Random Forests Assignment

Data Set - Fraud_check

1. Import Necessary libraries

In [1]:
```
import pandas as pd
import numpy as np

from matplotlib import pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

2. Import Data

In [2]:
```
fraud_details = pd.read_csv('Fraud_check.csv')
fraud_details
```

Out[2]:

	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

3. Data Understanding

In [3]:
```
fraud_details.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_details.shape
```

Out[4]:
```
(600, 6)
```

In [5]:
```
fraud_details.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 [6]:
```
fraud_details.isna().sum()
```

Out[6]:
```
Undergrad          0
Marital.Status     0
Taxable.Income     0
City.Population    0
Work.Experience    0
Urban              0
dtype: int64
```

In [7]:
```
fraud_details.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_details.dtypes
```

Out[8]:
```
Undergrad          object
Marital.Status     object
Taxable.Income      int64
City.Population     int64
Work.Experience     int64
Urban              object
dtype: object
```

In [9]:
```
fraud_details.columns
```

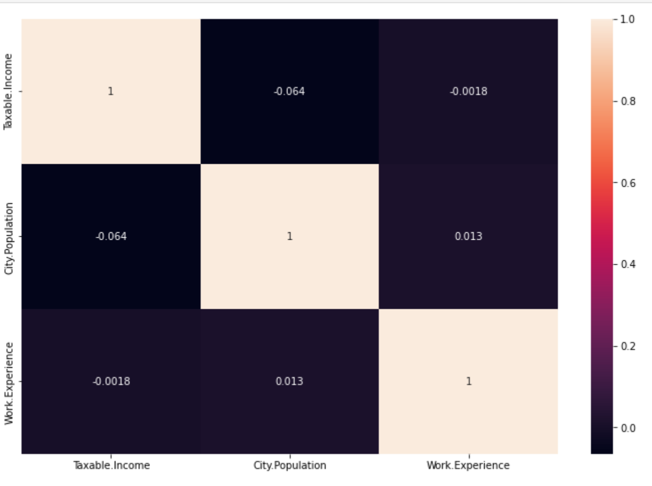
Out[9]:
```
Index(['Undergrad', 'Marital.Status', 'Taxable.Income', 'City.Population',
       'Work.Experience', 'Urban'],
      dtype='object')
```

In [10]:
```
fraud_details['Marital.Status'].value_counts()
```

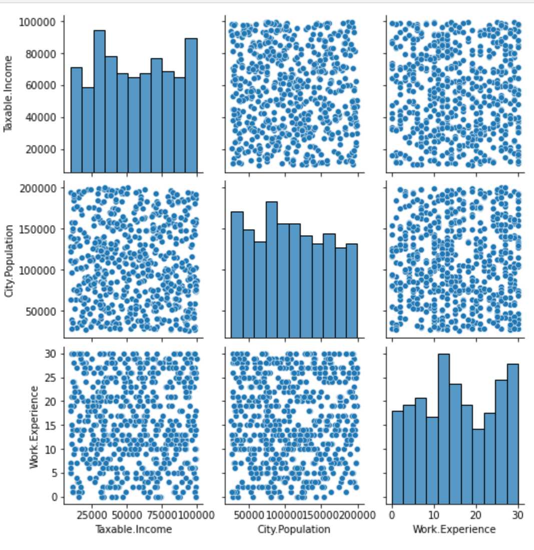
Out[10]:
```
Single      217
Married     194
Divorced    189
Name: Marital.Status, dtype: int64
```

3.1 Correlation Matrix :

In [11]:
```
plt.figure(figsize = (12,8))
sns.heatmap(fraud_details.corr(),annot = True)
plt.show()
```



In [12]:
```
sns.pairplot(fraud_details)
plt.show()
```



3.2 Label Encoder :

In [13]:
```
from sklearn import preprocessing
```

In [14]:
```
label_encoder = preprocessing.LabelEncoder()
label_encoder
```

Out[14]:
```
LabelEncoder()
```

In [15]:
```
fraud_details['Undergrad'] = label_encoder.fit_transform(fraud_details['Undergrad'])
fraud_details['Marital.Status'] = label_encoder.fit_transform(fraud_details['Marital.Status'])
fraud_details['Urban'] = label_encoder.fit_transform(fraud_details['Urban'])
```

In [16]:
```
fraud_details
```

Out[16]:

	Undergrad	Marital.Status	Taxable.Income	City.Population	Work.Experience	Urban
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 × 6 columns

3.3 Adding New Column :

In [17]:
```
fraud_details['Status'] = fraud_details['Taxable.Income'].apply(lambda Income: 'Risky' if Income <= 30000 else 'Good')
fraud_details
```

Out[17]:

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

600 rows × 7 columns

In [18]:
```
fraud_details['Status'].unique()
```

Out[18]:
```
array(['Good', 'Risky'], dtype=object)
```

In [19]:
```
fraud_details['Status'] = label_encoder.fit_transform(fraud_details['Status'])
fraud_details
```

Out[19]:

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

600 rows × 7 columns

3.4 Splitting in x and y :

In [20]:
```
x = fraud_details.iloc[:,0:4]
x
```

Out[20]:

	Undergrad	Marital.Status	Taxable.Income	City.Population
0	0	2	68833	50047
1	1	0	33700	134075
2	0	1	36925	160205
3	1	2	50190	193264
4	0	1	81002	27533
...	...	...	...	...
595	1	0	76340	39492
596	1	0	69967	55369
597	0	0	47334	154058
598	1	1	98592	180083
599	0	0	96519	158137

600 rows × 4 columns

In [21]:
```
y = fraud_details['Status']
y
```

Out[21]:
```
0      0
1      0
2      0
3      0
4      0
      ..
595    0
596    0
597    0
598    0
599    0
Name: Status, Length: 600, dtype: int32
```

4. Bagged Decision Trees for Classification

In [22]:
```
from sklearn.model_selection import KFold
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.model_selection import cross_val_score
```

In [23]:
```
cart_1 = DecisionTreeClassifier()
```

In [24]:
```
num_trees_1 = 100
```

In [25]:
```
k_fold_1 = KFold(n_splits = 10,shuffle = True, random_state = 8)
model_1 = BaggingClassifier(base_estimator = cart_1, n_estimators = num_trees_1, random_state = 8)
```

In [26]:
```
results = cross_val_score(model_1, x,y, cv = k_fold_1)
print(results.mean())
```

```
0.9983333333333334
```

5. Random Forest Classification

In [27]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [28]:
```
cart_2 = RandomForestClassifier()
```

In [29]:
```
num_trees_2 = 100
max_features = 3
```

In [30]:
```
k_fold_2 = KFold(n_splits = 10, shuffle = True ,random_state = 8)
model_2 = RandomForestClassifier(n_estimators = num_trees_2, max_features = max_features)
```

In [31]:
```
results = cross_val_score(model_2, x, y, cv = k_fold_2)
print(results.mean())
```

```
0.9983333333333334
```

6. Boost Classification

In [32]:
```
from sklearn.ensemble import AdaBoostClassifier
```

In [33]:
```
cart_3 = AdaBoostClassifier()
```

In [34]:
```
seed = 8
num_trees_3 = 100
```

In [35]:
```
k_fold_3 = KFold(n_splits = 100, shuffle = True, random_state = seed)
model_3 = AdaBoostClassifier(n_estimators = num_trees_3, random_state = seed)
```

In [36]:
```
results = cross_val_score(model_3, x,y, cv = k_fold_3)
print(results.mean())
```

```
0.9983333333333334
```

7. Stacking Ensemble for Classification

In [37]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier
```

In [38]:
```
k_fold_4 = KFold(n_splits = 10,shuffle = True, random_state = 8)
estimators = []
```

Create the sub models

In [39]:
```
model_4 = LogisticRegression(max_iter = 100)
estimators.append(('logistic', model_4))
```

In [40]:
```
model_5 = DecisionTreeClassifier()
estimators.append(('cart', model_5))
```

In [41]:
```
model_6 = SVC()
estimators.append(('svm', model_6))
```

Create the ensemble model

In [42]:
```
ensemble = VotingClassifier(estimators)
results = cross_val_score(ensemble, x, y, cv = k_fold_4)
print(results.mean())
```

```
0.9833333333333332
```

Conclusion :

```
For Decision Tree Classifier, Random Forest Classifier & Ada Boost Classifier Models the Accuracy is : 0.9983

For Stacking Ensemble Model the Accuracy is : 0.9833
```