In [1]:
```
!pip install plotly
```

```
Requirement already satisfied: plotly in c:\programdata\anaconda3\lib\site-pack
ages (5.5.0)
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-package
s (from plotly) (1.15.0)
Requirement already satisfied: tenacity>=6.2.0 in c:\programdata\anaconda3\lib
\site-packages (from plotly) (8.0.1)
```

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import  DecisionTreeClassifier
from sklearn import tree
from sklearn.metrics import classification_report
from sklearn import preprocessing
from sklearn.metrics import confusion_matrix
from scipy.special import boxcox1p
import warnings
warnings.filterwarnings("ignore")
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
```

In [4]:
```
fraud = pd.read_csv("Fraud_check .csv")
fraud
```

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
...	...	...	...	...	...	...
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

In [5]:
```
fraud.head()
```

Out[5]:

	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.T
```

Out[6]:

	0	1	2	3	4	5	6	7	8	
Undergrad	NO	YES	NO	YES	NO	NO	NO	YES	NO	
Marital.Status	Single	Divorced	Married	Single	Married	Divorced	Divorced	Single	Single	Di
Taxable.Income	68833	33700	36925	50190	81002	33329	83357	62774	83519	
City.Population	50047	134075	160205	193264	27533	116382	80890	131253	102481	1
Work.Experience	10	18	30	15	28	0	8	3	12	
Urban	YES	YES	YES	YES	NO	NO	YES	YES	YES	

6 rows × 600 columns

In [7]:
```
fraud.describe()
```

Out[7]:

	Taxable.Income	City.Population	Work.Experience
count	600.000000	600.000000	600.000000
mean	55208.375000	108747.368333	15.558333
std	26204.827597	49850.075134	8.842147
min	10003.000000	25779.000000	0.000000
25%	32871.500000	66966.750000	8.000000
50%	55074.500000	106493.500000	15.000000
75%	78611.750000	150114.250000	24.000000
max	99619.000000	199778.000000	30.000000

In [8]:
```
fraud.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 [9]:
```
fraud.shape
```

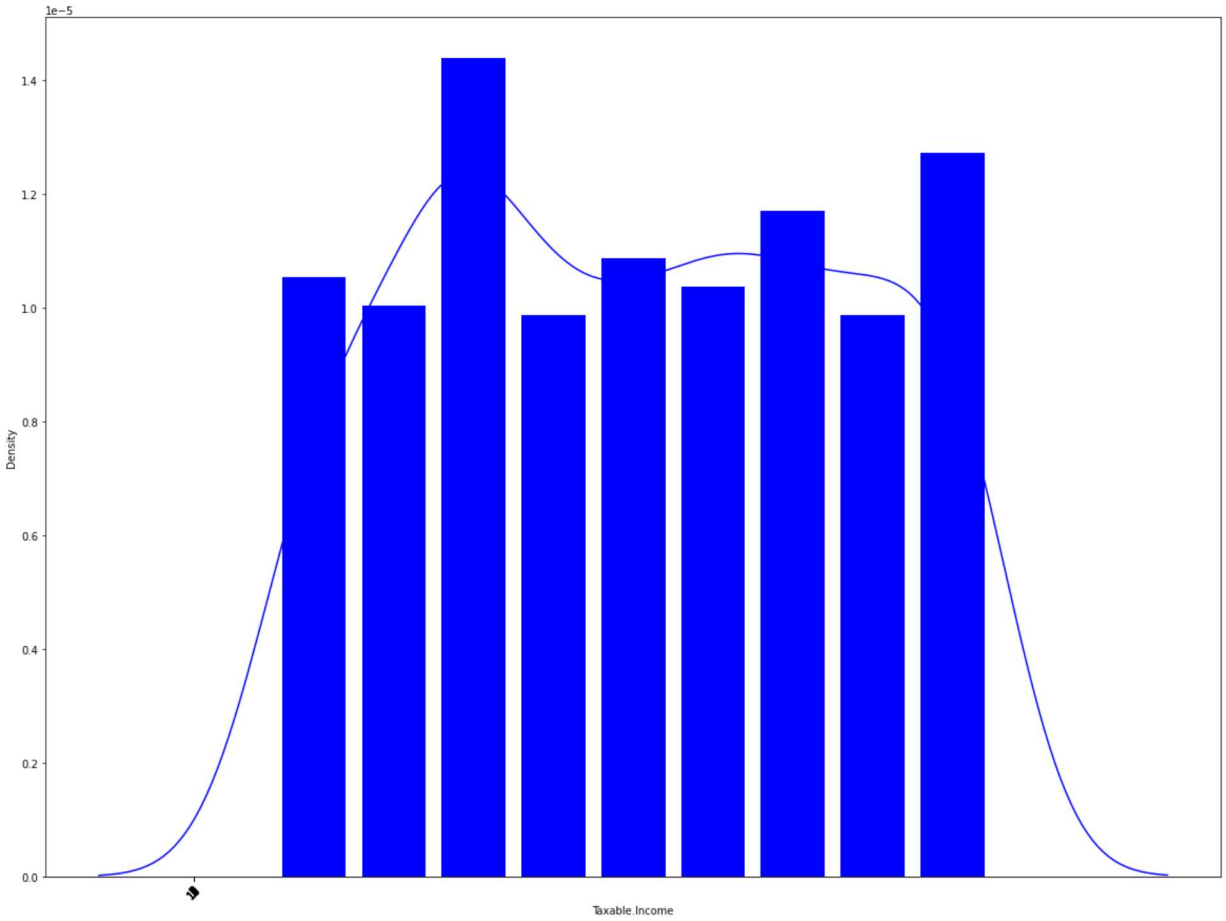
Out[9]:
```
(600, 6)
```

In [10]:
```
import seaborn as sns
plt.figure(figsize=(10,5))
sns.heatmap(fraud.corr(),annot=True)
```

Out[10]:
```
<AxesSubplot:>
```

In [11]:
```
def distplot(param):
    plt.figure(figsize=(20,15))
    sns.distplot(fraud[param], color = "blue", hist_kws={"rwidth":0.80, 'alpha':1
    plt.xticks(np.arange(0,20,1),rotation=45)
    plt.show()
```

In [12]:
```
distplot("Taxable.Income")
```



Changing the categorical variables into dummies.

In [13]:
```
fraud1 = pd.get_dummies(fraud)
```

Converting the Target variable i.e. Sales into Categorical

In [14]:
```
fraud1['Category'] = pd.cut(fraud1['Taxable.Income'],
                            bins=[0,30000, np.inf],
                            labels=['Risky','Good'],
                            include_lowest=True)
fraud1
```

Out[14]:

	Taxable.Income	City.Population	Work.Experience	Undergrad_NO	Undergrad_YES	Marital.Stat
0	68833	50047	10	1	0	
1	33700	134075	18	0	1	
2	36925	160205	30	1	0	
3	50190	193264	15	0	1	
4	81002	27533	28	1	0	
...	...	...	...	...	...	
595	76340	39492	7	0	1	
596	69967	55369	2	0	1	
597	47334	154058	0	1	0	
598	98592	180083	17	0	1	
599	96519	158137	16	1	0	

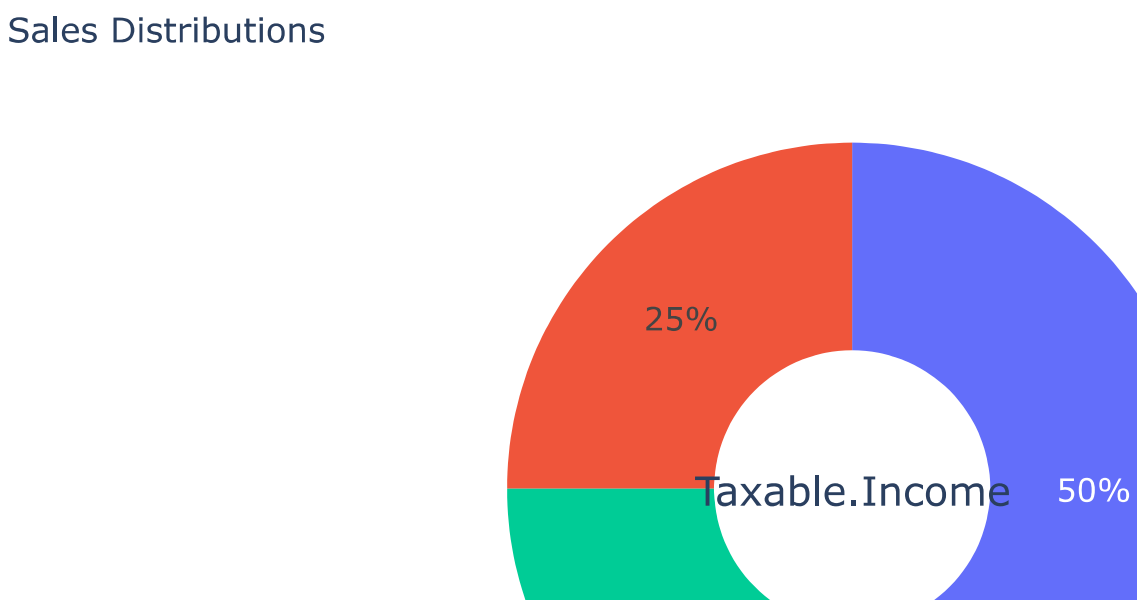
600 rows × 11 columns

In [15]:
```
from plotly.subplots import make_subplots
import plotly.graph_objects as go
type_ = ["Medium", "Low", "High"]
fig = make_subplots(rows=1, cols=1)

fig.add_trace(go.Pie(labels=type_, values=fraud['Taxable.Income'].value_counts(),

# 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="Sales Distributions",
    # Add annotations in the center of the donut pies.
    annotations=[dict(text='Taxable.Income', x=0.5, y=0.5, font_size=20, showarrc
fig.show()
```



Random Forest

In [16]:
```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
```

In [17]:
```
array = fraud1.values
X = array[:,1:10]
Y = array[:,10]
```

In [18]:
```
num_trees = 100
max_features = 3
kfold = KFold(n_splits=10, random_state=7, shuffle = True)
model = RandomForestClassifier(n_estimators=num_trees, max_features=max_features)
results = cross_val_score(model, X, Y, cv=kfold)
print(round(results.mean()*100,2))
```

```
73.33
```

Ensemble techniques

Bagging

In [19]:
```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
```

In [20]:
```
seed = 7

cart = DecisionTreeClassifier()
model1 = BaggingClassifier(base_estimator=cart, n_estimators=num_trees, random_st
results1 = cross_val_score(model1, X, Y, cv=kfold)
print(round(results.mean()*100,2))
```

```
73.33
```

Boosting

AdaBoost Classification

In [21]:
```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import AdaBoostClassifier
```

In [22]:
```
model2 = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
results2 = cross_val_score(model2, X, Y, cv=kfold)
print(round(results.mean()*100,2))
```

```
73.33
```

Stacking

Stacking Ensemble for Classification

In [24]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier
```

Iteration = 1

In [25]:
```
estimators = []
model3 = LogisticRegression(max_iter=500)
estimators.append(('logistic', model3))
model4 = DecisionTreeClassifier()
estimators.append(('cart', model4))
model5 = SVC()
estimators.append(('svm', model5))
model6 = BaggingClassifier(base_estimator=cart, n_estimators=num_trees, random_st
estimators.append(('bagging', model6))
model7 = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
estimators.append(('boosting', model7))

# create the ensemble modelIter
ensemble = VotingClassifier(estimators)
results3 = cross_val_score(ensemble, X, Y, cv=kfold)
print(results3.mean()*100)
```

```
78.83333333333333
```

Iteration = 2

In [26]:
```
estimators = []
model8 = LogisticRegression(max_iter=500)
estimators.append(('logistic', model8))
model9 = DecisionTreeClassifier()
estimators.append(('cart', model9))
model10 = BaggingClassifier(base_estimator=cart, n_estimators=num_trees, random_s
estimators.append(('bagging', model10))
model11 = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
estimators.append(('boosting', model11))

# create the ensemble model
ensemble = VotingClassifier(estimators)
results4 = cross_val_score(ensemble, X, Y, cv=kfold)
print(results4.mean()*100)
```

```
78.83333333333333
```

Iteration = 3

In [27]:
```
estimators = []
model12 = LogisticRegression(max_iter=500)
estimators.append(('logistic', model12))
model13 = DecisionTreeClassifier()
estimators.append(('cart', model13))
model14 = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
estimators.append(('boosting', model14))

# create the ensemble model
ensemble = VotingClassifier(estimators)
results5 = cross_val_score(ensemble, X, Y, cv=kfold)
print(results5.mean()*100)
```

```
78.16666666666666
```

Iteration = 4

In [28]:
```
estimators = []
model15 = DecisionTreeClassifier()
estimators.append(('cart', model15))
model16 = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
estimators.append(('boosting', model16))

# create the ensemble model
ensemble = VotingClassifier(estimators)
results6 = cross_val_score(ensemble, X, Y, cv=kfold)
print(results6.mean()*100)
```

```
78.33333333333333
```

Iteration = 5

In [29]:
```
estimators = []
model15 = DecisionTreeClassifier()
estimators.append(('logistic', model15))
model16 = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
estimators.append(('boosting', model16))

# create the ensemble model
ensemble = VotingClassifier(estimators)
results6 = cross_val_score(ensemble, X, Y, cv=kfold)
print(results6.mean()*100)
```

```
78.33333333333333
```

In [ ]: