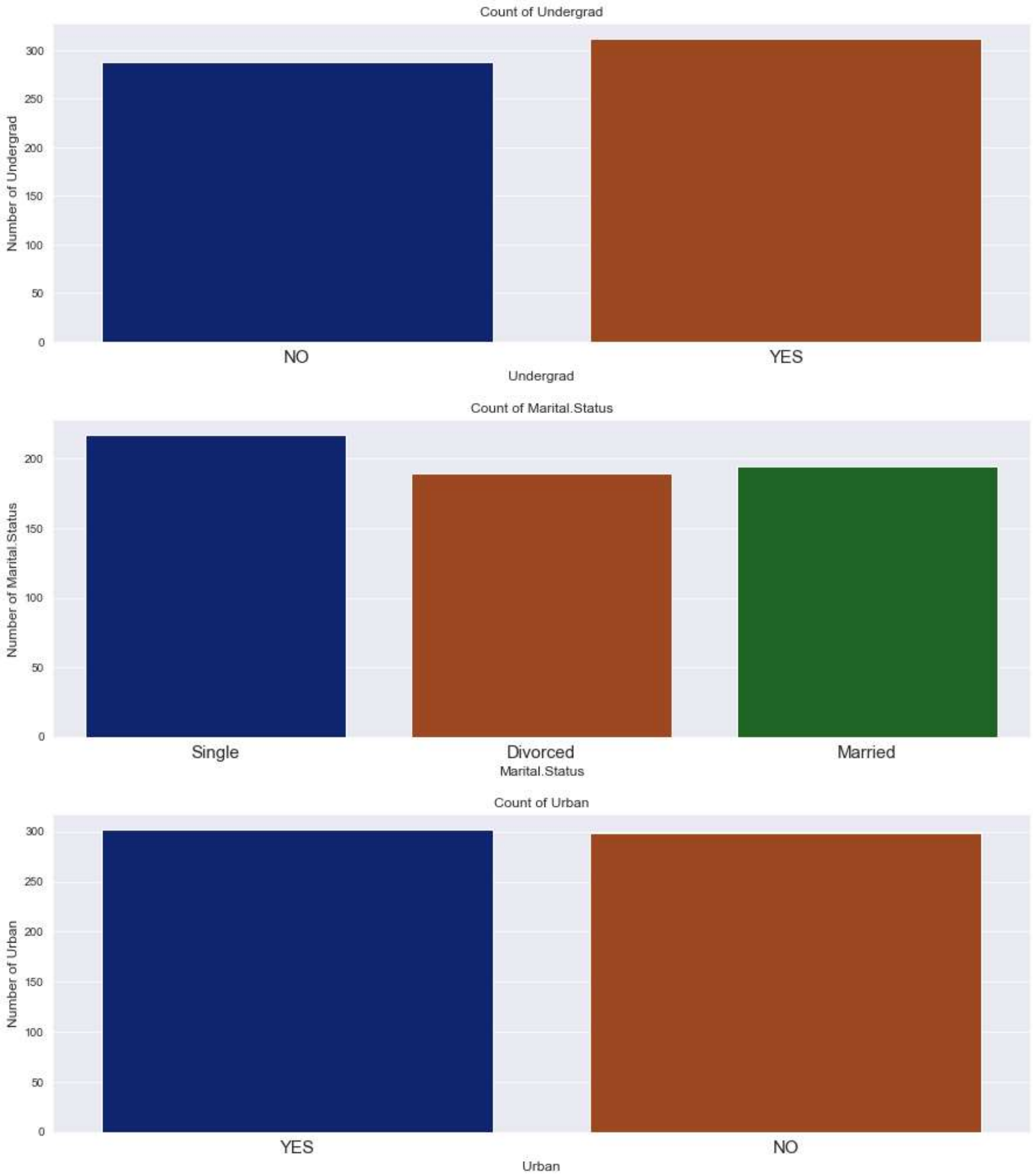
In [7]:
```
numerical_features = fraud_check.describe(include=["int64"]).columns
numerical_features
```

Out[7]: Index(['Taxable.Income', 'City.Population', 'Work.Experience'], dtype='object')

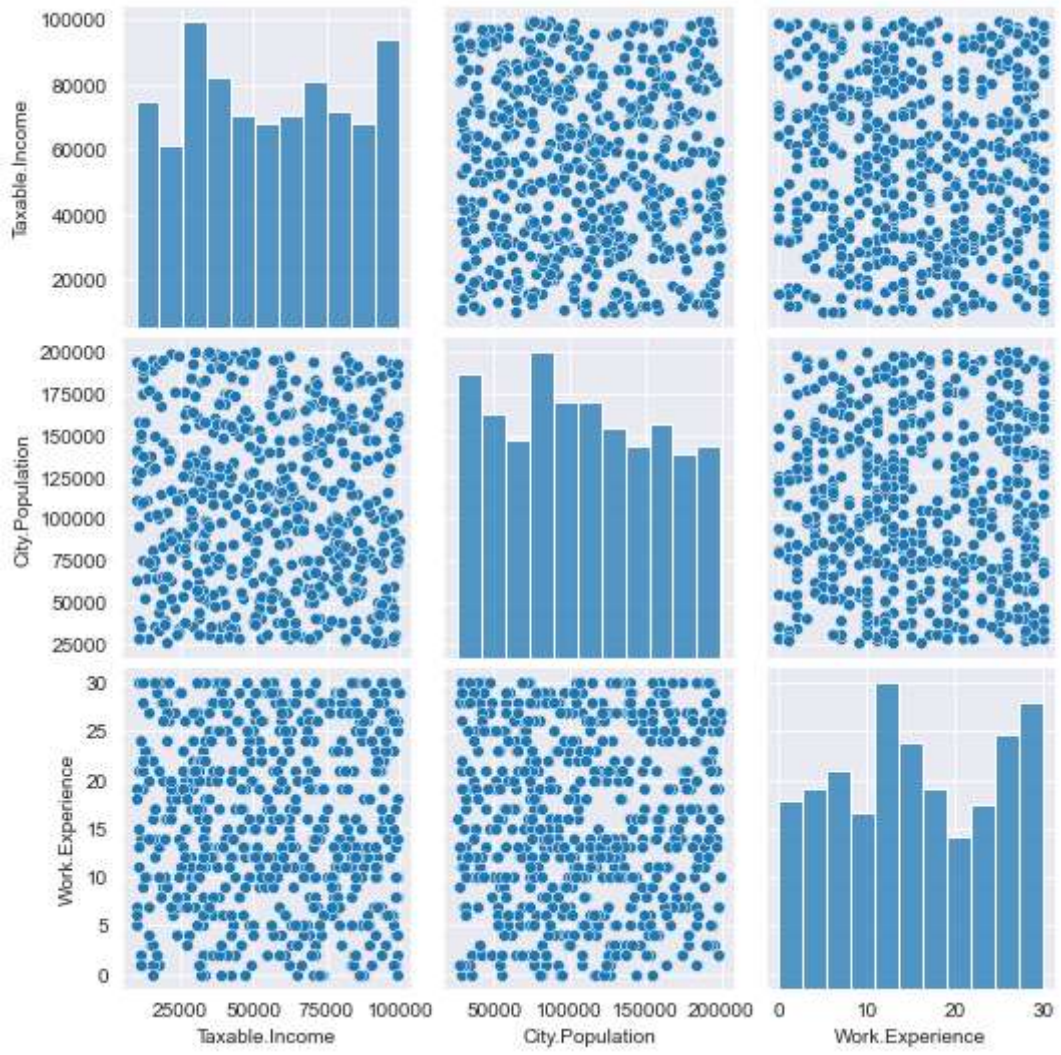
In [10]:
```
print(categorical_features)



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')
```

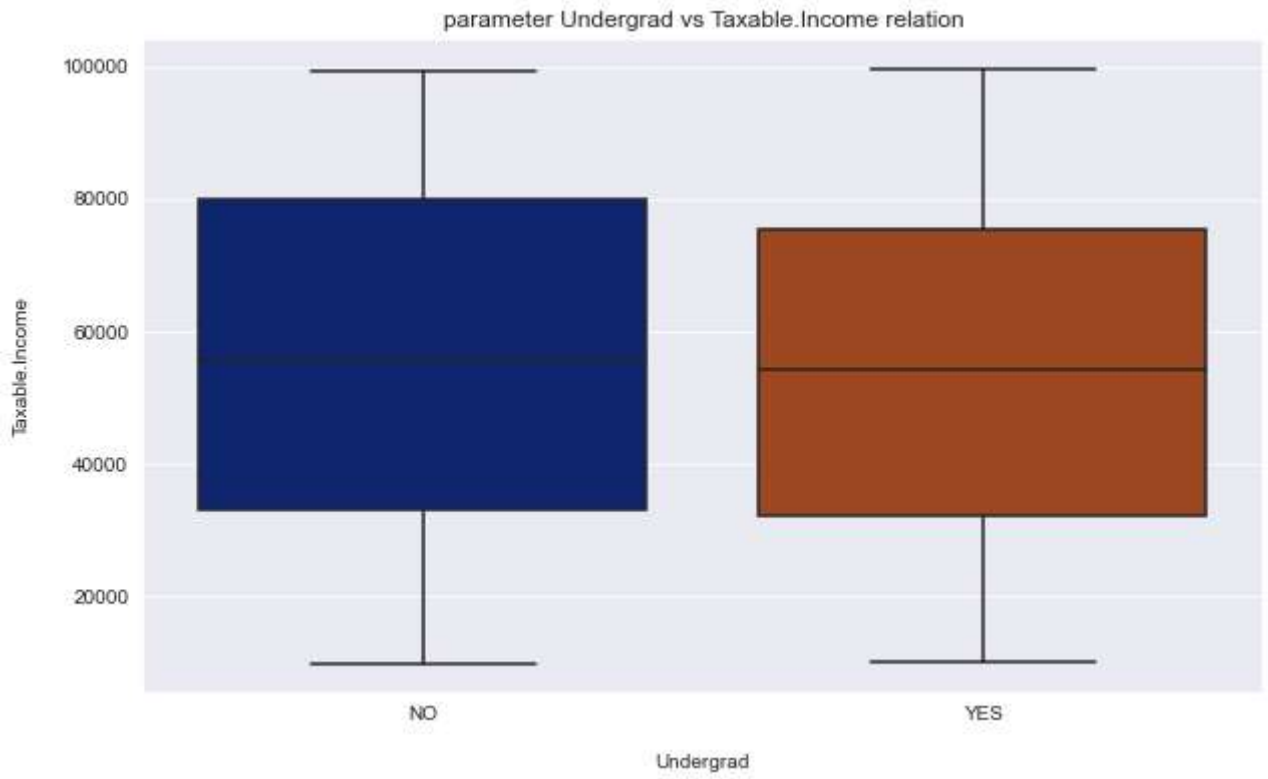


In [12]:
```
sns.set_style('darkgrid')
sns.pairplot(fraud_check[numerical_features])
plt.show()
```

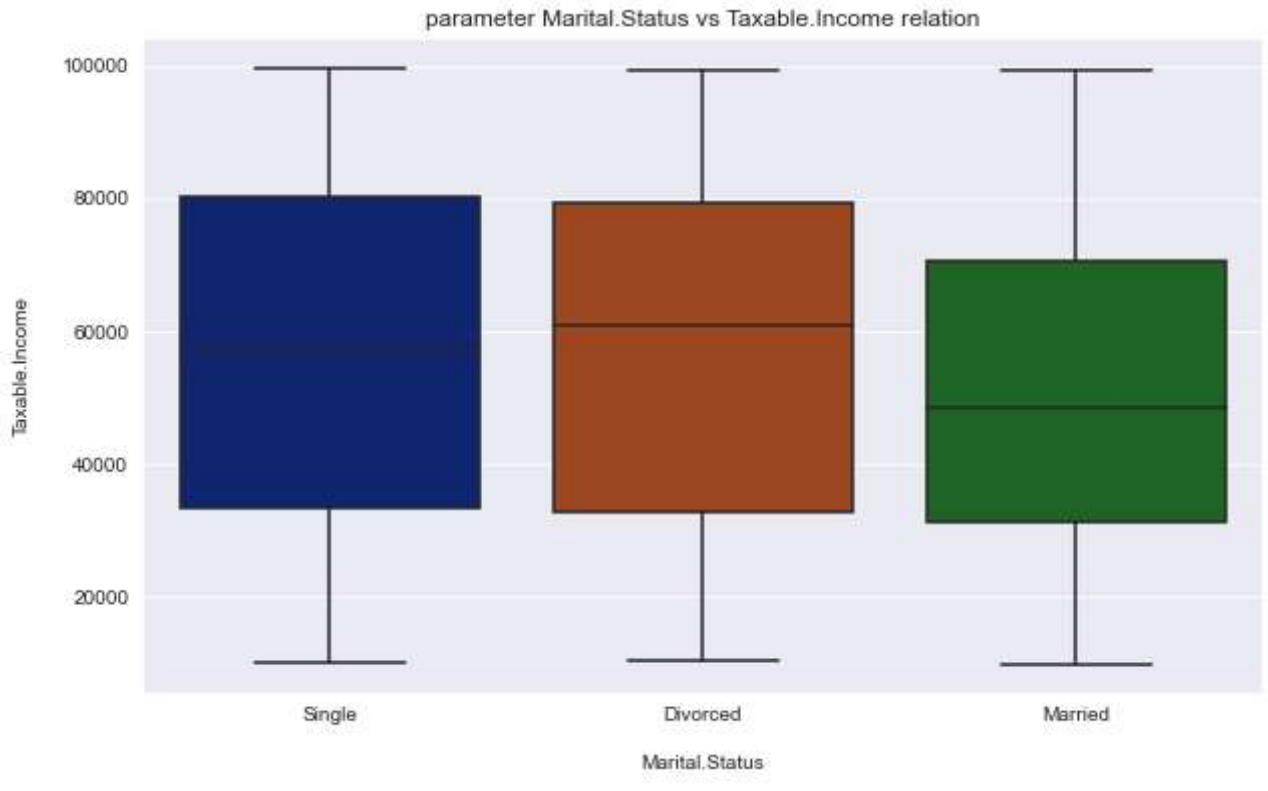


In [27]:
```
def boxplot(x_param, y_param):
    plt.figure(figsize=(10,6))
    sns.boxplot(x=x_param, data=fraud_check,y=y_param, palette = "dark")
    plt.xlabel('\n'+ x_param)
    plt.ylabel(y_param + '\n')
    plt.title("parameter " + x_param + " vs " + y_param + " relation")
    plt.show()
```

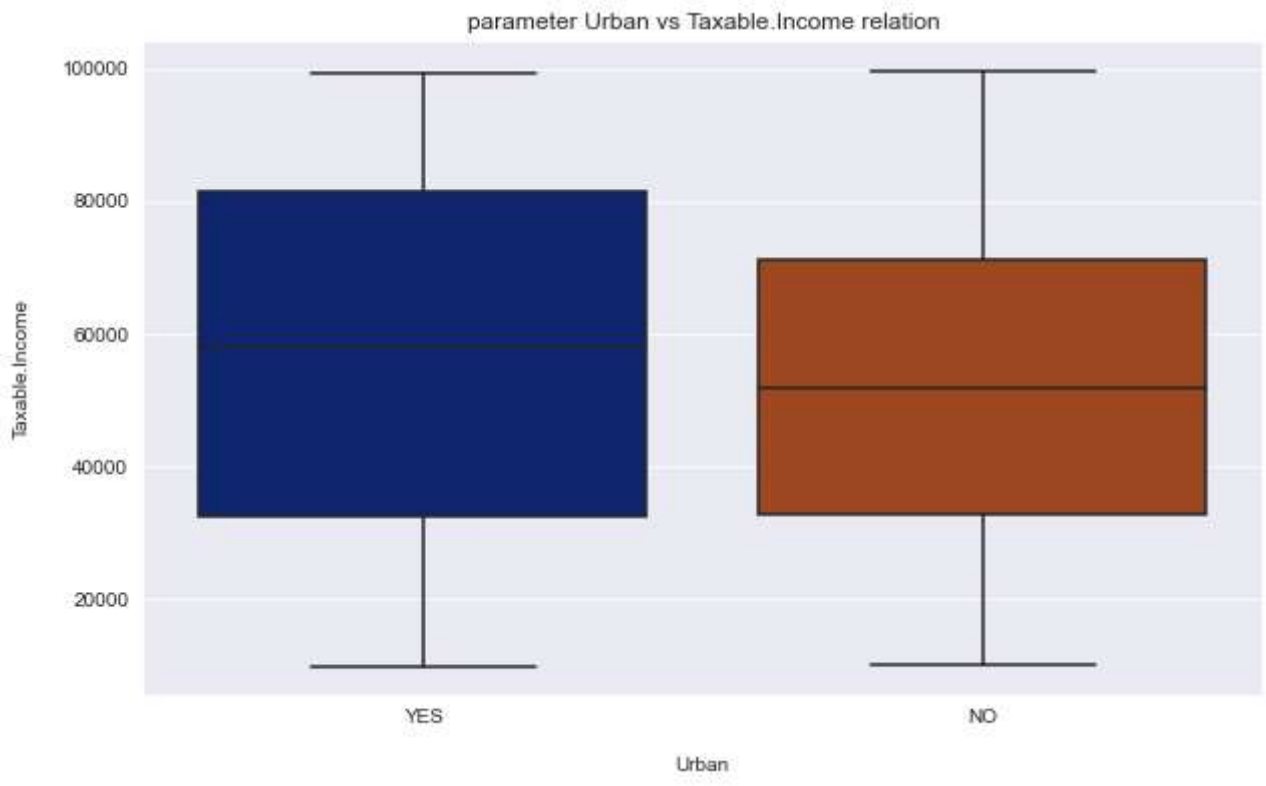
In [28]:
```
boxplot('Undergrad','Taxable.Income')
```



In [29]:
```
boxplot('Marital.Status', 'Taxable.Income')
```

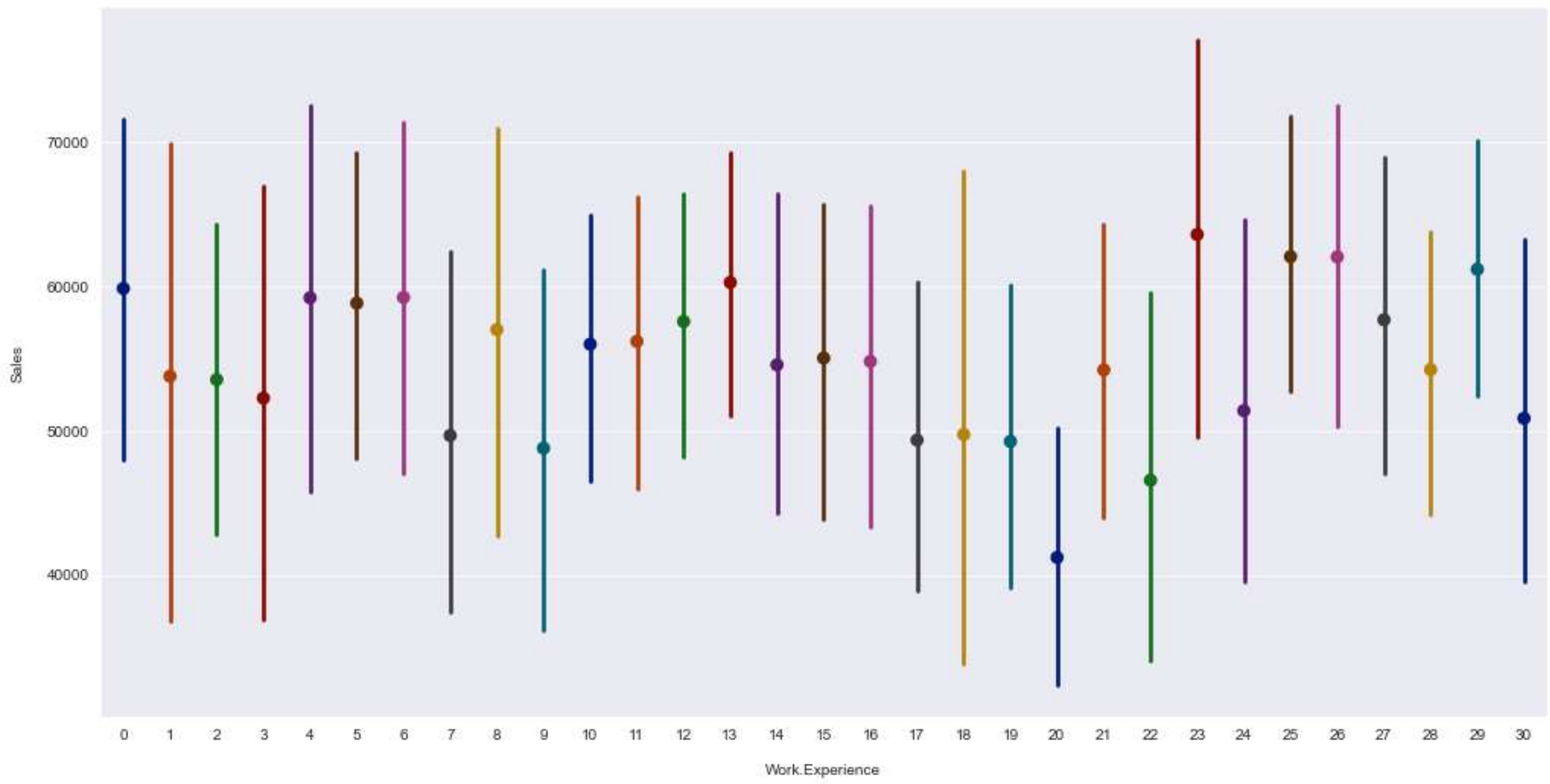


In [30]:
```
boxplot('Urban','Taxable.Income')
```



In [31]:
```
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 [32]:
```
factorplot("Work.Experience")
```



In [34]:
```
fraud_check["Taxable.Income"].min()
```

Out[34]:
```
10003
```

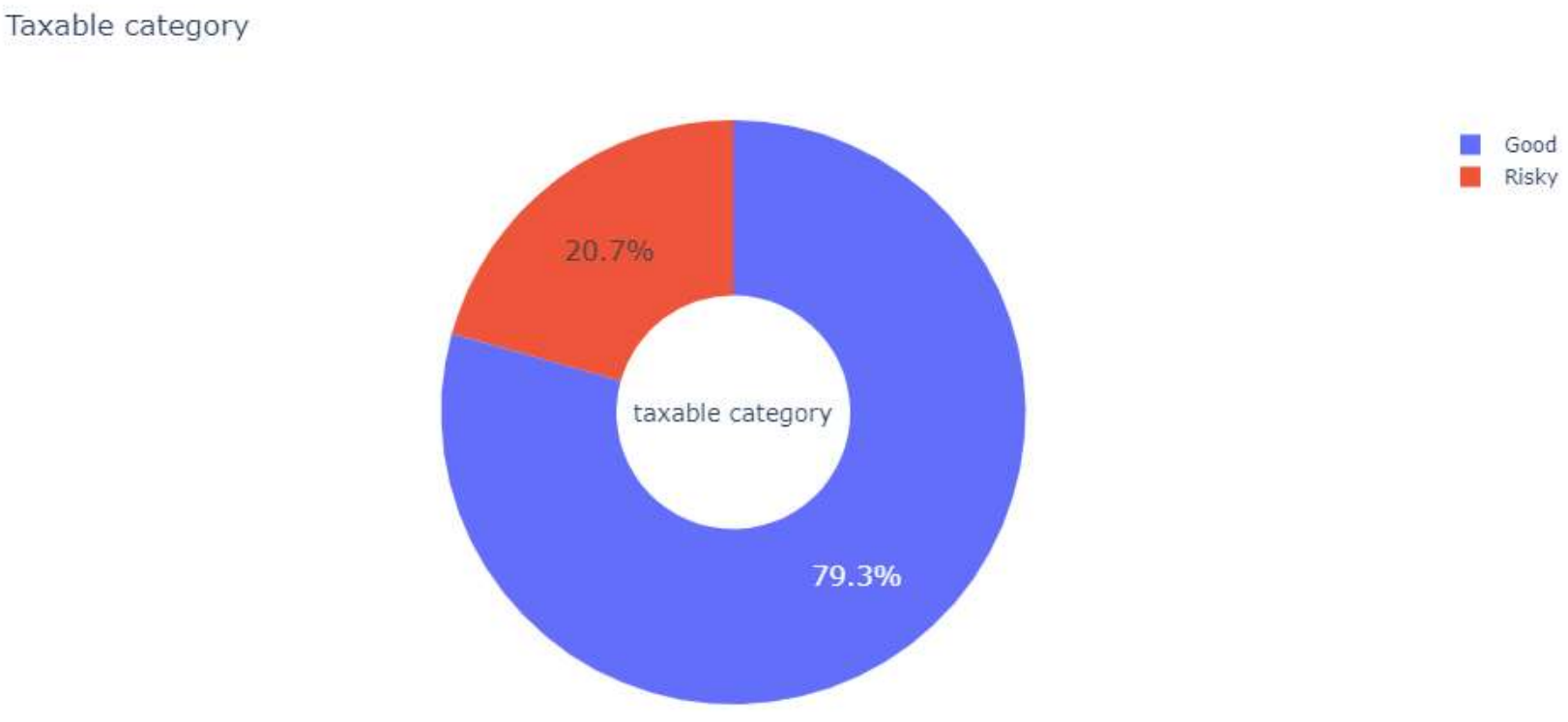
In [35]:
```
# Converting taxable_income <= 30000 as "Risky" and others are "Good"
fraud_check['taxable_category'] = pd.cut(x = fraud_check['Taxable.Income'], bins = [10002,30000,99620], labels = ['Risky'
fraud_check.head()
```

Out[35]:

	Undergrad	Marital.Status	Taxable.Income	City.Population	Work.Experience	Urban	taxable_category
0	NO	Single	68833	50047	10	YES	Good
1	YES	Divorced	33700	134075	18	YES	Good
2	NO	Married	36925	160205	30	YES	Good
3	YES	Single	50190	193264	15	YES	Good
4	NO	Married	81002	27533	28	NO	Good

In [ ]:
```
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_category"))

# 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 [26]:

```
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 [42]:

```
fraud_check["taxable_category"][fraud_check["taxable_category"] == 'Risky'].groupby(by = fraud_check.Undergrad).count()
```

Out[42]:

```
Undergrad
NO     58
YES    66
Name: taxable_category, dtype: int64
```

In [43]:

```
fraud_check["taxable_category"][fraud_check["taxable_category"] == 'Good'].groupby(by = fraud_check.Undergrad).count()
```

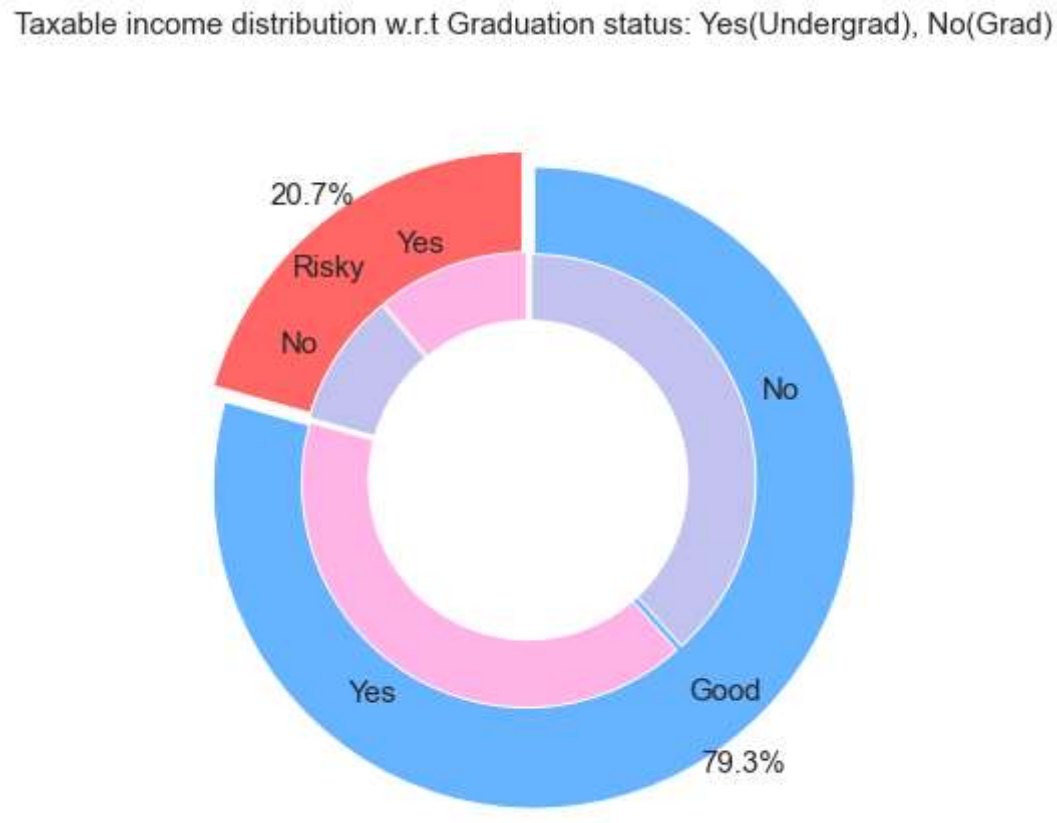
Out[43]:

```
Undergrad
NO     230
YES    246
Name: taxable_category, dtype: int64
```

In [46]:

```
plt.figure(figsize=(6, 6))
labels =["Risky", "Good"]
values = [fraud_check["taxable_category"][fraud_check["taxable_category"] == 'Risky'].groupby(by = fraud_check.Undergrad)
          fraud_check["taxable_category"][fraud_check["taxable_category"] == 'Good'].groupby(by = fraud_check.Undergrad).c
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, startangle=90,frame=Tr
plt.pie(sizes_gender,labels=labels_gender,colors=colors_gender,startangle=90, explode=explode_gender,radius=7, textprops
```

```
#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=1.1)

# show plot

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



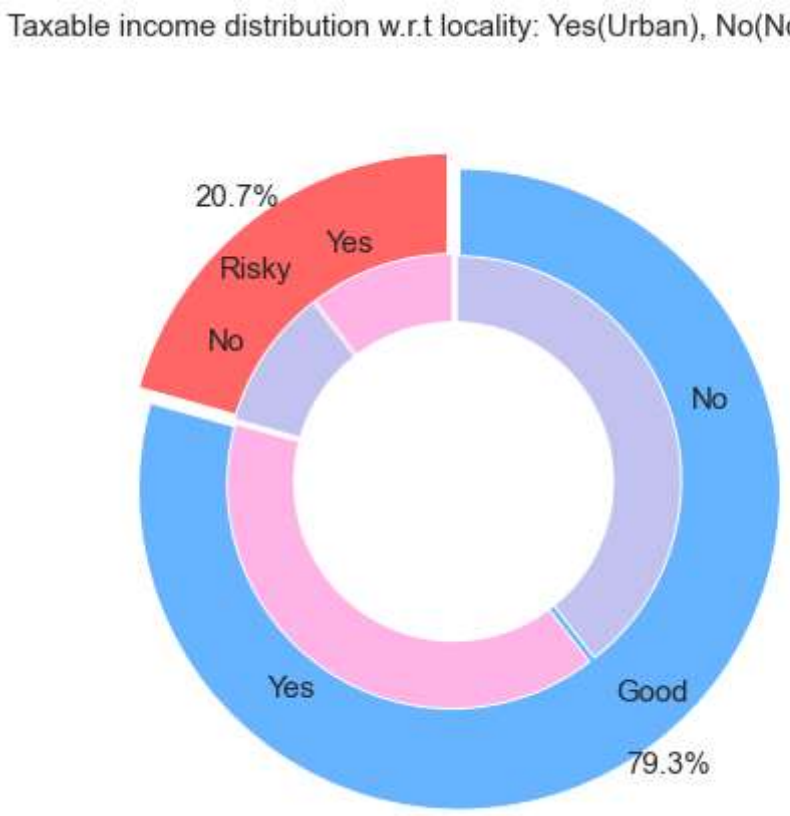
In [54]:
```
fraud_check["taxable_category"][fraud_check["taxable_category"] == 'Risky'].groupby(by = fraud_check.Urban).count()
```

Out[54]:
```
Urban
NO     61
YES    63
Name: taxable_category, dtype: int64
```

In [55]:
```
fraud_check["taxable_category"][fraud_check["taxable_category"] == 'Good'].groupby(by = fraud_check.Urban).count()
```

Out[55]:
```
Urban
NO     237
YES    239
Name: taxable_category, dtype: int64
```

In [56]:
```
plt.figure(figsize=(6, 6))
labels =["Risky", "Good"]
values = [fraud_check["taxable_category"][fraud_check["taxable_category"] == 'Risky'].groupby(by = fraud_check.Urban).cou
          fraud_check["taxable_category"][fraud_check["taxable_category"] == 'Good'].groupby(by = fraud_check.Urban).count
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, startangle=90,frame=Tr
plt.pie(sizes_gender,labels=labels_gender,colors=colors_gender,startangle=90, explode=explode_gender,radius=7, textprops
#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()
```



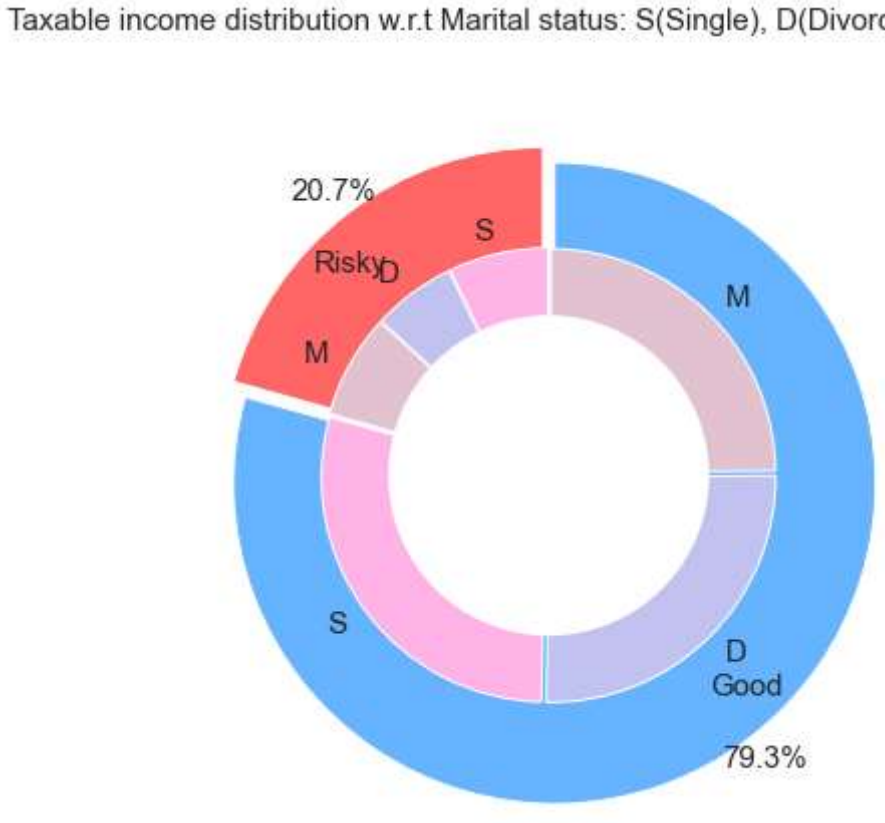
In [48]:
```
fraud_check["taxable_category"][fraud_check["taxable_category"] == 'Risky'].groupby(by = fraud_check["Marital.Status"]).c
```

Out[48]:
```
Marital.Status
Divorced    36
Married     45
Single      43
Name: taxable_category, dtype: int64
```

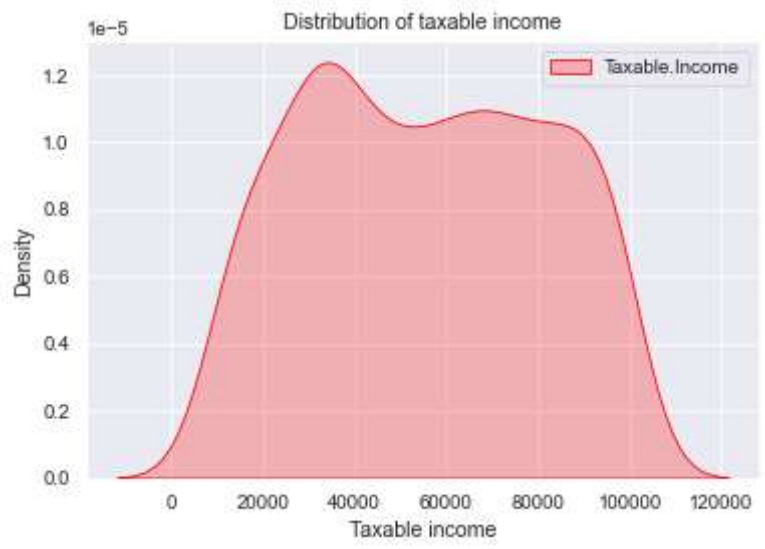
In [49]:
```
fraud_check["taxable_category"][fraud_check["taxable_category"] == 'Good'].groupby(by = fraud_check["Marital.Status"]).cc
```

Out[49]:
```
Marital.Status
Divorced    153
Married     149
Single      174
Name: taxable_category, dtype: int64
```

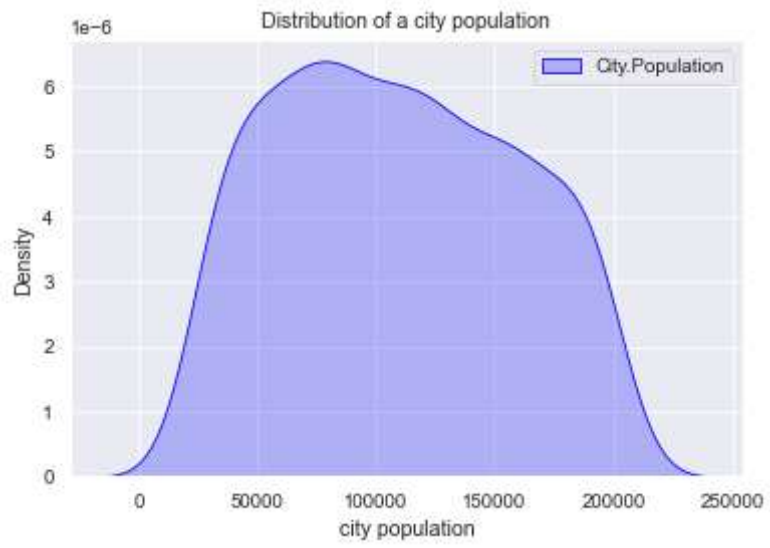
In [53]:
```
plt.figure(figsize=(6, 6))
labels =["Risky", "Good"]
values = [fraud_check["taxable_category"][fraud_check["taxable_category"] == 'Risky'].groupby(by = fraud_check["Marital.S
         fraud_check["taxable_category"][fraud_check["taxable_category"] == 'Good'].groupby(by = fraud_check["Marital.Sta
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, startangle=90,frame=Tr
plt.pie(sizes_gender,labels=labels_gender,colors=colors_gender,startangle=90, explode=explode_gender,radius=7, textprops
#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=15, y=1.1)

# show plot

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



In [65]:
```
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 [68]:
```
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 [107…
```
"""# 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 [111…
```
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[111…

	Taxable.Income	City.Population	Work.Experience	Undergrad_NO	Undergrad_YES	Marital.Status_Divorced	Marital.Status_Married	Marital.St
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	

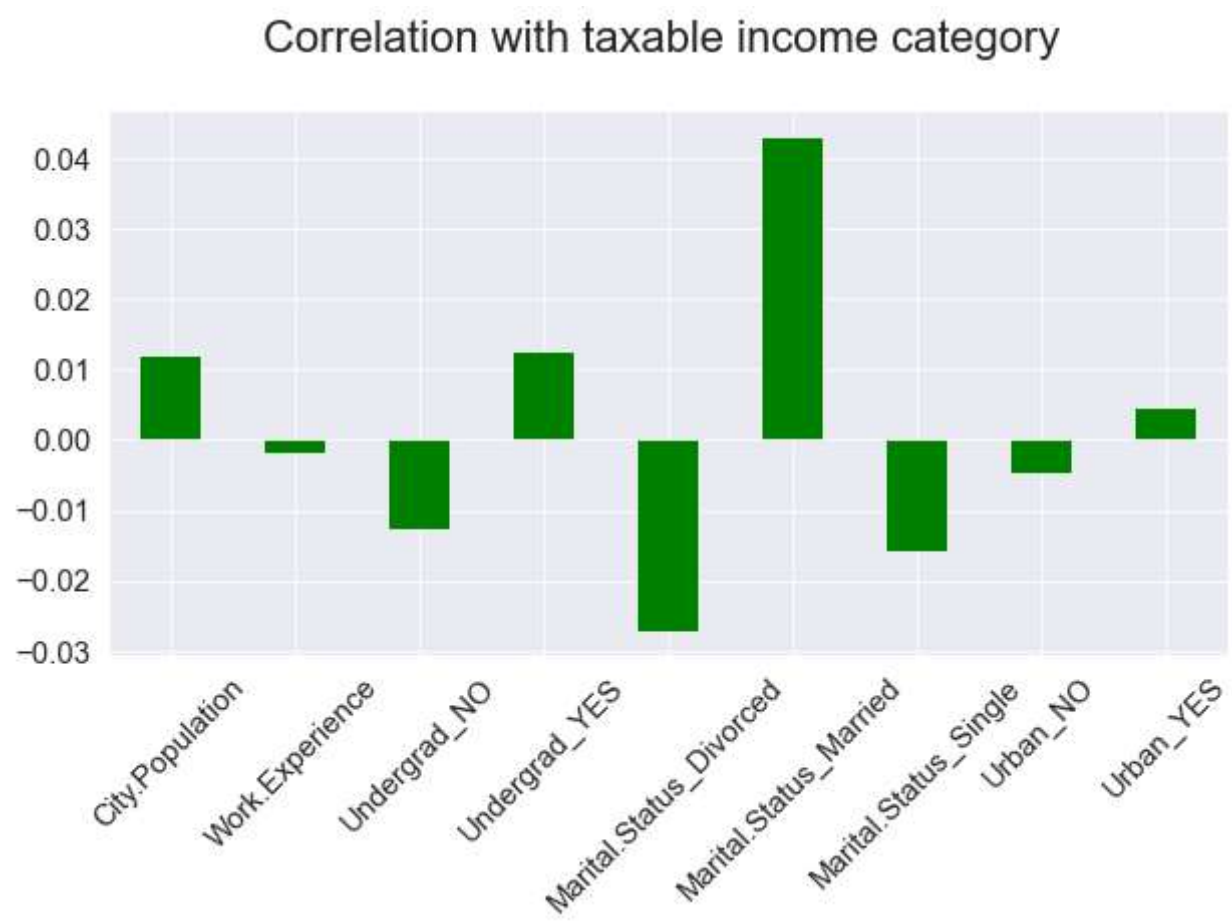
In [112…
```
fraudCheck_data = data_copy.drop('Taxable.Income', axis = 1)
fraudCheck_data.head()
```

Out[112…

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

In [113…
```
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[113… `Text(0.5, 1.0, 'Correlation with taxable income category \n')`



In [114…
```
y = fraudCheck_data['taxable_category']
X = fraudCheck_data.drop('taxable_category', axis = 1)
```

In [117…
```
def norm_func(i):
    x= (i-i.min())/(i.max()-i.min())
    return (x)

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

Out[117…

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

In [119…
```
X_train, X_test, y_train, y_test = train_test_split(X_, y, test_size=0.33, random_state=42)
```

In [120…
```
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 [136…
```
#base model
score_array = []
for each in range(1,300):
    rf_loop = RandomForestClassifier(n_estimators = each, random_state = 1)
    rf_loop.fit(X_train,y_train)
    score_array.append(rf_loop.score(X_test,y_test))
```

In [137…
```
for i,j in enumerate(score_array):
    print(i+1,":",j)
```

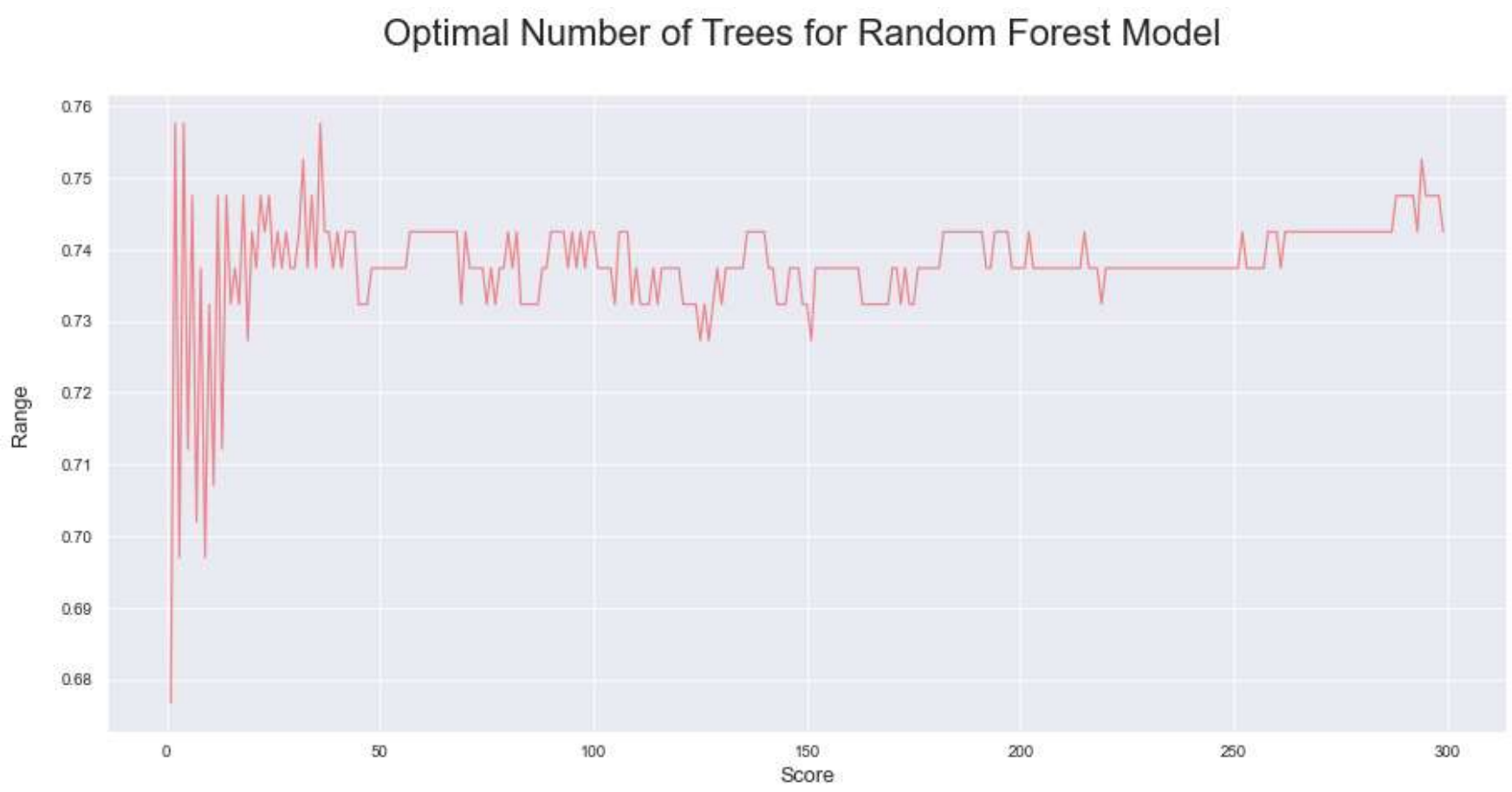
```
1 : 0.6767676767676768
2 : 0.7575757575757576
3 : 0.696969696969697
4 : 0.7575757575757576
5 : 0.7121212121212122
6 : 0.7474747474747475
7 : 0.702020202020202
8 : 0.7373737373737373
9 : 0.696969696969697
10 : 0.7323232323232324
11 : 0.7070707070707071
12 : 0.7474747474747475
13 : 0.7121212121212122
14 : 0.7474747474747475
15 : 0.7323232323232324
16 : 0.7373737373737373
17 : 0.7323232323232324
18 : 0.7474747474747475
19 : 0.7272727272727273
20 : 0.7424242424242424
21 : 0.7373737373737373
22 : 0.7474747474747475
23 : 0.7424242424242424
24 : 0.7474747474747475
25 : 0.7373737373737373
26 : 0.7424242424242424
27 : 0.7373737373737373
28 : 0.7424242424242424
29 : 0.7373737373737373
30 : 0.7373737373737373
31 : 0.7424242424242424
32 : 0.7525252525252525
33 : 0.7373737373737373
34 : 0.7474747474747475
35 : 0.7373737373737373
36 : 0.7575757575757576
37 : 0.7424242424242424
38 : 0.7424242424242424
39 : 0.7373737373737373
40 : 0.7424242424242424
41 : 0.7373737373737373
42 : 0.7424242424242424
43 : 0.7424242424242424
44 : 0.7424242424242424
45 : 0.7323232323232324
46 : 0.7323232323232324
47 : 0.7323232323232324
48 : 0.7373737373737373
49 : 0.7373737373737373
50 : 0.7373737373737373
51 : 0.7373737373737373
52 : 0.7373737373737373
53 : 0.7373737373737373
54 : 0.7373737373737373
55 : 0.7373737373737373
56 : 0.7373737373737373
57 : 0.7424242424242424
58 : 0.7424242424242424
59 : 0.7424242424242424
60 : 0.7424242424242424
61 : 0.7424242424242424
62 : 0.7424242424242424
63 : 0.7424242424242424
64 : 0.7424242424242424
65 : 0.7424242424242424
66 : 0.7424242424242424
67 : 0.7424242424242424
68 : 0.7424242424242424
69 : 0.7323232323232324
70 : 0.7424242424242424
71 : 0.7373737373737373
72 : 0.7373737373737373
73 : 0.7373737373737373
74 : 0.7373737373737373
75 : 0.7323232323232324
76 : 0.7373737373737373
77 : 0.7323232323232324
78 : 0.7373737373737373
79 : 0.7373737373737373
80 : 0.7424242424242424
81 : 0.7373737373737373
82 : 0.7424242424242424
83 : 0.7323232323232324
84 : 0.7323232323232324
85 : 0.7323232323232324
86 : 0.7323232323232324
87 : 0.7323232323232324
88 : 0.7373737373737373
89 : 0.7373737373737373
```

```
90 : 0.7424242424242424
91 : 0.7424242424242424
92 : 0.7424242424242424
93 : 0.7424242424242424
94 : 0.7373737373737373
95 : 0.7424242424242424
96 : 0.7373737373737373
97 : 0.7424242424242424
98 : 0.7373737373737373
99 : 0.7424242424242424
100 : 0.7424242424242424
101 : 0.7373737373737373
102 : 0.7373737373737373
103 : 0.7373737373737373
104 : 0.7373737373737373
105 : 0.7323232323232324
106 : 0.7424242424242424
107 : 0.7424242424242424
108 : 0.7424242424242424
109 : 0.7323232323232324
110 : 0.7373737373737373
111 : 0.7323232323232324
112 : 0.7323232323232324
113 : 0.7323232323232324
114 : 0.7373737373737373
115 : 0.7323232323232324
116 : 0.7373737373737373
117 : 0.7373737373737373
118 : 0.7373737373737373
119 : 0.7373737373737373
120 : 0.7373737373737373
121 : 0.7323232323232324
122 : 0.7323232323232324
123 : 0.7323232323232324
124 : 0.7323232323232324
125 : 0.7272727272727273
126 : 0.7323232323232324
127 : 0.7272727272727273
128 : 0.7323232323232324
129 : 0.7373737373737373
130 : 0.7323232323232324
131 : 0.7373737373737373
132 : 0.7373737373737373
133 : 0.7373737373737373
134 : 0.7373737373737373
135 : 0.7373737373737373
136 : 0.7424242424242424
137 : 0.7424242424242424
138 : 0.7424242424242424
139 : 0.7424242424242424
140 : 0.7424242424242424
141 : 0.7373737373737373
142 : 0.7373737373737373
143 : 0.7323232323232324
144 : 0.7323232323232324
145 : 0.7323232323232324
146 : 0.7373737373737373
147 : 0.7373737373737373
148 : 0.7373737373737373
149 : 0.7323232323232324
150 : 0.7323232323232324
151 : 0.7272727272727273
152 : 0.7373737373737373
153 : 0.7373737373737373
154 : 0.7373737373737373
155 : 0.7373737373737373
156 : 0.7373737373737373
157 : 0.7373737373737373
158 : 0.7373737373737373
159 : 0.7373737373737373
160 : 0.7373737373737373
161 : 0.7373737373737373
162 : 0.7373737373737373
163 : 0.7323232323232324
164 : 0.7323232323232324
165 : 0.7323232323232324
166 : 0.7323232323232324
167 : 0.7323232323232324
168 : 0.7323232323232324
169 : 0.7323232323232324
170 : 0.7373737373737373
171 : 0.7373737373737373
172 : 0.7323232323232324
173 : 0.7373737373737373
174 : 0.7323232323232324
175 : 0.7323232323232324
176 : 0.7373737373737373
177 : 0.7373737373737373
178 : 0.7373737373737373
179 : 0.7373737373737373
180 : 0.7373737373737373
181 : 0.7373737373737373
182 : 0.7424242424242424
183 : 0.7424242424242424
184 : 0.7424242424242424
185 : 0.7424242424242424
186 : 0.7424242424242424
187 : 0.7424242424242424
188 : 0.7424242424242424
189 : 0.7424242424242424
190 : 0.7424242424242424
191 : 0.7424242424242424
192 : 0.7373737373737373
```

```
193 : 0.7373737373737373
194 : 0.7424242424242424
195 : 0.7424242424242424
196 : 0.7424242424242424
197 : 0.7424242424242424
198 : 0.7373737373737373
199 : 0.7373737373737373
200 : 0.7373737373737373
201 : 0.7373737373737373
202 : 0.7424242424242424
203 : 0.7373737373737373
204 : 0.7373737373737373
205 : 0.7373737373737373
206 : 0.7373737373737373
207 : 0.7373737373737373
208 : 0.7373737373737373
209 : 0.7373737373737373
210 : 0.7373737373737373
211 : 0.7373737373737373
212 : 0.7373737373737373
213 : 0.7373737373737373
214 : 0.7373737373737373
215 : 0.7424242424242424
216 : 0.7373737373737373
217 : 0.7373737373737373
218 : 0.7373737373737373
219 : 0.7323232323232324
220 : 0.7373737373737373
221 : 0.7373737373737373
222 : 0.7373737373737373
223 : 0.7373737373737373
224 : 0.7373737373737373
225 : 0.7373737373737373
226 : 0.7373737373737373
227 : 0.7373737373737373
228 : 0.7373737373737373
229 : 0.7373737373737373
230 : 0.7373737373737373
231 : 0.7373737373737373
232 : 0.7373737373737373
233 : 0.7373737373737373
234 : 0.7373737373737373
235 : 0.7373737373737373
236 : 0.7373737373737373
237 : 0.7373737373737373
238 : 0.7373737373737373
239 : 0.7373737373737373
240 : 0.7373737373737373
241 : 0.7373737373737373
242 : 0.7373737373737373
243 : 0.7373737373737373
244 : 0.7373737373737373
245 : 0.7373737373737373
246 : 0.7373737373737373
247 : 0.7373737373737373
248 : 0.7373737373737373
249 : 0.7373737373737373
250 : 0.7373737373737373
251 : 0.7373737373737373
252 : 0.7424242424242424
253 : 0.7373737373737373
254 : 0.7373737373737373
255 : 0.7373737373737373
256 : 0.7373737373737373
257 : 0.7373737373737373
258 : 0.7424242424242424
259 : 0.7424242424242424
260 : 0.7424242424242424
261 : 0.7373737373737373
262 : 0.7424242424242424
263 : 0.7424242424242424
264 : 0.7424242424242424
265 : 0.7424242424242424
266 : 0.7424242424242424
267 : 0.7424242424242424
268 : 0.7424242424242424
269 : 0.7424242424242424
270 : 0.7424242424242424
271 : 0.7424242424242424
272 : 0.7424242424242424
273 : 0.7424242424242424
274 : 0.7424242424242424
275 : 0.7424242424242424
276 : 0.7424242424242424
277 : 0.7424242424242424
278 : 0.7424242424242424
279 : 0.7424242424242424
280 : 0.7424242424242424
281 : 0.7424242424242424
282 : 0.7424242424242424
283 : 0.7424242424242424
284 : 0.7424242424242424
285 : 0.7424242424242424
286 : 0.7424242424242424
287 : 0.7424242424242424
288 : 0.7474747474747475
289 : 0.7474747474747475
290 : 0.7474747474747475
291 : 0.7474747474747475
292 : 0.7474747474747475
293 : 0.7424242424242424
294 : 0.7525252525252525
295 : 0.7474747474747475
```

```
296 : 0.7474747474747475
297 : 0.7474747474747475
298 : 0.7474747474747475
299 : 0.7424242424242424
```

In [139…
```
fig = plt.figure(figsize=(15, 7))
plt.plot(range(1,300),score_array, color = '#ec838a')
plt.ylabel('Range\n',horizontalalignment="center",
fontstyle = "normal", fontsize = "large",
fontfamily = "sans-serif")
plt.xlabel('Score\n',horizontalalignment="center",
fontstyle = "normal", fontsize = "large",
fontfamily = "sans-serif")
plt.title('Optimal Number of Trees for Random Forest Model \n',horizontalalignment="center", fontstyle = "normal", fontsi
#plt.legend(loc='top right', fontsize = "medium")
plt.xticks(rotation=0, horizontalalignment="center")
plt.yticks(rotation=0, horizontalalignment="right")
plt.show()
```



In [146…
```
num_trees = 35
max_features = 'auto'
model = RandomForestClassifier(n_estimators=num_trees, max_features=max_features)
model.fit(X_train, y_train)
```

Out[146… `RandomForestClassifier(n_estimators=35)`

In [147…
```
y_pred = model.predict(X_test)
acc= accuracy_score(y_test,y_pred)
print("The accuracy is  {}".format(acc))
```

```
The accuracy is  0.7525252525252525
```

In [148…
```
kfold = KFold(n_splits=10, random_state=42)

results = cross_val_score(model, x_train, y_train, cv=kfold)
print(results.mean())
```

```
0.7363414634146341
```

In [162…
```
kfold2 = KFold(n_splits=15, random_state=42)
```

In [159…
```
#Random Forest
rf_2 = {"max_depth": [8,6,12,24,26],
            "max_features": [8,10,12],
            "min_samples_split": [3,5],
            "min_samples_leaf": [4,6],
            "bootstrap": [True],
            "n_estimators" :[30,35,40],
            "criterion": ["gini","entropy"],
            "max_leaf_nodes":[6,8,12,26,28],
            "min_impurity_decrease":[0.0],
            "min_weight_fraction_leaf":[0.0]}

tuning_2 = GridSearchCV(estimator = RandomForestClassifier(), param_grid = rf_2, scoring = 'accuracy', n_jobs = 6, cv = k

tuning_2.fit(X_train,np.ravel(y_train))

rf_best_2 = tuning_2.best_estimator_

tuning_2.best_score_
```

Out[159… `0.7986585365853658`

In [160…
```
tuning_2.best_params_
```

Out[160…
```
{'bootstrap': True,
```

```
'criterion': 'gini',
'max_depth': 12,
'max_features': 8,
'max_leaf_nodes': 28,
'min_impurity_decrease': 0.0,
'min_samples_leaf': 6,
'min_samples_split': 5,
'min_weight_fraction_leaf': 0.0,
'n_estimators': 40}
```

In [161…
```
pred_2 = rf_best_2.predict(X_test)
accuracy_test_2 = accuracy_score(y_test,pred_2)
accuracy_test_2
```

Out[161…
```
0.7929292929292929
```

In [170…
```
X_.columns
```

Out[170…
```
Index(['City.Population', 'Work.Experience', 'Undergrad_NO', 'Undergrad_YES',
       'Marital.Status_Divorced', 'Marital.Status_Married',
       'Marital.Status_Single', 'Urban_NO', 'Urban_YES'],
      dtype='object')
```

In [174…
```
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 = rf_best_2.estimators_[20]
dot_data = StringIO()
```

In [175…
```
export_graphviz(tree1,out_file = dot_data,
                feature_names =predictors,
                class_names = target, filled =True,
                rounded=True,impurity =False,proportion=False,precision =2)
```

In [178…
```
import os
os.environ["PATH"] += os.pathsep + 'C:\Program Files\Graphviz/bin/'

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



##Creating pdf file
graph.write_pdf('FraudCheck_RF.pdf')

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

Out[178…
```
True
```

In [179…
```
Image(graph.create_png())
```

Out[179…