Assignment 3

```
from sklearn import svm
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.utils import resample
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
df = pd.read_csv("original.csv")
df.head()
```

[2]:

	clientid	income	age	loan	default
0	1	66155.925095	59.017015	8106.532131	0
1	2	34415.153966	48.117153	6564.745018	0
2	3	57317.170063	63.108049	8020.953296	0
3	4	42709.534201	45.751972	6103.642260	0
4	5	66952.688845	18.584336	8770.099235	1

Data Pre-Processing

Checked if is there any nan value in the dataset and dropped the index for 3 Nan value which is in the age column.

```
print(df.isna().sum())
print(f"\nTotal records before removing NAN values: {df.shape[0]}")
```

```
clientid    0
income      0
age         3
loan        0
default     0
dtype: int64

Total records before removing NAN values: 2000
```

```
df = df.dropna()
print(f"\nTotal records after removing NAN values: {df.shape[0]}")
```

```
Total records after removing NAN values: 1997
```

Unbalanced Data set

Here we have 1714 negative samples while only 283 positive samples.

```
df.default.value_counts()
```

[5]:

```
0    1714
1     283
Name: default, dtype: int64
```

```
minority = df[df.default == 1]
majority = df[df.default == 0]
```

Assignment 3

Upsampled the data to match the majority.

```python
minority_upsampled = resample(minority, replace=True, n_samples=1714, random_state=7)
df_upsampled = pd.concat([minority_upsampled, majority])
df_upsampled.default.value_counts()
```

```
7]: 1    1714
    0    1714
    Name: default, dtype: int64
```

Data Scaling

Here, we have used the MinMaxScaler function with feature range 0 to 1. It scales the values to a specific value range[0,1] without changing the shape of the original distribution.

```python
features = df.columns.values
scaler = MinMaxScaler(feature_range=(0,1))
df_upsampled_noscale = df_upsampled
scaler.fit(df_upsampled)
df_upsampled_scaled = pd.DataFrame(scaler.transform(df_upsampled))
df_upsampled_scaled.columns = features
```

Extracting Features from our pre-processed dataframe.

```python
X = df_upsampled_scaled.drop(['clientid', 'default'], axis=1)
y = df_upsampled_scaled['default']
```

Spliting the data with 80% train size.

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=7)
```

Support Vector Classifier

- **kernel** parameters selects the type of hyperplane used to separate the data. Using 'linear' will use a linear hyperplane, 'rbf' and 'poly' uses a non linear hyper-plane.
- **gamma** is a parameter for non linear hyperplanes. The higher the gamma value it tries to exactly fit the training data set.
- **C** is the penalty parameter of the error term. It controls the trade off between smooth decision boundary and classifying the training points correctly.

Below Cell represents the 'rbf' kernel with oprimized gamma and C value in order to achieve a acurate result.

```python
svc_classifier = svm.SVC(kernel='rbf', gamma=0.8, C=10).fit(X_train, y_train)
y_pred = svc_classifier.predict(X_test)
svc_score = svc_classifier.score(X_test, y_test)
svc_score
```

```
11]: 0.9533527696793003
```

Here we have achieved the 92% precision with the accuracy of 95%, which describes that our model is a good model.

```python
accuracy = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
f1score = f1_score(y_test, y_pred)
print(f' accuracy: {accuracy:.2f} \n recall: {recall:.2f} \n precision: {precision:.2f} \n f1_score: {f1score:.2f}' )
```

```
 accuracy: 0.95
 recall: 0.99
 precision: 0.92
 f1_score: 0.95
```

By using 'cross_val_score' we can cross validate our training samples with k-fold, we have used cv=10 means it will randomize the sample in 10 different groups. by taking the average of that 10 values we can get average cross validation accuracy score estimation.
Here we achieved the 96.20% average accuracy with polynomial kernel and C=10.

```python
svc = svm.SVC(kernel='poly', C=10)
scores = cross_val_score(svc, X, y, cv=10)
# print(scores)
print(scores.mean())
```

```
0.9620735512250015
```

Here we achieved the 96.79% average accuracy with rbf kernel, gamma=1.2 and C=100, which almost similar to the score achieved by 'poly' kernel.

```python
svc = svm.SVC(kernel='rbf', gamma=1.2, C=10)
scores = cross_val_score(svc, X, y, cv=10)
print(scores)
print(scores.mean())
```

```
[0.97084548 0.96793003 0.96793003 0.98250729 0.95626822 0.97667638
 0.97376093 0.96501458 0.9502924  0.96783626]
0.9679061599577174
```

Assignment 3

Experiment

Here I have tried to perform cross validation without performing the data scaling.
We can observe here that with the same gamma and C value it gives us 99.85% average accuracy, which actually not good because our model performs overfitting here because of the values in all 3 columns are not on the similar scale.

```
X1 = df_upsampled_noscale.drop(['clientid', 'default'], axis=1)
y1 = df_upsampled_noscale['default']
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_size=0.2, random_state=7)
svc = svm.SVC(kernel='rbf', gamma=1.2, C=10)
scores_noscale = cross_val_score(svc, X1, y1, cv=10)
print(scores_noscale)
print(f"average svc score without scaling the data: {scores_noscale.mean()*100:.2f}")
```

```
[1.         1.         1.         0.99708455 1.         1.
 1.         0.99125364 0.99707602 1.        ]
average svc score without scaling the data: 99.85
```

Observing the average accuracy with the different C values for 'poly' kernel. By increasing the C value we are getting better accuracy but computational time increased drastically.

```
C_values = [0.1, 1, 10, 50]
for C in C_values:
    svc = svm.SVC(kernel='poly', C=C)
    scores = cross_val_score(svc, X, y, cv=10)
    print(f"For C value {C}: {scores.mean()*100:.2f}%")
```

```
For C value 0.1: 95.24%
For C value 1: 95.86%
For C value 10: 96.21%
For C value 50: 96.53%
```

For 'rbf', By increasing the C value we are getting much better result and the computation time is comparatively low as compare to the 'poly'.

```
C_values = [0.1, 1, 10, 100]
for C in C_values:
    svc = svm.SVC(kernel='rbf', C=C)
    scores = cross_val_score(svc, X, y, cv=10)
    print(f"For C value {C}: {scores.mean()*100:.2f}%")
```

```
For C value 0.1: 95.80%
For C value 1: 96.91%
For C value 10: 98.19%
For C value 100: 98.92%
```

With different gamma values we are getting the better accaurary for higher value of gamma, which is as expected.

```
gamma_values = [0.1, 1, 10, 100]
for gamma in gamma_values:
    svc = svm.SVC(kernel='rbf', gamma=gamma)
    scores = cross_val_score(svc, X, y, cv=10)
    print(f"For gamma value {gamma}: {scores.mean()*100:.2f}%")
```

```
For gamma value 0.1: 93.23%
For gamma value 1: 95.59%
For gamma value 10: 97.29%
For gamma value 100: 98.77%
```

Note:- I have noticed that with 'poly' kernel and higher C value it takes very long in computation as compare to 'rbf'.