Classification

Supervised Learning

- Logistic Regression

In [1]:

```
# Libraries
import numpy as np
import pandas as pd
```

In [2]:

```
# Import dataset
df = pd.read_csv('loan_prediction.csv')
df.head()
```

Out[2]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	LP001002	Male	No	0.0	Graduate	No	5849	0.0	NaN	360.0	1.0	Urban	Y
1	LP001003	Male	Yes	1.0	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural	N
2	LP001005	Male	Yes	0.0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban	Y
3	LP001006	Male	Yes	0.0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban	Y
4	LP001008	Male	No	0.0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban	Y

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            614 non-null    object 
 1   Gender             601 non-null    object 
 2   Married            611 non-null    object 
 3   Dependents         599 non-null    float64
 4   Education          614 non-null    object 
 5   Self_Employed      582 non-null    object 
 6   ApplicantIncome    614 non-null    int64  
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object 
 12  Loan_Status        614 non-null    object 
dtypes: float64(5), int64(1), object(7)
memory usage: 62.5+ KB
```

In [4]:

```
df.describe()
```

Out[4]:

	Dependents	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	599.000000	614.000000	614.000000	592.000000	600.00000	564.000000
mean	0.762938	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	1.015216	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	0.000000	150.000000	0.000000	9.000000	12.00000	0.000000
25%	0.000000	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	0.000000	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	2.000000	5795.000000	2297.250000	168.000000	360.00000	1.000000
max	3.000000	81000.000000	41667.000000	700.000000	480.00000	1.000000

Data Preprocessing

Handling missing data

In [5]:

```
df.isnull().any()
```

Out[5]:

```
Loan_ID              False
Gender                True
Married               True
Dependents            True
Education            False
Self_Employed         True
ApplicantIncome      False
CoapplicantIncome    False
LoanAmount            True
Loan_Amount_Term      True
Credit_History        True
Property_Area        False
Loan_Status          False
dtype: bool
```

Checking the count of null values in each column..

In [6]:

```
df.isnull().sum()
```

Out[6]:

```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

We need to treat the null values by identifying which is categorical and which is continuous.

- Whenever we have categorical value, then we can use mode() to replace the null value. (i.e, replacing with the most occuring null value)
- When we have continuous value, we can replace the null values with mean() or median().
- When we have high range of numerical values, like salary, median() can be used.
- When we have small range, like age, we can use mean().

In [7]:

```
df['Gender'].fillna(df['Gender'].mode()[0], inplace = True)
df['Married'].fillna(df['Married'].mode()[0], inplace = True)
df['Dependents'].fillna(df['Dependents'].mode()[0], inplace = True)
df['Self_Employed'].fillna(df['Self_Employed'].mode()[0], inplace = True)
df['LoanAmount'].fillna(df['LoanAmount'].mean(), inplace = True)
df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mean(), inplace = True)
df['Credit_History'].fillna(df['Credit_History'].mode()[0], inplace = True)
```

Let us check if there are any null values left..

In [8]:

```
df.isnull().sum()
```

Out[8]:

```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

Handling text data

There are 6 columns in text data format except Loan_ID. (As Loan_ID has no effect on the prediction, we can ignore it)

Let us now convert the text columns into numeric data by applying LabelEncoding method:

In [9]:

```
from sklearn.preprocessing import LabelEncoder
```

In [10]:

```
le = LabelEncoder()
df['Gender'] = le.fit_transform(df['Gender'].astype(str))    # Male/Female
df['Married'] = le.fit_transform(df['Married'].astype(str))    # Yes/No
df['Education'] = le.fit_transform(df['Education'].astype(str))        #  Graduate/Not Graduate
df['Self_Employed'] = le.fit_transform(df['Self_Employed'].astype(str))      # Yes/No
df['Property_Area'] = le.fit_transform(df['Property_Area'].astype(str))      # Urban/Rural/SemiUrban
df['Loan_Status'] = le.fit_transform(df['Loan_Status'].astype(str))      #Yes/No
```

In [11]:

```
df.head()
```

Out[11]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	LP001002	1	0	0.0	0	0	5849	0.0	146.412162	360.0	1.0	2	1
1	LP001003	1	1	1.0	0	0	4583	1508.0	128.000000	360.0	1.0	0	0
2	LP001005	1	1	0.0	0	1	3000	0.0	66.000000	360.0	1.0	2	1
3	LP001006	1	1	0.0	1	0	2583	2358.0	120.000000	360.0	1.0	2	1
4	LP001008	1	0	0.0	0	0	6000	0.0	141.000000	360.0	1.0	2	1

There is no textual column now. All the columns are in numeric format Except Loan_ID.

Now, let us split the data into dependent and independent variables:

In [12]:

```
x = df.drop(columns=['Loan_ID', 'Loan_Status']).values
x
```

Out[12]:

```
array([[  1.,   0.,   0., ..., 360.,   1.,   2.],
       [  1.,   1.,   1., ..., 360.,   1.,   0.],
       [  1.,   1.,   0., ..., 360.,   1.,   2.],
       ...,
       [  1.,   1.,   1., ..., 360.,   1.,   2.],
       [  1.,   1.,   2., ..., 360.,   1.,   2.],
       [  0.,   0.,   0., ..., 360.,   0.,   1.]])
```

In [13]:

```
x.shape
```

Out[13]:

```
(614, 11)
```

In [14]:

```
y = df['Loan_Status'].values
y
```

Out[14]:

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

In [15]:

```
y.shape
```

Out[15]:

```
(614,)
```

Out of all the columns converted from string to numeric, Property_Area has three category of values. All the others have 2, which can be represented in binary. Hence, we can apply OneHotEncoder:

In [16]:

```
from sklearn.preprocessing import OneHotEncoder
```

In [17]:

```
one = OneHotEncoder()
z = one.fit_transform(x[:,10:11]).toarray()
z
```

Out[17]:

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

In [18]:

```
# Deleting Property_Area Column
x = np.delete(x, 10, axis = 1)
```

In [19]:

```
# Ading the three newly created column using concatinate function
x = np.concatenate((z, x), axis = 1)
```

In [20]:

```
x.shape
```

Out[20]:

```
(614, 13)
```

Train, Test and Fit data

In [21]:

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

In [22]:

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)
```

In [23]:

```
x_train.shape
```

Out[23]:

```
(491, 13)
```

In [24]:

```
x_test.shape
```

Out[24]:

```
(123, 13)
```

In [25]:

```
x
```

Out[25]:

```
array([[  0.        ,   0.        ,   1.        , ..., 146.41216216,
        360.        ,   1.        ],
       [  1.        ,   0.        ,   0.        , ..., 128.        ,
        360.        ,   1.        ],
       [  0.        ,   0.        ,   1.        , ...,  66.        ,
        360.        ,   1.        ],
       ...,
       [  0.        ,   0.        ,   1.        , ..., 253.        ,
        360.        ,   1.        ],
       [  0.        ,   0.        ,   1.        , ..., 187.        ,
        360.        ,   1.        ],
       [  0.        ,   1.        ,   0.        , ..., 133.        ,
        360.        ,   0.        ]])
```

Now, let us scale the data with standard scaler..

In [26]:

```
from sklearn.preprocessing import StandardScaler
```

In [27]:

```
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.fit_transform(x_test)
```

Model Building

In [28]:

```
from sklearn.linear_model import LogisticRegression
```

In [29]:

```
log = LogisticRegression()
log.fit(x_train,y_train)
```

Out[29]:

```
LogisticRegression()
```

Prediction

In [30]:

```
logprediction = log.predict(x_test)
logprediction
```

Out[30]:

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

In [31]:

```
y_test
```

Out[31]:

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

Evaluating the performance

In [32]:

```
from sklearn.metrics import accuracy_score
```

In [33]:

```
logacc = accuracy_score(y_test, logprediction)
logacc
```

Out[33]:

```
0.8373983739837398
```

Using one more evaluation metric: confusion_matrix

In [34]:

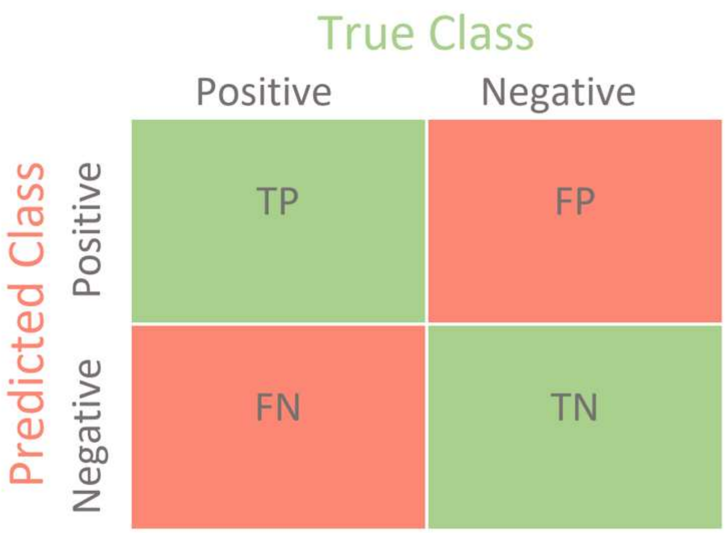
```
from sklearn.metrics import confusion_matrix
```

In [35]:

```
logcm = confusion_matrix(logprediction, y_test)
logcm
```

Out[35]:

```
array([[15,  2],
       [18, 88]], dtype=int64)
```



Decision Tree Algorithm

Model Building with Decision Tree Classifier

In [36]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [37]:

```
dtc = DecisionTreeClassifier(criterion='entropy')
dtc.fit(x_train, y_train)
```

Out[37]:

```
DecisionTreeClassifier(criterion='entropy')
```

Prediction

In [38]:

```
dtcprediction = dtc.predict(x_test)
dtcprediction
```

Out[38]:

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

In [39]:

```
y_test
```

Out[39]:

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

Accuracy Score

In [40]:

```
dtcacc = accuracy_score(dtcprediction, y_test)
dtcacc
```

Out[40]:

```
0.6829268292682927
```

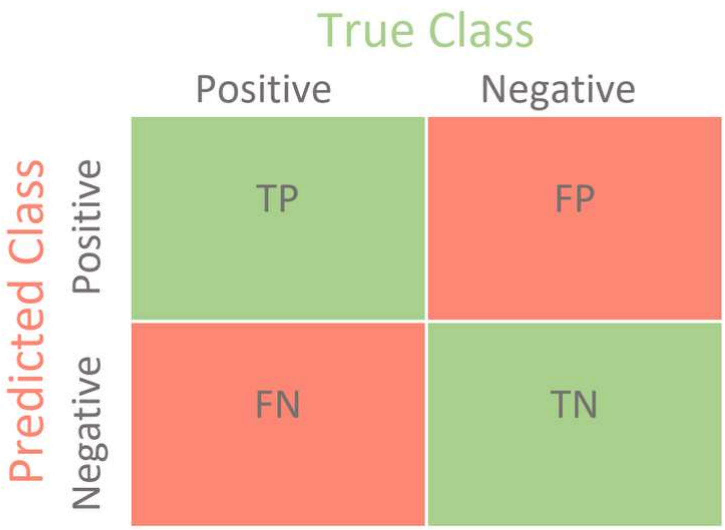
Confusion matrix

In [41]:

```
dtccm = confusion_matrix(dtcprediction, y_test)
dtccm
```

Out[41]:

```
array([[20, 26],
       [13, 64]], dtype=int64)
```



Randon Forest Algorithm

Model Building using Random Forest Classifier

In [42]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [43]:

```
rfc = RandomForestClassifier(criterion = 'entropy')
rfc.fit(x_train, y_train)
```

Out[43]:

```
RandomForestClassifier(criterion='entropy')
```

Prediction

In [44]:

```
rfc_predict = rfc.predict(x_test)
rfc_predict
```

Out[44]:

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

In [45]:

```
y_test
```

Out[45]:

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

Evaluating accuracy

In [46]:

```
rfcacc = accuracy_score(y_test,rfc_predict)
rfcacc
```

Out[46]:

```
0.7967479674796748
```

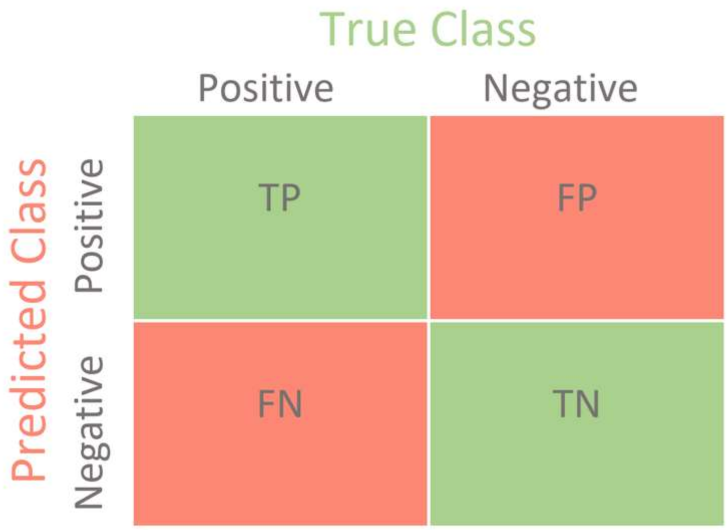
Confusion Matrix

In [47]:

```
rfccm = confusion_matrix(y_test,rfc_predict)
rfccm
```

Out[47]:

```
array([[17, 16],
       [ 9, 81]], dtype=int64)
```



In [ ]: