Assignment-15- Random-Forest-01(Company_Data)

Quick info. about Random Forest.

Info. :-

A random forest is a supervised machine learning algorithm that is constructed from decision tree algorithms. This algorithm is applied in various industries such as banking and e-commerce to predict behavior and outcomes. A random forest is a machine learning technique that's used to solve regression and classification problems. It utilizes ensemble learning, which is a technique that combines many classifiers to provide solutions to complex problems. A random forest algorithm consists of many decision trees. The 'forest' generated by the random forest algorithm is trained through bagging or bootstrap aggregating. Bagging is an ensemble meta-algorithm that improves the accuracy of machine learning algorithms. The (random forest) algorithm establishes the outcome based on the predictions of the decision trees. It predicts by taking the average or mean of the output from various trees. Increasing the number of trees increases the precision of the outcome.

Features of a Random Forest Algorithm

It's more accurate than the decision tree algorithm. It provides an effective way of handling missing data. It can produce a reasonable prediction without hyper-parameter tuning. It solves the issue of overfitting in decision trees. In every random forest tree, a subset of features is selected randomly at the node's splitting point.

Applications of random forest

Banking Health care Stock market E-commerce

Advantages of random forest

It can perform both regression and classification tasks. A random forest produces good predictions that can be understood easily. It can handle large datasets efficiently. The random forest algorithm provides a higher level of accuracy in predicting outcomes over the decision tree algorithm.

Disadvantages of random forest

When using a random forest, more resources are required for computation. It consumes more time compared to a decision tree algorithm.

In [1]:

```
# important libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
```

In [2]:

```
# Load dataset
company=pd.read_csv("C:/Users/LENOVO/Documents/Custom Office Templates/Company_Data.csv")
company.head()
```

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

In [3]:

```
company.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Sales        400 non-null    float64
 1   CompPrice    400 non-null    int64
 2   Income       400 non-null    int64
 3   Advertising  400 non-null    int64
 4   Population   400 non-null    int64
 5   Price        400 non-null    int64
 6   ShelveLoc    400 non-null    object
 7   Age          400 non-null    int64
 8   Education    400 non-null    int64
 9   Urban        400 non-null    object
 10  US           400 non-null    object
dtypes: float64(1), int64(7), object(3)
memory usage: 34.5+ KB
```

In [4]:

```
company.describe()
```

Out[4]:

	Sales	CompPrice	Income	Advertising	Population	Price	Age	E
count	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	40
mean	7.496325	124.975000	68.657500	6.635000	264.840000	115.795000	53.322500	1
std	2.824115	15.334512	27.986037	6.650364	147.376436	23.676664	16.200297	
min	0.000000	77.000000	21.000000	0.000000	10.000000	24.000000	25.000000	1
25%	5.390000	115.000000	42.750000	0.000000	139.000000	100.000000	39.750000	1
50%	7.490000	125.000000	69.000000	5.000000	272.000000	117.000000	54.500000	1
75%	9.320000	135.000000	91.000000	12.000000	398.500000	131.000000	66.000000	1
max	16.270000	175.000000	120.000000	29.000000	509.000000	191.000000	80.000000	1

In [5]:

```
# Duplicates
company[company.duplicated()].shape
```

Out[5]:

```
(0, 11)
```

In [6]:

```
company.isnull().sum()
```

Out[6]:

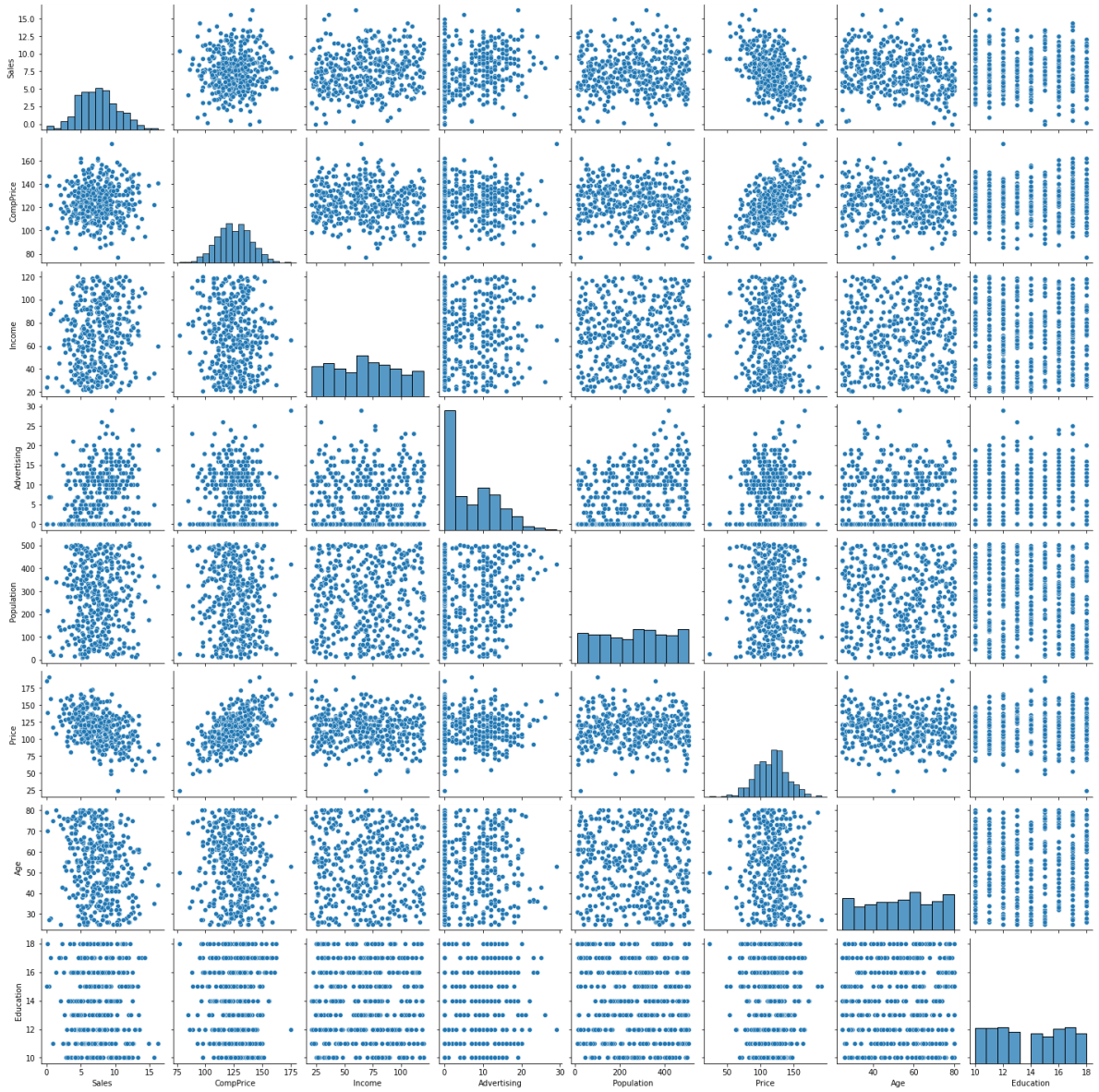
```
Sales          0
CompPrice      0
Income         0
Advertising    0
Population     0
Price          0
ShelveLoc      0
Age            0
Education      0
Urban          0
US             0
dtype: int64
```

In [7]:

```
# Scatter plot and Correaltion Analysis
sns.pairplot(company)
```

Out[7]:

```
<seaborn.axisgrid.PairGrid at 0x63da448490>
```



In [8]:

```
company.corr()
```

Out[8]:

	Sales	CompPrice	Income	Advertising	Population	Price	Age	Ed
Sales	1.000000	0.064079	0.151951	0.269507	0.050471	-0.444951	-0.231815	-0
CompPrice	0.064079	1.000000	-0.080653	-0.024199	-0.094707	0.584848	-0.100239	0
Income	0.151951	-0.080653	1.000000	0.058995	-0.007877	-0.056698	-0.004670	-0
Advertising	0.269507	-0.024199	0.058995	1.000000	0.265652	0.044537	-0.004557	-0
Population	0.050471	-0.094707	-0.007877	0.265652	1.000000	-0.012144	-0.042663	-0
Price	-0.444951	0.584848	-0.056698	0.044537	-0.012144	1.000000	-0.102177	0
Age	-0.231815	-0.100239	-0.004670	-0.004557	-0.042663	-0.102177	1.000000	0
Education	-0.051955	0.025197	-0.056855	-0.033594	-0.106378	0.011747	0.006488	1

In [9]:

```
company_df= company.copy()
```

In [10]:

```
# Converting the sales data in to Categorical
company_df.loc[company['Sales']>=8, 'Sale']='1'  # High=1 Taken based on the mean value
company_df.loc[company['Sales']<=8, 'Sale']='0'  # Low=0
```

In [11]:

```
company_df.head()
```

Out[11]:

	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

In [12]:

```
company_df= company_df.drop('Age', axis=1)
company_df= company_df.drop('Education', axis=1)
company_df= company_df.drop('Urban', axis=1)
company_df= company_df.drop('US', axis=1)
company_df=company_df.drop('Sales', axis=1)
```

In [13]:

```
company_df.head()
```

Out[13]:

	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Sale
0	138	73	11	276	120	Bad	1
1	111	48	16	260	83	Good	1
2	113	35	10	269	80	Medium	1
3	117	100	4	466	97	Medium	0
4	141	64	3	340	128	Bad	0

In [14]:

```
label_encoder= preprocessing.LabelEncoder()
company_df['ShelveLoc']=label_encoder.fit_transform(company_df['ShelveLoc'])
```

In [15]:

```
company_df.head()
```

Out[15]:

	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Sale
0	138	73	11	276	120	0	1
1	111	48	16	260	83	1	1
2	113	35	10	269	80	2	1
3	117	100	4	466	97	2	0
4	141	64	3	340	128	0	0

Note- ShelveLoc= Bad=0, Good=1, Medium=2 and Sales= High=1, Low=0

In [16]:

```
company_df=company_df.astype({'Sale':'int'})
```

Normalization Function

In [17]:

```
def norm_func(i):
    x=(i-i.min())/(i.max()-i.min())
    return(x)
```

In [18]:

```
company_df=norm_func(company_df)
```

In [19]:

```
company_df
```

Out[19]:

	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Sale
0	0.622449	0.525253	0.379310	0.533066	0.574850	0.0	1.0
1	0.346939	0.272727	0.551724	0.501002	0.353293	0.5	1.0
2	0.367347	0.141414	0.344828	0.519038	0.335329	1.0	1.0
3	0.408163	0.797980	0.137931	0.913828	0.437126	1.0	0.0
4	0.653061	0.434343	0.103448	0.661323	0.622754	0.0	0.0
...	...	...	...	...	...	...	...
395	0.622449	0.878788	0.586207	0.386774	0.622754	0.5	1.0
396	0.632653	0.020202	0.103448	0.054108	0.574850	1.0	0.0
397	0.867347	0.050505	0.413793	0.717435	0.808383	1.0	0.0
398	0.234694	0.585859	0.241379	0.549098	0.425150	0.0	0.0

In [20]:

```
company_df1=company_df.astype({"ShelveLoc":'int',"Sale":'int'})
```

In [21]:

```
company_df1.head()
```

Out[21]:

	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Sale
0	0.622449	0.525253	0.379310	0.533066	0.574850	0	1
1	0.346939	0.272727	0.551724	0.501002	0.353293	0	1
2	0.367347	0.141414	0.344828	0.519038	0.335329	1	1
3	0.408163	0.797980	0.137931	0.913828	0.437126	1	0
4	0.653061	0.434343	0.103448	0.661323	0.622754	0	0

In [22]:

```
x= company_df1.iloc[:,0:6]
y= company_df1.iloc[:,6]
```

In [23]:

```
x
```

Out[23]:

	CompPrice	Income	Advertising	Population	Price	ShelveLoc
0	0.622449	0.525253	0.379310	0.533066	0.574850	0
1	0.346939	0.272727	0.551724	0.501002	0.353293	0
2	0.367347	0.141414	0.344828	0.519038	0.335329	1
3	0.408163	0.797980	0.137931	0.913828	0.437126	1
4	0.653061	0.434343	0.103448	0.661323	0.622754	0
...	...	...	...	...	...	...
395	0.622449	0.878788	0.586207	0.386774	0.622754	0
396	0.632653	0.020202	0.103448	0.054108	0.574850	1
397	0.867347	0.050505	0.413793	0.717435	0.808383	1
398	0.234694	0.585859	0.241379	0.549098	0.425150	0
399	0.581633	0.161616	0.000000	0.034068	0.574850	0

400 rows × 6 columns

In [24]:

```
y  # Sale - High=1, Low=0
```

Out[24]:

```
0      1
1      1
2      1
3      0
4      0
      ..
395    1
396    0
397    0
398    0
399    1
Name: Sale, Length: 400, dtype: int32
```

In [25]:

```
company_df1['Sale'].unique()
```

Out[25]:

```
array([1, 0])
```

In [26]:

```
company_df1.Sale.value_counts()
```

Out[26]:

```
0    236
1    164
Name: Sale, dtype: int64
```

Splitting data into Training and Testing Dataset

In [27]:

```
from sklearn.model_selection import train_test_split

# Separate out dataset into train test split

# using this train and test we prevent the over fitting part of dataseet
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=40)
```

In [28]:

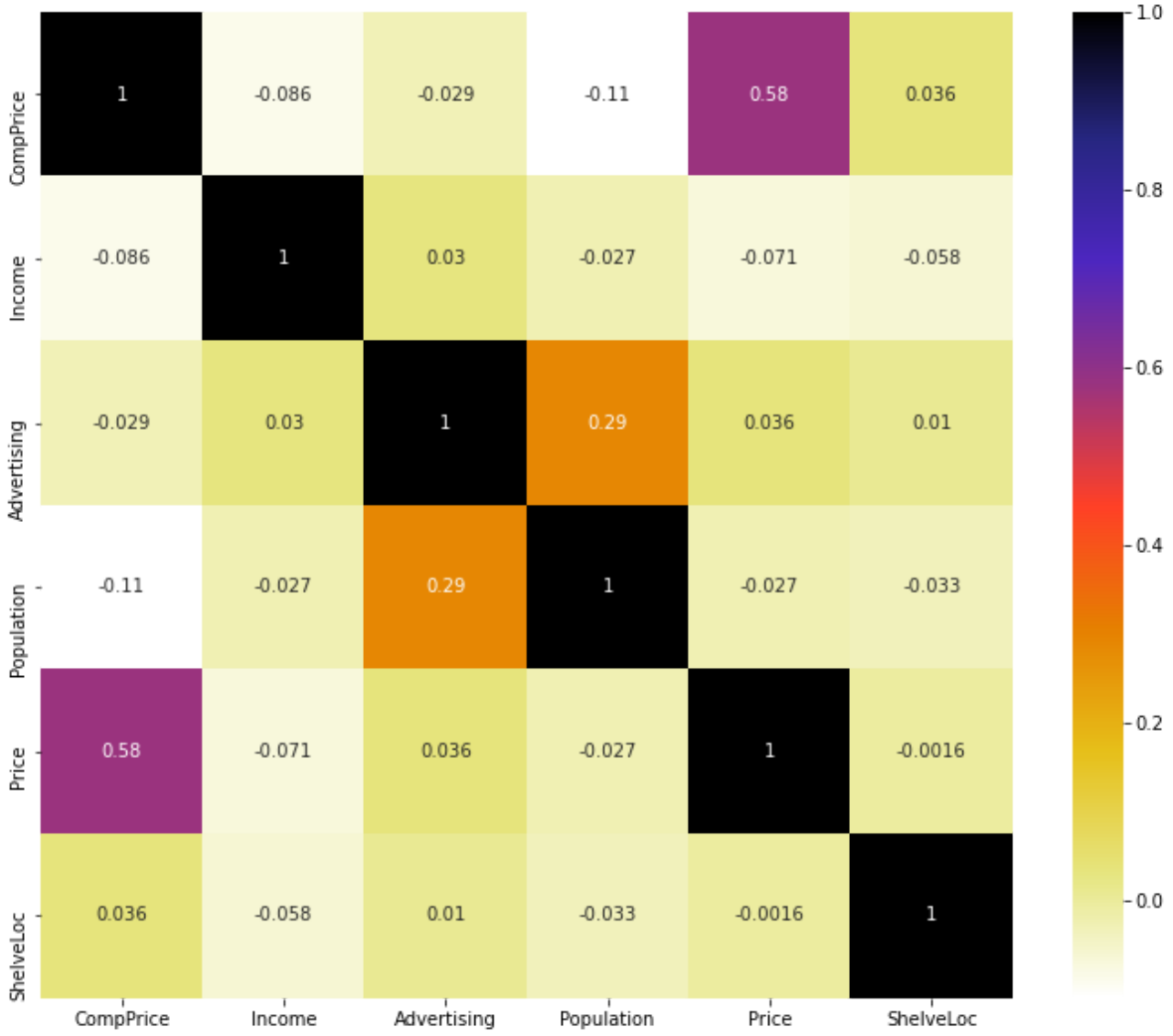
```
### Correlation on X_train

x_train.corr()
```

Out[28]:

	CompPrice	Income	Advertising	Population	Price	ShelveLoc
CompPrice	1.000000	-0.085962	-0.029148	-0.109752	0.579708	0.036428
Income	-0.085962	1.000000	0.030434	-0.026637	-0.070657	-0.057838
Advertising	-0.029148	0.030434	1.000000	0.293126	0.035999	0.010429
Population	-0.109752	-0.026637	0.293126	1.000000	-0.027049	-0.032543
Price	0.579708	-0.070657	0.035999	-0.027049	1.000000	-0.001627
ShelveLoc	0.036428	-0.057838	0.010429	-0.032543	-0.001627	1.000000

In [29]:

```
import seaborn as sns
# using pearson correlation,we doing correlation on training dataset only
plt.figure(figsize=(12,10))
cor = x_train.corr()
sns.heatmap(cor,annot=True,cmap=plt.cm.CMRmap_r)
plt.show()
```



In [30]:

```
# with the following feature we can select highly correlated feature
# it wll remove the first feature that is correlated with anything other feature

def correlation(dataset,threshold):
    col_corr = set()  # set of all the names of correlated cols
    corr_metrix = dataset.corr()
    for i in range(len(corr_metrix.columns)):
        for j in range(i):
            if abs(corr_metrix.iloc[i,j]) > threshold: # we r interesting in absolute coeff
                colnames = corr_metrix.columns[i]  # getting the names of cols
                col_corr.add(colnames)
    return col_corr
```

In [31]:

```
corr_features = correlation(x_train,0.8)
len(set(corr_features))
```

Out[31]:

```
0
```

Here we found Zero correlation in our dataset x_train

Model Building

Random Forest

In [32]:

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier


num_forest = 100
max_features = 3
kfold = KFold(n_splits = 10, random_state = 40, shuffle = True)
model = RandomForestClassifier(n_estimators = num_forest, max_features = max_features)
result = cross_val_score(model, x, y, cv=kfold)

print(result.mean())
```

```
0.735
```

Test-Accuracy: 73.5%

In [33]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
# ensemble learning it takes learning from multiple model and combine it
num_forest = 100
max_features = 1
model = RandomForestClassifier(n_estimators = 100, max_features = 1)
kfold = KFold(n_splits = 20, random_state = 40, shuffle = True)
model.fit(x_train, y_train)
result = cross_val_score(model, x, y, cv=kfold)
print(result.mean())
```

```
0.7174999999999999
```

Train-Accuracy: 72.%

In [ ]:

```

```

Assignment-15- Random-Forest-02-(Fraud-Check)

Use decision trees to prepare a model on fraud data treating those who have taxable_income <= 30000 as "Risky" and others are "Good"

In [34]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
```

In [35]:

```
data=pd.read_csv("C:/Users/LENOVO/Documents/Custom Office Templates/Fraud_check.csv")
data.head()
```

Out[35]:

	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 [36]:

```
data.shape
```

Out[36]:

```
(600, 6)
```

In [37]:

```
data.columns
```

Out[37]:

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

In [38]:

```
data.describe()
```

Out[38]:

	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 [39]:

```
data.loc[data['Taxable.Income']>=30000, 'Income']='Good'
data.loc[data['Taxable.Income']<=30000, 'Income']='Risky'
```

In [40]:

```
data
```

Out[40]:

	Undergrad	Marital.Status	Taxable.Income	City.Population	Work.Experience	Urban	Incom
0	NO	Single	68833	50047	10	YES	Goo
1	YES	Divorced	33700	134075	18	YES	Goo
2	NO	Married	36925	160205	30	YES	Goo
3	YES	Single	50190	193264	15	YES	Goo
4	NO	Married	81002	27533	28	NO	Goo
...	...	...	...	...	...	...	.
595	YES	Divorced	76340	39492	7	YES	Goo
596	YES	Divorced	69967	55369	2	YES	Goo
597	NO	Divorced	47334	154058	0	YES	Goo
598	YES	Married	98592	180083	17	NO	Goo
599	NO	Divorced	96519	158137	16	NO	Goo

600 rows × 7 columns

In [41]:

```
label_encoder= preprocessing.LabelEncoder()
data['Income']= label_encoder.fit_transform(data['Income'])
data['Undergrad']= label_encoder.fit_transform(data['Undergrad'])
data['Urban']= label_encoder.fit_transform(data['Urban'])
data['Marital.Status']= label_encoder.fit_transform(data['Marital.Status'])
```

In [42]:

```
data
```

Out[42]:

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

600 rows × 7 columns

In [43]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 600 entries, 0 to 599
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Undergrad        600 non-null    int32
 1   Marital.Status   600 non-null    int32
 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    int32
 6   Income           600 non-null    int32
dtypes: int32(4), int64(3)
memory usage: 23.6 KB
```

In [44]:

```
#Droping the Taxable income variable - we taken that as Income description
data.drop(["Taxable.Income"],axis=1,inplace=True)
```

In [45]:

```
data.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    int32
 1   Marital.Status   600 non-null    int32
 2   City.Population  600 non-null    int64
 3   Work.Experience  600 non-null    int64
 4   Urban            600 non-null    int32
 5   Income           600 non-null    int32
dtypes: int32(4), int64(2)
memory usage: 18.9 KB
```

No Null Values

In [46]:

```
data['Income'].value_counts()
```

Out[46]:

```
0    476
1    124
Name: Income, dtype: int64
```

In [47]:

```
#diving data for traing and test
X = data.iloc[:,0:5]
Y = data.iloc[:,5]
```

In [48]:

```
X
```

Out[48]:

	Undergrad	Marital.Status	City.Population	Work.Experience	Urban
0	0	2	50047	10	1
1	1	0	134075	18	1
2	0	1	160205	30	1
3	1	2	193264	15	1
4	0	1	27533	28	0
...	...	...	...	...	...
595	1	0	39492	7	1
596	1	0	55369	2	1
597	0	0	154058	0	1
598	1	1	180083	17	0
599	0	0	158137	16	0

600 rows × 5 columns

In [49]:

```
Y
```

Out[49]:

```
0      0
1      0
2      0
3      0
4      0
      ..
595    0
596    0
597    0
598    0
599    0
Name: Income, Length: 600, dtype: int32
```

In [50]:

```
data['Income'].unique()
```

Out[50]:

```
array([0, 1])
```

In [51]:

```
# Splitting data into training and testing data
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.20, random_state=40)
```

In [52]:

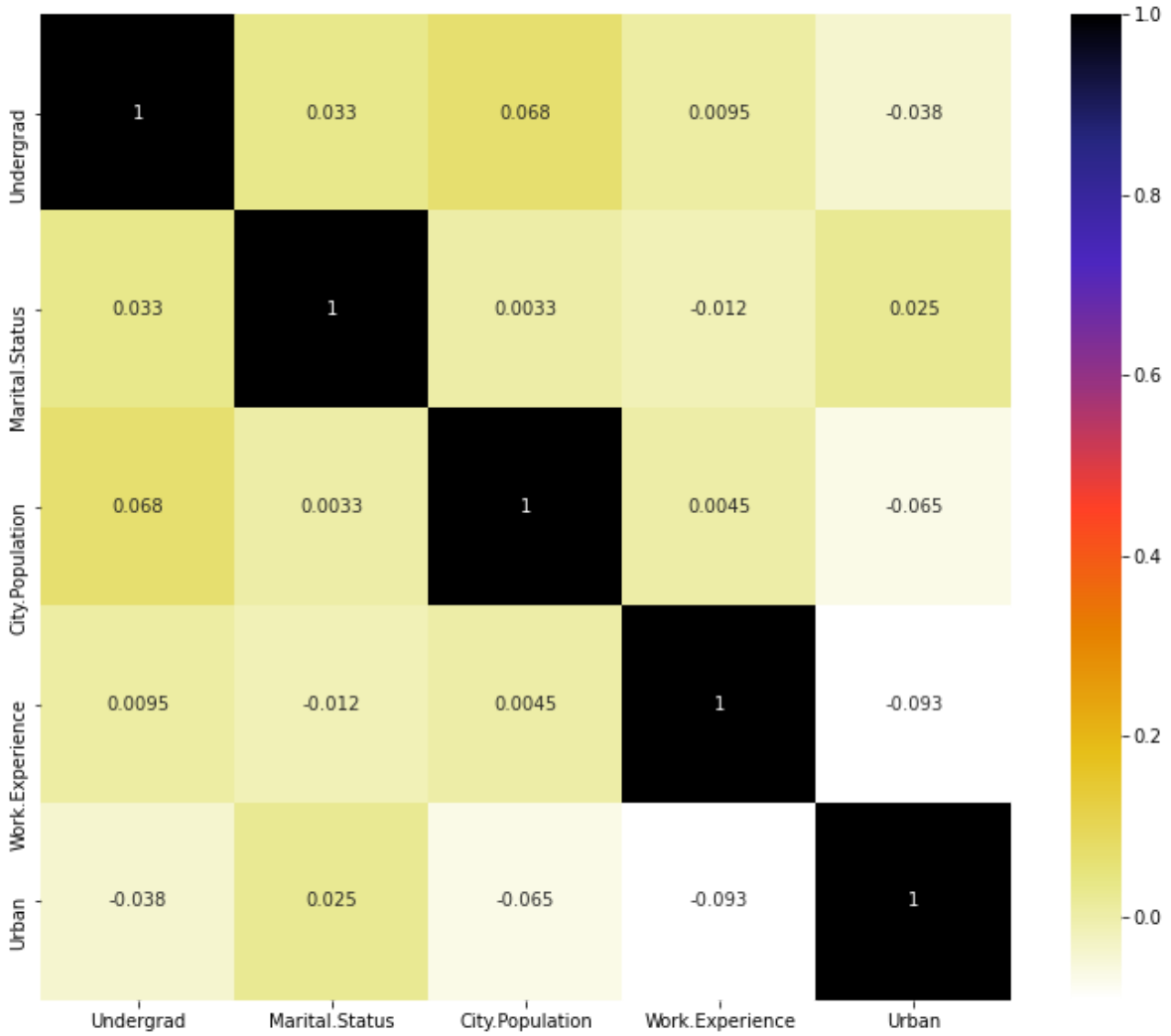
```
### Correlation on X_train

X_train.corr()
```

Out[52]:

	Undergrad	Marital.Status	City.Population	Work.Experience	Urban
Undergrad	1.000000	0.033021	0.067849	0.009482	-0.037526
Marital.Status	0.033021	1.000000	0.003301	-0.012498	0.025334
City.Population	0.067849	0.003301	1.000000	0.004496	-0.065461
Work.Experience	0.009482	-0.012498	0.004496	1.000000	-0.092568
Urban	-0.037526	0.025334	-0.065461	-0.092568	1.000000

In [53]:

```
# using pearson corrilation,we doing corrilation on X_training dataset  only
plt.figure(figsize=(12,10))
cor = X_train.corr()
sns.heatmap(cor,annot=True,cmap=plt.cm.CMRmap_r)
plt.show()
```



In [54]:

```
# with the following feature we can select highly correlated feature
# it wll remove the first feature that is if it is highly correlated with anything other fe

def correlation(dataset,threshold):
    col_corr = set()  # set of all the names of correlated cols
    corr_metrix = dataset.corr()
    for i in range(len(corr_metrix.columns)):
        for j in range(i):
            if (corr_metrix.iloc[i,j]) > threshold: # we r interesting in absolute coeff.va
                colnames = corr_metrix.columns[i]  # getting the names of cols
                col_corr.add(colnames)
    return col_corr
```

In [55]:

```
corr_features = correlation(X_train,0.8)
len(set(corr_features))
```

Out[55]:

```
0
```

Here we found Zero correlation in our dataset X_train

Model building

Random forest

In [56]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
# ensemble learning it takes learning from multiple model and combine it
num_forest = 100
max_features = 1
model = RandomForestClassifier(n_estimators = 100, max_features = 1)
kfold = KFold(n_splits = 20, random_state = 40, shuffle = True)
model.fit(X_train, Y_train)
result = cross_val_score(model, X, Y, cv=kfold)
print(result.mean())
```

```
0.7316666666666666
```

Accuracy of test model :- 73.16 %

In [57]:

```
model.score(X_train, Y_train)
```

Out[57]:

```
1.0
```

Accuracy of train model :- 100 %

In [58]:

```
Y_predict = model.predict(X_test)
```

In [59]:

```
from sklearn.metrics import confusion_matrix
result = confusion_matrix(Y_test, Y_predict)
```

In [60]:

```
# it plot 2 dimentional array
result
```

Out[60]:

```
array([[82, 10],
       [27,  1]], dtype=int64)
```

In [ ]:

```

```