Classification Case study - PresCorp Personal loan

In [6]:
```
import pandas as pd
```

In [7]:
```
loan = pd.read_csv(r'C:\Users\Admin\Desktop\PresCorp - Personal Loan.csv')
loan.head()
```

Out[7]:

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	CD Account	Online	CreditCard
0	1	25	1	49	91107	4	1.6	No Degree	0	0	1	0	0	0
1	2	45	19	34	90089	3	1.5	No Degree	0	0	1	0	0	0
2	3	39	15	11	94720	1	1.0	No Degree	0	0	0	0	0	0
3	4	35	9	100	94112	1	2.7	Bacherlors	0	0	0	0	0	0
4	5	35	8	45	91330	4	1.0	Bacherlors	0	0	0	0	0	1

In [8]:
```
loan.shape
```

Out[8]:
```
(5000, 14)
```

In [9]:
```
loan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 14 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   ID                  5000 non-null   int64  
 1   Age                 5000 non-null   int64  
 2   Experience          5000 non-null   int64  
 3   Income              5000 non-null   int64  
 4   ZIP Code            5000 non-null   int64  
 5   Family              5000 non-null   int64  
 6   CCAvg               5000 non-null   float64
 7   Education           5000 non-null   object 
 8   Mortgage            5000 non-null   int64  
 9   Personal Loan       5000 non-null   int64  
 10  Securities Account  5000 non-null   int64  
 11  CD Account          5000 non-null   int64  
 12  Online              5000 non-null   int64  
 13  CreditCard          5000 non-null   int64  
dtypes: float64(1), int64(12), object(1)
memory usage: 547.0+ KB
```

In [10]:
```
loan.isnull().sum()
```

Out[10]:
```
ID                    0
Age                   0
Experience            0
Income                0
ZIP Code              0
Family                0
CCAvg                 0
Education             0
Mortgage              0
Personal Loan         0
Securities Account    0
CD Account            0
Online                0
CreditCard            0
dtype: int64
```

In [9]:
```
loan['Education'].unique()
```

Out[9]:
```
array(['No Degree', 'Bacherlors', 'Masters'], dtype=object)
```

Since Education column is categorical variable, we need to convert it into numeric column, so that we can build the model. This process is called as creating dummy variables. This can be acheived through pd.get_dummies

In [11]:
```
# Creating Dummy Variables
```

In [11]:
```
loan_dummy = pd.get_dummies(loan,columns=['Education'],drop_first=True)
```

In [12]:
```
loan_dummy.head()
```

Out[12]:

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Mortgage	Personal Loan	Securities Account	CD Account	Online	CreditCard	Education_Masters	Education_No Degree
0	1	25	1	49	91107	4	1.6	0	0	1	0	0	0	0	1
1	2	45	19	34	90089	3	1.5	0	0	1	0	0	0	0	1
2	3	39	15	11	94720	1	1.0	0	0	0	0	0	0	0	1
3	4	35	9	100	94112	1	2.7	0	0	0	0	0	0	0	0
4	5	35	8	45	91330	4	1.0	0	0	0	0	0	1	0	0

Identify the input and output features

In [13]:
```
y = loan_dummy[['Personal Loan']]
x = loan_dummy.drop(columns=['Personal Loan','ID','ZIP Code'])
```

Split the data into train and test

In [2]:
```
from sklearn.model_selection import train_test_split
```

In [14]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=42)
```

In [18]:
```
len(x_train),len(x_test),len(y_train),len(y_test)
```

Out[18]:
```
(4000, 1000, 4000, 1000)
```

Building logistic regression

In [15]:
```
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression()
model = log_reg.fit(x_train,y_train)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example u
sing ravel().
  return f(*args, **kwargs)
C:\Users\Admin\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Predicting the model on test data

In [16]:
```
y_test.head()
```

Out[16]:

	Personal Loan
1501	0
2586	1
2653	0
1055	0
705	0

In [17]:
```
y_test['Prediction_LR'] = model.predict(x_test)
```

```
C:\Users\Admin\AppData\Local\Temp/ipykernel_12224/535613758.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  y_test['Prediction_LR'] = model.predict(x_test)
```

In [18]:
```
y_test.head()
```

Out[18]:

	Personal Loan	Prediction_LR
1501	0	0
2586	1	0
2653	0	0
1055	0	0
705	0	0

Accuracy = number of correct predictions/Total value.

Syntax for accuracy and confusion matrix is

print(accuracy(actual column, predicted column))

In [19]:
```
from sklearn.metrics import accuracy_score,confusion_matrix
```

In [21]:
```
print(confusion_matrix(y_test['Personal Loan'],y_test['Prediction_LR']))
```

```
[[887   8]
 [ 28  77]]
```

In [22]:
```
print(accuracy_score(y_test['Personal Loan'],y_test['Prediction_LR']))
```

```
0.964
```

The Logistic Regression model is able to predict the data with 96.4% accuracy. This is a good value.

Desicion Tree

In [23]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
model = dt.fit(x_train,y_train)
```

In [24]:
```
y_test['Prediction_DT'] = model.predict(x_test)
```

```
C:\Users\Admin\AppData\Local\Temp/ipykernel_12224/2931032371.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  y_test['Prediction_DT'] = model.predict(x_test)
```

In [25]:
```
y_test.head()
```

Out[25]:

	Personal Loan	Prediction_LR	Prediction_DT
1501	0	0	0
2586	1	0	1
2653	0	0	0
1055	0	0	0
705	0	0	0

In [27]:
```
print(confusion_matrix(y_test['Personal Loan'],y_test['Prediction_DT']))
```

```
[[891   4]
 [  8  97]]
```

In [28]:
```
print(accuracy_score(y_test['Personal Loan'],y_test['Prediction_DT']))
```

```
0.988
```

The Decision Tree algorithm is performing better than logistic regression and it is giving an accuracy of 98.8%.

In [ ]:
```

```