Problem Statement

To identify potential defaulters from a pool of new customers based on learning based tree metho

Dataset

Source of Data

We have a dataset of 1319 records of previous credit card applicants to the bank.

Goal: Predict whether a credit card application will be accepted based upon various data about the

Importing necessary packages

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
%matplotlib inline
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: Futur
  import pandas.util.testing as tm
```

```
print(np.__version__)
print(pd.__version__)
print(sns.__version__)
print(sklearn.__version__)
```

```
1.18.4
1.0.3
0.10.1
0.22.2.post1
```

load the dataset from the csv file using pandas

```
!wget https://gist.githubusercontent.com/ucalyptus/905b6e56e5db73b2953624be72
data=pd.read_csv("/content/dataset.csv")
data.shape
```

```
--2020-05-24 14:30:29--  https://gist.githubusercontent.com/ucalyptus/905b6e5
Resolving gist.githubusercontent.com (gist.githubusercontent.com)... 151.101.0
Connecting to gist.githubusercontent.com (gist.githubusercontent.com)|151.101.
HTTP request sent, awaiting response... 200 OK
Length: 73250 (72K) [text/plain]
Saving to: 'dataset.csv'
```

displaying the data

```
2020-05-24 14:30:29 (2.58 MB/s) - 'dataset.csv' saved [73250/73250]
```

```
data.head()
```

	card	reports	age	income	share	expenditure	owner	selfemp	depend
0	yes	0	37.66667	4.5200	0.033270	124.983300	yes	no	
1	yes	0	33.25000	2.4200	0.005217	9.854167	no	no	
2	yes	0	33.66667	4.5000	0.004156	15.000000	yes	no	
3	yes	0	30.50000	2.5400	0.065214	137.869200	no	no	
4	yes	0	32.16667	9.7867	0.067051	546.503300	yes	no	

Exploratory Data Analysis

Column Description

- card: Dummy variable, 1 if application for credit card accepted, 0 if not
- reports: Number of major derogatory reports
- age: Age n years plus twelfths of a year
- income: Yearly income (divided by 10,000)
- share: Ratio of monthly credit card expenditure to yearly income
- expenditure: Average monthly credit card expenditure
- owner: 1 if owns their home, 0 if rent
- selfempl: 1 if self employed, 0 if not.
- dependents: 1 + number of dependents
- months: Months living at current address
- majorcards: Number of major credit cards held
- active: Number of active credit accounts

What are the attributes of our dataset?

```
# printing the column names
print(data.columns)
```

```
Index(['card', 'reports', 'age', 'income', 'share', 'expenditure', 'owner',
```

```
# displaying the dimension of dataset
print(data.shape)
```

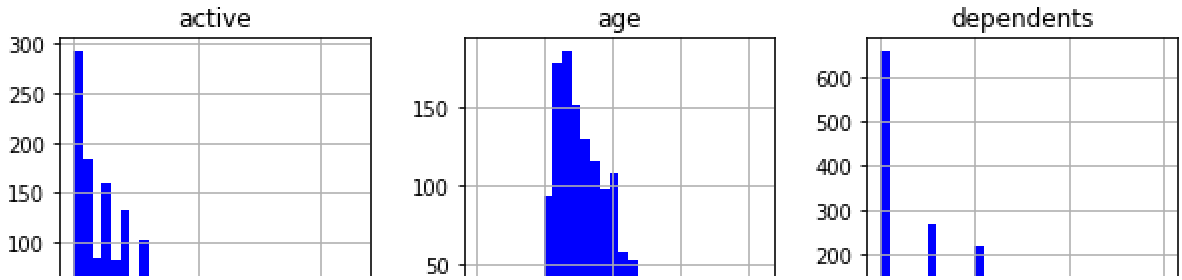
```
(1319, 12)
```

Let's check the statistics of our dataset and look for missing values,other disc

```
#describing the numerical attributes
print(data.describe())
```

```
           reports          age  ...   majorcards       active
count  1319.000000  1319.000000  ...  1319.000000  1319.000000
mean      0.456406    33.213103  ...     0.817286     6.996967
std       1.345267    10.142783  ...     0.386579     6.305812
min       0.000000     0.166667  ...     0.000000     0.000000
25%       0.000000    25.416670  ...     1.000000     2.000000
50%       0.000000    31.250000  ...     1.000000     6.000000
75%       0.000000    39.416670  ...     1.000000    11.000000
max      14.000000    83.500000  ...     1.000000    46.000000

[8 rows x 9 columns]
```

```
#plot the histogram of each parameter
import warnings
warnings.filterwarnings('ignore')
data.hist(color='blue',bins=30,figsize=(10,10))
plt.show()
```

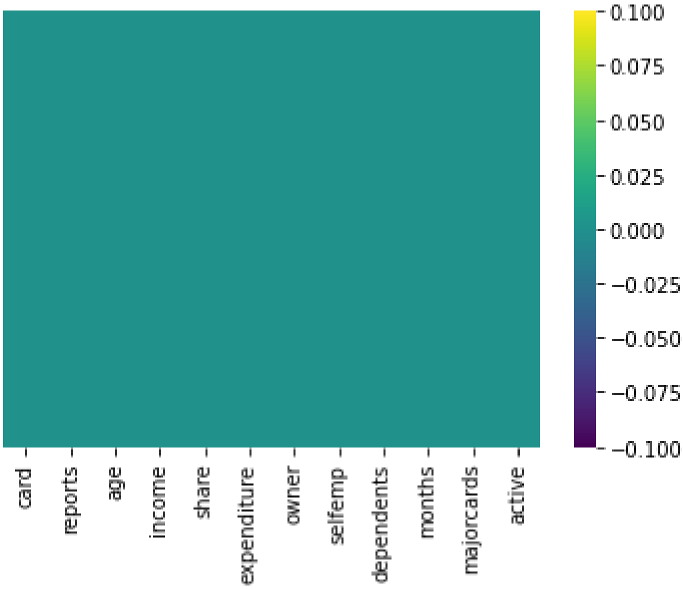


Does my data have missing values?

expenditure income majorcards

```
sns.heatmap(data.isnull(),yticklabels=False,cbar=True,cmap='viridis')
#no null values exists in any of the attributes
```

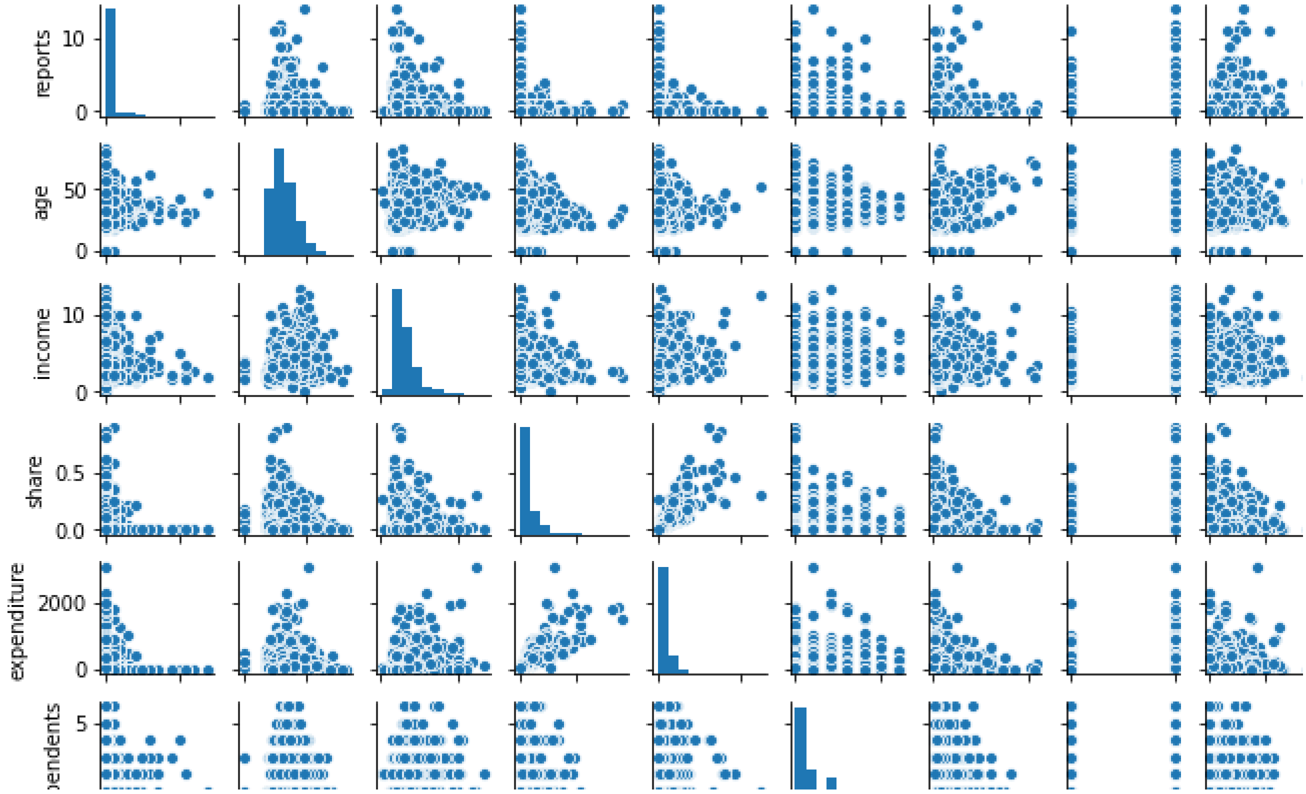
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff0b9738b38>
```



No missing values found. Can we check out for any correlations betw dataset?

```
sns.pairplot(data,height=1)
```

```
<seaborn.axisgrid.PairGrid at 0x7ff0b5c5e860>
```



As you can see

There is no linear relation between pairs of age,income,expenditure,reports. These seem to be the manually removed the features which have correlations with these features for dimensionality redu essential for small datasets to avoid overfitting.We do not want our model to memorize the datase a new input during inference phase. Common algorithms for Dimensionality Reduction are :

- Prinicpal Component Analysis
- t-SNE
- UMAP
- Linear Discriminant Analysis (Gaussian)

```
#displaying the dataset
data.head()
```

	card	reports	age	income	share	expenditure	owner	selfemp	depend
0	yes	0	37.66667	4.5200	0.033270	124.983300	yes	no	
1	yes	0	33.25000	2.4200	0.005217	9.854167	no	no	
2	yes	0	33.66667	4.5000	0.004156	15.000000	yes	no	
3	yes	0	30.50000	2.5400	0.065214	137.869200	no	no	
4	yes	0	32.16667	9.7867	0.067051	546.503300	yes	no	

Converting Categorical features to numerical variables.

This can be done using One-Hot Encoding. However, one-hot encoding is recommended only if you

```
# replacing category to numerical value
data.card.replace(['yes','no'], ['1', '0'], inplace=True)
data.owner.replace(['yes','no'], ['1', '0'], inplace=True)
data.selfemp.replace(['yes','no'], ['1', '0'], inplace=True)
data.head()
```

	card	reports	age	income	share	expenditure	owner	selfemp	depend
0	1	0	37.66667	4.5200	0.033270	124.983300	1	0	
1	1	0	33.25000	2.4200	0.005217	9.854167	0	0	
2	1	0	33.66667	4.5000	0.004156	15.000000	1	0	
3	1	0	30.50000	2.5400	0.065214	137.869200	0	0	
4	1	0	32.16667	9.7867	0.067051	546.503300	1	0	

Classwise distribution

```
# Count Target Variable Values
data.card.value_counts()

```

We see that this is an imbalanced dataset.

Below are the percentages of the two classes.

```
# count target variable percentage
round(data.card.value_counts()*100/len(data.axes[0]),2)
```

selecting the feature attributes

```
X=data[["reports","expenditure","age","income"]]
y=data["card"]
```

```
X.head()
```

```
y.head()
```

```
0    1
1    1
2    1
3    1
4    1
Name: card, dtype: object
```

splitting the train and test set

```
from sklearn.model_selection import train_test_split
```

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_sta
```

```
#length of train set
len(X_train)
```

```
923
```

```
#length of test set
len(X_test)
```

```
396
```

```
# displaying train set
X_train.head()
```

	reports	expenditure	age	income
831	0	105.32500	39.91667	4.60
1239	0	0.00000	53.00000	11.00
280	0	80.45084	29.83333	2.80
82	0	108.60750	19.58333	1.65
1117	0	128.07250	22.91667	1.56

```
# displaying test set
X_test.head()
```

	reports	expenditure	age	income
56	0	1898.03300	34.33333	4.800
711	0	20.33333	36.41667	2.200
912	0	0.00000	29.33333	3.500
1053	1	0.00000	28.50000	2.000

```
 #Load library
#from sklearn import tree

# Building Decision Tree - CART Algorithm (gini criteria)
dt_train_gini_decision = DecisionTreeClassifier(criterion = "gini", random_sta
                               max_depth=5, min_samples_leaf=5)
# Train
dt_train_gini_decision.fit(X_train,y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=5, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=5, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=100, splitter='best')
```

```
#apply the classifier we trained to the test data(which,remember,it has never
dt_train_gini_decision.predict(X_test) #test matching
#passing x and getting y in predict func
```

```
array(['1', '1', '0', '0', '0', '0', '0', '0', '1', '1', '1', '1', '0',
       '1', '1', '1', '1', '0', '1', '1', '1', '0', '0', '1', '1', '1',
       '1', '1', '0', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1',
```

```
#my classifier is trained now
#predict the probability of 1st 10 obs
dt_train_gini_decision.predict_proba(X_test)[0:10]
```

```
array([[0.        , 1.        ],
       [0.        , 1.        ],
       [1.        , 0.        ],
       [1.        , 0.        ],
       [1.        , 0.        ],
       [1.        , 0.        ],
       [0.55555556, 0.44444444],
       [0.91666667, 0.08333333],
       [0.        , 1.        ],
       [0.        , 1.        ]])
       '1', '1', '1', '1', '1', '0', '1', '1', '1', '1', '1', '0', '1',
```

```
# displaying the predicted value for test set
preds= dt_train_gini_decision.predict(X_test)
preds[0:10]
```

```
array(['1', '1', '0', '0', '0', '0', '0', '0', '1', '1'], dtype=object)
       '0', '1', '1', '1', '0', '1', '1', '1', '1', '1', '0', '0', '1',
```

```
#displaying the actual value of test set
y_test.head(10)
```

```
56      1
711     1
912     0
1053    0
1142    0
305     0
461     0
1243    0
582     1
963     1
Name: card, dtype: object
```

creating a confusion matrix

```
pd.crosstab(y_test,preds,rownames=["Actual result"],colnames=["Predicted resul
```

Predicted result	0	1
Actual result		
0	87	0
1	8	301

```
import sklearn
from sklearn.metrics import classification_report,roc_auc_score
print(roc_auc_score(y_test,preds))
print(classification_report(y_test,preds))
```

```
0.9870550161812297
              precision    recall  f1-score   support

           0       0.92      1.00      0.96        87
           1       1.00      0.97      0.99       309

    accuracy                           0.98       396
   macro avg       0.96      0.99      0.97       396
weighted avg       0.98      0.98      0.98       396
```

```
#view a list of the features and theirv imoportant score
list(zip(X_train,dt_train_gini_decision.feature_importances_))
#which attributes are more  decisive...used in overfitting to drop columns

```

```
[('reports', 0.004721763304312109),
 ('expenditure', 0.9783798350814344),
 ('age', 0.010235979525498706),
 ('income', 0.0066624220887548545)]
```

As you can see

Expenditure has a feature importance out 0.97 in a scale of 0 to 1 (continuous). This shows that m basis of expenditure which is logically meaningful as **Repayment Styles** and "**How we spend mone** deserve creidit approval or not.

```
eff=dt_train_gini_decision.score(X_test,y_test)
```

```
print(eff*100)
```

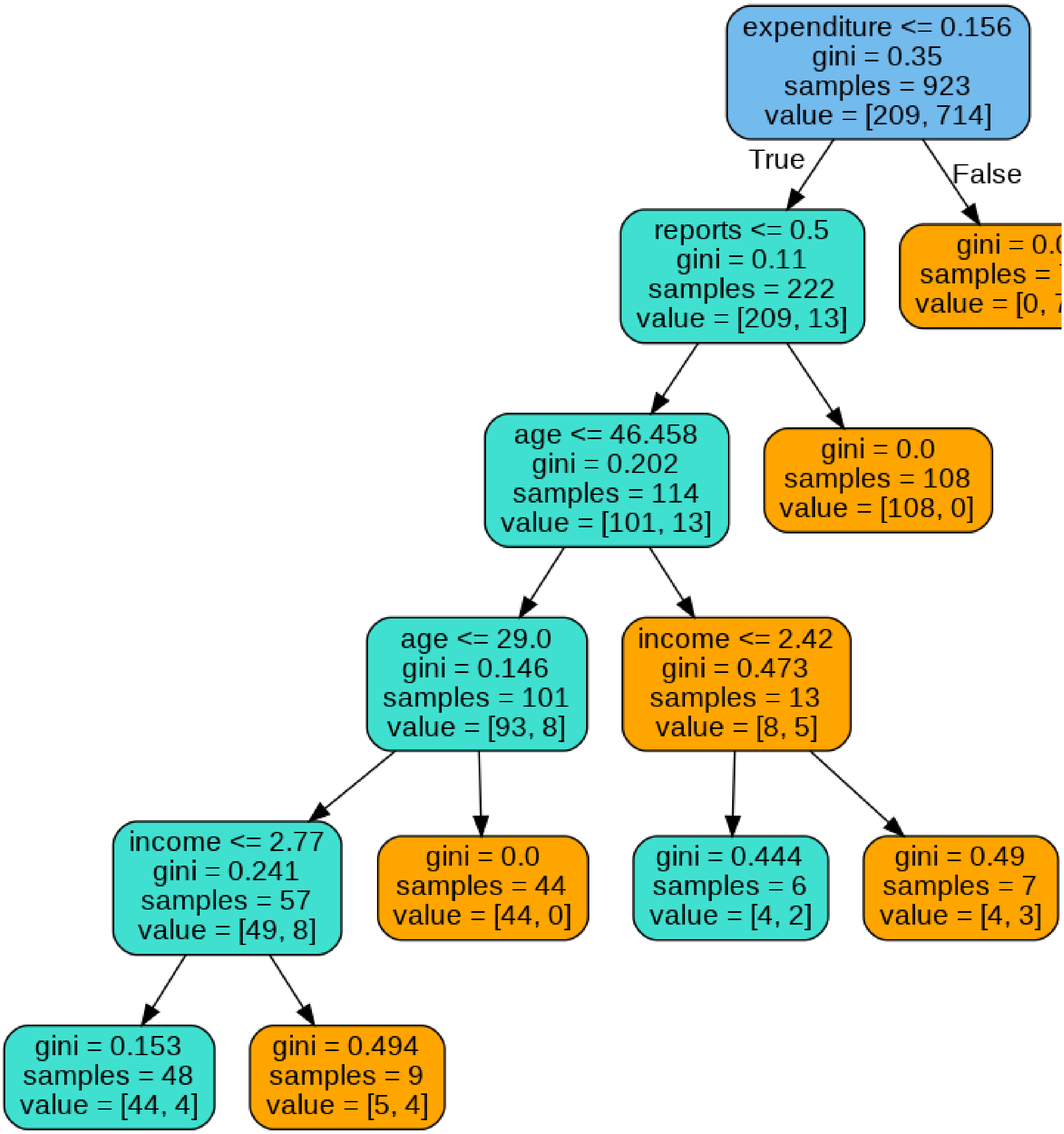
```
97.97979797979798
```

```
import pydotplus
import collections
from sklearn import tree
## sudo apt-get install graphviz
def visualize_graph(clf,features,name):
    # Visualize data
    dot_data = tree.export_graphviz(clf,
                                    feature_names=features,
                                    out_file=None,
                                    filled=True,
                                    rounded=True)
    graph = pydotplus.graph_from_dot_data(dot_data)

    colors = ('turquoise', 'orange')
    edges = collections.defaultdict(list)

    for edge in graph.get_edge_list():
        edges[edge.get_source()].append(int(edge.get_destination()))

```

```
    for edge in edges:
        edges[edge].sort()
        for i in range(2):
            dest = graph.get_node(str(edges[edge][i]))[0]
            dest.set_fillcolor(colors[i])

    graph.write_png(name+".png")
```

```
visualize_graph(dt_train_gini_decision,X.columns,"dt_en")
```

```
from IPython.display import Image
Image('/content/dt_en.png',width=700, height=700)
```



Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
clf=RandomForestClassifier(n_jobs=-1,random_state=0)
clf.fit(X_train,y_train)
clf.predict(X_test)
clf.predict_proba(X_test)[0:10]
```

```
array([[0.  , 1.  ],
       [0.  , 1.  ],
       [0.97, 0.03],
       [1.  , 0.  ],
       [0.95, 0.05],
       [0.95, 0.05],
       [0.29, 0.71],
       [0.91, 0.09],
       [0.  , 1.  ],
       [0.  , 1.  ]])
```

```
list(zip(X_train,clf.feature_importances_))
```

```
[('reports', 0.11000605280019297),
 ('expenditure', 0.7961741426706236),
 ('age', 0.04579285219474255),
 ('income', 0.048026952334440993)]
```

```
eff=clf.score(X_test,y_test)*100
print(eff)
```

```
97.72727272727273
```

```
import joblib
from joblib import dump, load
dump(clf, 'model.joblib')
```

```
['model.joblib']
```

```
data.income.min()
#data.income.max()
```

```
0.21
```

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(clf, X_train, y_train, cv=5)
```

```
scores
```

```
array([0.97837838, 0.98918919, 0.98378378, 0.97282609, 0.97282609])
```

```
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Accuracy: 0.98 (+/- 0.01)
```

```
scores = cross_val_score(clf, X_train, y_train, cv=5, scoring='f1_macro')

```

```
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Accuracy: 0.97 (+/- 0.02)
```

```
from sklearn.model_selection import ShuffleSplit
n_samples = X.shape[0]
cv = ShuffleSplit(n_splits=5, test_size=0.3, random_state=0)
cross_val_score(clf, X, y, cv=cv)
```

```
array([0.97979798, 0.98232323, 0.97222222, 0.97474747, 0.97222222])
```

```
X_test
```

	reports	expenditure	age	income
56	0	1898.03300	34.33333	4.800
711	0	20.33333	36.41667	2.200
912	0	0.00000	29.33333	3.500
1053	1	0.00000	28.50000	2.000
1142	1	0.00000	20.91667	2.625
...	...	...	...	...
481	0	3.75000	28.16667	3.500
378	0	398.85080	26.58333	3.200
1038	0	66.73250	44.08333	4.000
980	1	42.31083	30.66667	2.740
1299	0	38.99750	38.75000	3.200

396 rows × 4 columns

```
len(clf.predict(X_test))
```