Assignment 2 : Credit card fraud detection

The credit card fraud detection assignment we discussed in our previous meeting. The assignment focuses on conducting a detailed exploratory data analysis (EDA) and modelling using the "Credit Card Fraud Detection" dataset available on Kaggle.

Assignment Details:

Task: Perform an in-depth EDA and modeling on the credit card fraud dataset.

Deadline: Sunday, 5 pm.

Evaluation: Your performance will be assessed during the Sunday evening meeting, where we will discuss the results and plan for the upcoming week.

Expectations:

I expect you to invest significant effort into this assignment, as it will demonstrate your analytical skills and provide practical experience in fraud detection analysis.

While the end result is important, I value your effort and commitment equally. Concentrate on thoroughly exploring the dataset, extracting meaningful insights, and documenting your analysis process.

Additionally, I want to offer you three important tips to handle the fraud detection problem in machine learning effectively:

Imbalanced Dataset Handling: Address the imbalance between fraudulent and legitimate transactions in the dataset. Techniques like oversampling, undersampling, or using algorithms like SMOTE (Synthetic Minority Over-sampling Technique) can help balance the dataset and avoid biased model performance.

Feature Engineering: Improve your fraud detection model by identifying and creating relevant features from the existing dataset. Explore techniques such as aggregating transaction data, creating time-based features, or deriving statistical measures from transaction amounts to capture meaningful patterns and anomalies.

Model Evaluation Metrics: Use appropriate evaluation metrics for fraud detection, considering that standard classification metrics like accuracy might not be sufficient. Metrics such as precision, recall, F1-score, and area under the ROC curve (AUC-ROC) provide better insights into your model's performance in correctly identifying fraud while minimizing false positives.

Remember to submit a comprehensive email documenting your observations, findings, and any challenges you faced during the analysis as your assignment submission.

If you have any questions or concerns about the assignment or the provided tips, please feel free to reach out to me. I am here to assist you and provide guidance throughout the process.

Best regards,

Anuj.

Import necessory libraries :

In [37]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

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

Import Dataset :

In [2]:
```
credit_df = pd.read_csv('creditcard.csv')
```

In [3]:
```
credit_df.head()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0

5 rows × 31 columns

In [4]:
```
credit_df.tail()
```

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	V7	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0

5 rows × 31 columns

From Dataframe nothing is clear only **Time**, **Amount**, **Class** features understand.

In [5]:
```
credit_df.shape
```

Out[5]:
```
(284807, 31)
```

This Data is having **284807 rows & 31 columns** data is large.

In [6]:
```
credit_df.columns
```

Out[6]:
```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

Time : the format of this column is doesn't understand it is in float datatype.

Amount : this is transaction amount in US dollars.

Class : whether the transaction is fraud or not. Here **0** ---> not fraud transaction and **1** --->fraud transaction

In [7]:
```
credit_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

In [8]:
```
credit_df.isnull().sum()
```

Out[8]:
```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

In [9]:
```
credit_df.describe()
```

Out[9]:

	Time	V1	V2	V3	V4	V5
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
mean	94813.859575	3.918649e-15	5.682686e-16	-8.761736e-15	2.811118e-15	-1.552103e-15
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01

8 rows × 31 columns

In [10]:
```
# minutes = seconds // 60

minutes  = 172792 // 60
hrs = 2879 // 60
days = 47 / 24
```

Looking at **Time** feature , Time is in second.

minutes = seconds // 60

minutes = 172792 // 60 hrs = 2879 // 60 days = 47 / 24 == 2 days

means this data is 2 days data.

Univariate Analysis

In [11]:
```
credit_df['Class'].nunique()
```

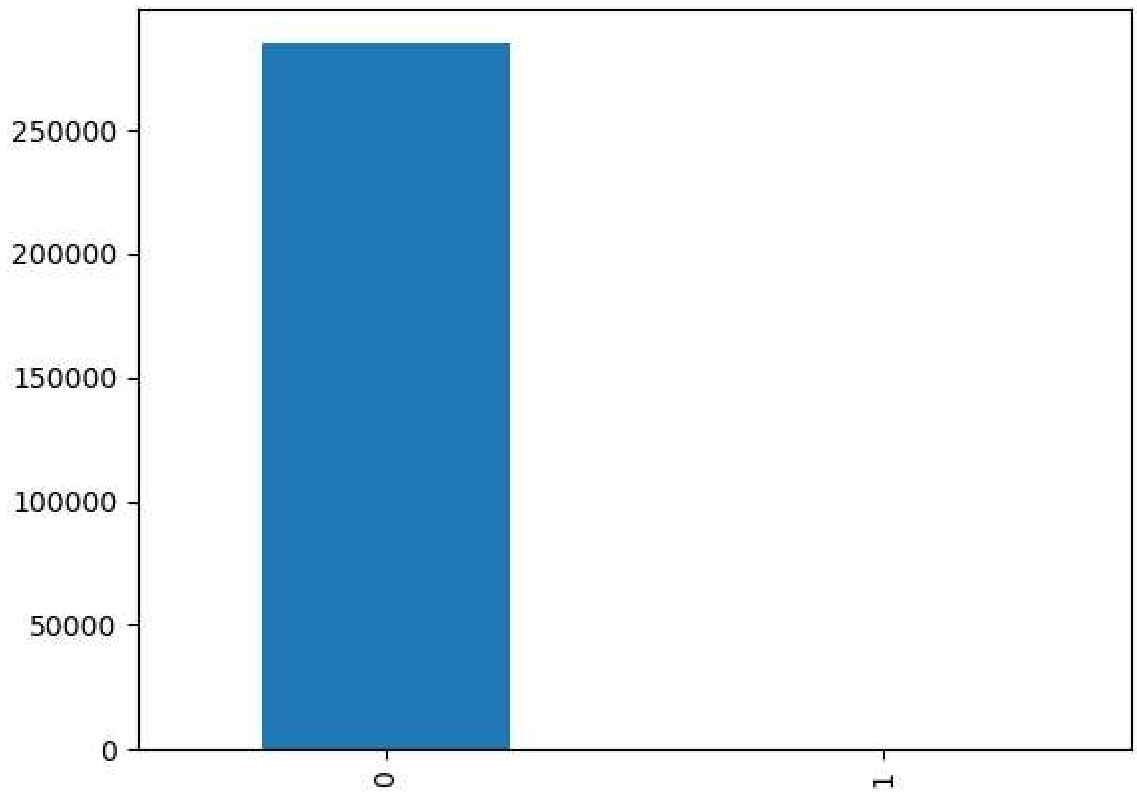
Out[11]:
```
2
```

In [12]:
```
credit_df['Class'].unique()
```

Out[12]:
```
array([0, 1], dtype=int64)
```

In [13]:
```
credit_df['Class'].value_counts().plot(kind = 'bar')
```

Out[13]: <AxesSubplot:>



From above graph we clearly understand that this data is highly **Imbalance Data**.

In [14]:
```
Non_Fraud_transaction = credit_df[credit_df['Class'] == 0]
```

In [15]:
```
Fraud_transaction = credit_df[credit_df['Class'] == 1]
```

In [16]:
```
Non_Fraud_transaction.shape
```

Out[16]: (284315, 31)

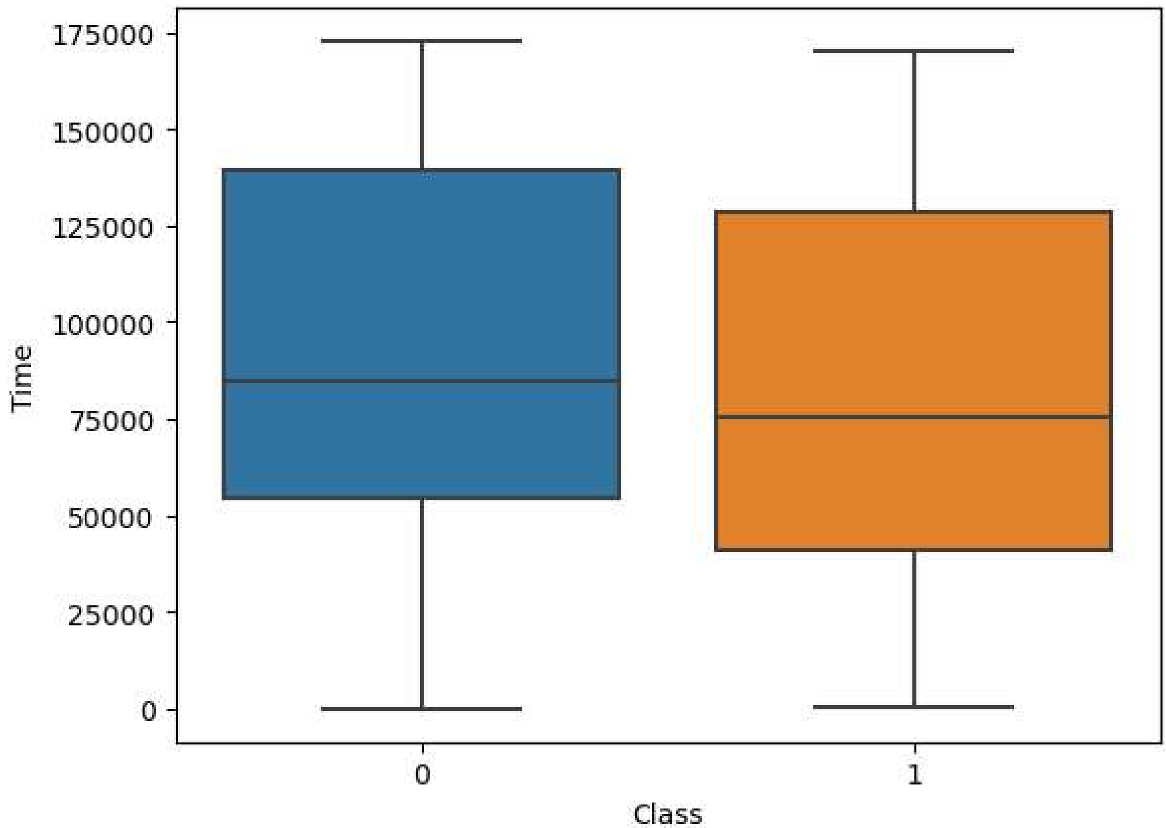
In [17]:
```
Fraud_transaction.shape
```

Out[17]: (492, 31)

Here, Total transactions are **284807** in that **284315** are Not Fraud transactions **492** are Fraud transactions.

In [18]:
```
sns.boxplot(x = 'Class',y = 'Time',data = credit_df)
```

Out[18]:
```
<AxesSubplot:xlabel='Class', ylabel='Time'>
```



From above fig, we can say that after 125000 **(2083 minutes ,34 hrs,almost 1 day)** seconds there is no Fraud.

In [19]:
```
Non_Fraud_transaction['Amount'].describe()
```

Out[19]:
```
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
```

Findings : From normal transctions **mean** transction amount is **88 USD** , From normal transctions **minimum** transction amount is **0 USD** and **maximum** transction amount is **25691 USD**

In [20]:
```
Fraud_transaction['Amount'].describe()
```

Out[20]:
```
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
```

Findings : From Fraud transctions mean transction amount is **122 USD** , From Fraud transctions minimum transction amount is **0 USD** and maximum transction amount is **2125 USD**

In [21]:
```
credit_df.groupby('Class').mean()
```

Out[21]:

Class	Time	V1	V2	V3	V4	V5	V6	V7
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731

2 rows × 30 columns

Findings : From above function, see the difference between mean of Class 0 (Non_Fraud_transaction) and Class 1 (Fraud_transaction) is very High,

```
eg: Class 0 - V1 = 0.008258 & Class 1 - V1 = -4.771948;
    Class 0 - V2 = -0.006271 & Class 1 - V2 = 3.623778;
    Class 0 - V3 = 0.012171 & Class 1 - V3 = -7.033281;.... so on
```

it is easy to our model to differ class.

Handle Imbalance data

To handle Imbalance data there are two sampling techniques :

1. Under Sampling
2. Over Sampling

we are going to use Under Sampling because we have 284807 data in that 492 is in class 1 if we do Under sampling there will not load on our model,as we go for Over Sampling 284315 will add in this dataset and hard to model to predict due to huge data.

Now for Under Sampling we need to create a sample dataset from Non fraud tranctions which is random in nature.

In [22]:
```
Non_Fraud_transaction_sample = Non_Fraud_transaction.sample(n = 492)
```

Now we need to concate Non_Fraud_transaction_sample and Fraud_transaction so we are ready to built model

In [23]:
```
New_dataSet = pd.concat([Non_Fraud_transaction_sample,Fraud_transaction], axis
```

In [24]:
```
New_dataSet.head()
```

Out[24]:

	Time	V1	V2	V3	V4	V5	V6	V7	
214897	139861.0	-0.388608	0.526256	0.083710	-0.182225	0.650576	-1.227485	0.783434	-0.4
69199	53313.0	1.273311	0.385348	-0.044679	0.432672	0.349779	-0.088898	0.086319	-0.0
244666	152440.0	1.737297	-0.677723	0.329980	1.449002	-1.150426	-0.007702	-0.929469	0.1
264575	161491.0	-2.669965	3.182635	-2.226723	-1.370713	0.446672	0.707433	-1.712728	-7.9
128889	78877.0	-1.074384	1.279473	1.092473	-0.474557	-0.035360	-0.622835	0.385803	0.4

5 rows × 31 columns

In [25]:
```
New_dataSet.tail()
```

Out[25]:

	Time	V1	V2	V3	V4	V5	V6	V7	
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850	0.69
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170	0.24
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739	1.21
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002	1.05
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050	-0.06

5 rows × 31 columns

In [26]:
```
New_dataSet.shape
```

Out[26]:
```
(984, 31)
```

In [27]:
```
New_dataSet['Class'].value_counts()
```

Out[27]:
```
0    492
1    492
Name: Class, dtype: int64
```

Now will check mean of new dataset on class or target feature.

In [28]:
```
New_dataSet.groupby('Class').mean()
```

Out[28]:

Class	Time	V1	V2	V3	V4	V5	V6	V7
0	95303.975610	-0.096418	0.091988	-0.014116	-0.117817	0.094483	-0.061954	-0.019011
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731

2 rows × 30 columns

Again difference is high so model can differciate easily class of each transaction.

Let's split Train and Test data :

In [29]:
```
X = New_dataSet.drop(columns = 'Class',axis = 1) # Drop Target Variable ,axis
Y = New_dataSet['Class'] # Only Target Variable
```

In [30]:
```
X.shape
```

Out[30]:
```
(984, 30)
```

In [31]:
```
Y.shape
```

Out[31]:
```
(984,)
```

In [32]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, strat
```

In [33]:
```
print('X',X.shape,'X_train',X_train.shape,'X_test',X_test.shape)
```

```
X (984, 30) X_train (787, 30) X_test (197, 30)
```

In [34]:
```
print('Y',Y.shape,'Y_train',Y_train.shape,'Y_test',Y_test.shape)
```

```
Y (984,) Y_train (787,) Y_test (197,)
```

LOGISTIC REGRESSION ModeL TraiNing :

We will Build Logistic Regression classification Model :

WHEN TO USE LOGISTIC REGRESSION ? Logistic Regression is used for classification problems when the output or dependent variable is categorical. Logistic regression is used when your Y variable can take only two values. Logistic regression is used when data is small in quantity.

thats why we are going for **Logistic Regression**.

In [35]:
```
model = LogisticRegression()
model.fit(X_train,Y_train)  # submit Training data to model
```

Out[35]:
```
LogisticRegression()
```

ModeL Evaluation :

Accuracy Score:

In [39]:
```
X_train_Predict = model.predict(X_train)

train_accuracy = accuracy_score(X_train_Predict,Y_train)
print('Training Data Accuracy : ',train_accuracy)

X_test_predict = model.predict(X_test)

test_accuracy = accuracy_score(X_test_predict,Y_test)
print('Testing Data Accuracy : ',test_accuracy)

print("Precision : ",metrics.precision_score(Y_test, X_test_predict))
print("Recall : ",metrics.recall_score(Y_test, X_test_predict))
```

```
Training Data Accuracy :  0.9415501905972046
Testing Data Accuracy :  0.9187817258883249
Precision :  0.9456521739130435
Recall :  0.8877551020408163
```

See Here, Accuracy of model is 94% on train data and for testing data accuracy is 92%. Well, we got a precision of 94% and recall of 88%, which means high precision but low recall , returning very few results, but most of its predicted labels are correct when compared to the training labels.

SVM ModeL TraiNing :

We will Build SVM classification Model :

WHEN TO USE SVM ?

1. SVM is Effective on datasets with multiple features, like financial or medical data.
2. Different kernel functions can be specified for the decision function. You can use common kernels, but it's also possible to specify custom kernels.
3. we use SVMs because It can handle both classification and regression on linear and non-linear data.

In [ ]:
```
from sklearn import svm  #Import svm model
from sklearn import metrics

clf = svm.SVC(kernel='linear') #Create a svm Classifier # Linear Kernel

clf.fit(X_train, Y_train) # submit Training data to model


X_pred = clf.predict(X_train)
SVM_train_accuracy = accuracy_score(X_pred,Y_train)
print('Training Data SVM Accuracy : ',SVM_train_accuracy)

Y_pred = clf.predict(X_test)
SVM_test_accuracy = accuracy_score(Y_pred,Y_test)
print('Testing Data SVM Accuracy : ',SVM_test_accuracy)

print("Precision : ",metrics.precision_score(Y_test, Y_pred))
print("Recall : ",metrics.recall_score(Y_test, Y_pred))
```

we have used different kernels in SVM to check accuracy difference ,

For LINEAR : Training Data SVM Accuracy : 0.8983481575603558 Testing Data SVM Accuracy : 0.8934010152284264 Precision : 0.9873417721518988 Recall : 0.7959183673469388

For POLINOMINAL : Training Data SVM Accuracy : 0.567979669631512 Testing Data SVM Accuracy : 0.5228426395939086 Precision : 0.5140845070422535 Recall : 0.7448979591836735

For SIGMOID : Training Data SVM Accuracy : 0.4777636594663278 Testing Data SVM Accuracy : 0.48223350253807107 Precision : 0.47619047619047616 Recall : 0.40816326530612246

for RBF : Training Data SVM Accuracy : 0.5527318932655655 Testing Data SVM Accuracy : 0.5228426395939086 Precision : 0.5178571428571429 Recall : 0.5918367346938775

From above kernel Linear is giving good accuracy with 89%.

See Here, Accuracy of model is 90% on train data and for testing data accuracy is 90% means same for both data. Well, we got a precision of 98% and recall of 79%, which are considered as very good values.

Decission Tree ModeL TraiNing :

We will Build Decission Tree classification Model :

WHEN TO USE Decission Tree ?

The majority of decision trees in machine learning will be used for classification problems, to categorise objects against learned features.

In [40]:
```
from sklearn import tree  #Import Decision Tree model

clf = tree.DecisionTreeClassifier() #Create a Decision Tree Classifier

clf = clf.fit(X_train, Y_train) # submit Training data to model


X_pred = clf.predict(X_train)
DT_train_accuracy = accuracy_score(X_pred,Y_train)
print('Training Data Decision Tree Accuracy : ',DT_train_accuracy)

Y_pred = clf.predict(X_test)
DT_test_accuracy = accuracy_score(Y_pred,Y_test)
print('Testing Data Decision Tree Accuracy : ',DT_test_accuracy)


print("Precision : ",metrics.precision_score(Y_test, Y_pred))
print("Recall : ",metrics.recall_score(Y_test, Y_pred))
```

```
Training Data Decision Tree Accuracy :  1.0
Testing Data Decision Tree Accuracy :  0.9137055837563451
Precision :  0.9550561797752809
Recall :  0.8673469387755102
```

See Here, Accuracy of model is 100% on train data and for testing data accuracy is 82%. Well, we got a precision of 85% and recall of 84%, which are considered as very good values.

Random Forest ModeL TraiNing :

We will Build Random Forest classification Model :

WHEN TO USE RF algorithm ?

Random forest is a Supervised Machine Learning Algorithm that is used widely in Classification and Regression problems.

In [41]:
```
from sklearn.ensemble import RandomForestClassifier  #Import Random Forest mod

clf = RandomForestClassifier(max_depth=2, random_state=0) #Create a Random For

clf = clf.fit(X_train, Y_train) # submit Training data to model


X_pred = clf.predict(X_train)
RF_train_accuracy = accuracy_score(X_pred,Y_train)
print('Training Data Random Forest Accuracy : ',RF_train_accuracy)

Y_pred = clf.predict(X_test)
RF_test_accuracy = accuracy_score(Y_pred,Y_test)
print('Testing Data Random Forest Accuracy : ',RF_test_accuracy)


print("Precision : ",metrics.precision_score(Y_test, Y_pred))
print("Recall : ",metrics.recall_score(Y_test, Y_pred))
```

```
Training Data Random Forest Accuracy :  0.9313850063532402
Testing Data Random Forest Accuracy :  0.9086294416243654
Precision :  0.9878048780487805
Recall :  0.826530612244898
```

See Here, Accuracy of model is 93% on train data and for testing data accuracy is 90%. Well, we got a precision of 100% and recall of 81%, which recall is bit low as compare to precision.