```
#Importing required libraries
import numpy as np
import pandas as pd
```

```
df=pd.read_csv('ctg_data.csv')
df
```

	b	e	AC	FM	UC	DL	DS	DP	DR	LB	...	C	D	E	AD	DE	LD	FS	SUSP
0	240	357	0	0	0	0	0	0	0	120	...	-1	-1	-1	-1	-1	-1	1	-1
1	5	632	4	0	4	2	0	0	0	132	...	-1	-1	-1	1	-1	-1	-1	-1
2	177	779	2	0	5	2	0	0	0	133	...	-1	-1	-1	1	-1	-1	-1	-1
3	411	1192	2	0	6	2	0	0	0	134	...	-1	-1	-1	1	-1	-1	-1	-1
4	533	1147	4	0	5	0	0	0	0	132	...	-1	-1	-1	-1	-1	-1	-1	-1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2121	2059	2867	0	0	6	0	0	0	0	140	...	-1	-1	1	-1	-1	-1	-1	-1
2122	1576	2867	1	0	9	0	0	0	0	140	...	-1	-1	1	-1	-1	-1	-1	-1
2123	1576	2596	1	0	7	0	0	0	0	140	...	-1	-1	1	-1	-1	-1	-1	-1
2124	1576	3049	1	0	9	0	0	0	0	140	...	-1	-1	1	-1	-1	-1	-1	-1
2125	2796	3415	1	1	5	0	0	0	0	142	...	-1	-1	-1	-1	-1	-1	-1	-1

2126 rows × 42 columns

```
df.dtypes
```

```
b            int64
e            int64
AC           int64
FM           int64
UC           int64
DL           int64
DS           int64
DP           int64
DR           int64
LB           int64
AC.1       float64
FM.1       float64
UC.1       float64
DL.1       float64
DS.1       float64
DP.1       float64
ASTV         int64
MSTV       float64
ALTV         int64
```

```
MLTV        float64
Width         int64
Min           int64
Max           int64
Nmax          int64
Nzeros        int64
Mode          int64
Mean          int64
Median        int64
Variance      int64
Tendency      int64
A             int64
B             int64
C             int64
D             int64
E             int64
AD            int64
DE            int64
LD            int64
FS            int64
SUSP          int64
CLASS         int64
NSP           int64
dtype: object
```

```
df.isna().sum()
```

```
b           0
e           0
AC          0
FM          0
UC          0
DL          0
DS          0
DP          0
DR          0
LB          0
AC.1        0
FM.1        0
UC.1        0
DL.1        0
DS.1        0
DP.1        0
ASTV        0
MSTV        0
ALTV        0
MLTV        0
Width       0
Min         0
Max         0
Nmax        0
Nzeros      0
Mode        0
Mean        0
Median      0
```

```
Variance    0
Tendency    0
A           0
B           0
C           0
D           0
E           0
AD          0
DE          0
LD          0
FS          0
SUSP        0
CLASS       0
NSP         0
dtype: int64
```

```
df.dropna()
```

	b	e	AC	FM	UC	DL	DS	DP	DR	LB	...	C	D	E	AD	DE	LD	FS	SUSP
0	240	357	0	0	0	0	0	0	0	120	...	-1	-1	-1	-1	-1	-1	1	-1
1	5	632	4	0	4	2	0	0	0	132	...	-1	-1	-1	1	-1	-1	-1	-1
2	177	779	2	0	5	2	0	0	0	133	...	-1	-1	-1	1	-1	-1	-1	-1
3	411	1192	2	0	6	2	0	0	0	134	...	-1	-1	-1	1	-1	-1	-1	-1
4	533	1147	4	0	5	0	0	0	0	132	...	-1	-1	-1	-1	-1	-1	-1	-1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2121	2059	2867	0	0	6	0	0	0	0	140	...	-1	-1	1	-1	-1	-1	-1	-1
2122	1576	2867	1	0	9	0	0	0	0	140	...	-1	-1	1	-1	-1	-1	-1	-1
2123	1576	2596	1	0	7	0	0	0	0	140	...	-1	-1	1	-1	-1	-1	-1	-1
2124	1576	3049	1	0	9	0	0	0	0	140	...	-1	-1	1	-1	-1	-1	-1	-1
2125	2796	3415	1	1	5	0	0	0	0	142	...	-1	-1	-1	-1	-1	-1	-1	-1

2126 rows × 42 columns

```
df.isna().sum()
```

```
b           0
e           0
AC          0
FM          0
UC          0
DL          0
DS          0
DP          0
DR          0
LB          0
```

```
AC.1        0
FM.1        0
UC.1        0
DL.1        0
DS.1        0
DP.1        0
ASTV        0
MSTV        0
ALTV        0
MLTV        0
Width       0
Min         0
Max         0
Nmax        0
Nzeros      0
Mode        0
Mean        0
Median      0
Variance    0
Tendency    0
A           0
B           0
C           0
D           0
E           0
AD          0
DE          0
LD          0
FS          0
SUSP        0
CLASS       0
NSP         0
dtype: int64
```

```
Features=df.drop('NSP', axis=1)
Label=df['NSP']
```

```
Features.shape
```

```
(2126, 41)
```

```
Features_T=Features.T
#Features_T.columns= Features_T.iloc[0]
#Features_T.columns
```

```
height, width = Features.shape
unique_classes = np.unique(Label)
unique_classes
```

```
array([1, 2, 3])
```

```
num_classes = len(unique_classes)

scatter_train = np.cov(Features_T)*(height - 1)
scatter_within = 0
```

```
for i in range(num_classes):
  class_items = np.flatnonzero(Label == unique_classes[i])
  scatter_within = scatter_within + np.cov(Features_T[class_items]) * (len(class_items)-1)

scatter_between = scatter_train - scatter_within
```

```
#Calculating Eigenvalues and Eigenvectors of the covariance matrix

eigen_values, eigen_vectors = np.linalg.eigh(np.linalg.pinv(scatter_within).dot(scatter_betwe
        #print(eig_vectors.shape)
        #pc = Features.dot(eig_vectors[:,::-1][:,:self.n_components])
```

```
#sort the eigenvalues in descending order
sorted_index = np.argsort(eigen_values)[::-1]

sorted_eigenvalue = eigen_values[sorted_index]
#similarly sort the eigenvectors
sorted_eigenvectors = eigen_vectors[:,sorted_index]
sorted_eigenvectors
```

```
array([[ 0.4732847 ,  0.24254814, -0.42273079, ..., -0.4187079 ,
        -0.2494802 , -0.47328548],
       [ 0.52515063, -0.21592378,  0.37658088, ...,  0.37297375,
         0.22209553, -0.52515194],
       [ 0.00600605, -0.0299112 ,  0.05207817, ...,  0.05190328,
         0.03087553, -0.00590249],
       ...,
       [-0.00531018,  0.34093247,  0.10848042, ..., -0.11113122,
         0.33409301, -0.00530762],
       [ 0.0030507 , -0.31402587, -0.08722904, ...,  0.09110984,
        -0.31046093,  0.00304867],
       [-0.0026597 ,  0.08939292,  0.03727651, ..., -0.03699031,
         0.08649941, -0.00265857]])
```

```
# select the first n eigenvectors, n is desired dimension
# of our final reduced data.

n_components = 30 #you can select any number of components.
eigenvector_subset = sorted_eigenvectors[:,0:n_components]
eigenvector_subset
```

```
array([[ 4.73284698e-01,  2.42548139e-01, -4.22730788e-01, ...,
         7.63688917e-10,  1.75092481e-04,  1.37595091e-08],
       [ 5.25150629e-01, -2.15923780e-01,  3.76580879e-01, ...,
```

```
        -6.85384611e-10, -1.59906798e-04, -1.25723302e-08],
       [ 6.00604869e-03, -2.99112002e-02,  5.20781660e-02, ...,
        -1.05252713e-09, -7.82502996e-05, -5.56236845e-09],
       ...,
       [-5.31017999e-03,  3.40932473e-01,  1.08480421e-01, ...,
        -1.00779962e-04,  2.07809477e-08, -5.76950176e-04],
       [ 3.05070046e-03, -3.14025872e-01, -8.72290427e-02, ...,
        -1.01895076e-04,  2.00832837e-08, -5.39644962e-04],
       [-2.65969668e-03,  8.93929215e-02,  3.72765087e-02, ...,
         3.94599154e-05, -7.25419056e-09,  1.90989064e-04]])
```

```
#Transform the data
Features_reduced = np.dot(eigenvector_subset.transpose(),Features.transpose()).transpose()
Features_reduced
```

```
array([[ 3.00219816e+02, -6.59347495e+01,  1.70109935e+01, ...,
         9.66392784e+01,  9.93696595e-02, -1.28167583e+02],
       [ 3.33399785e+02, -2.29947010e+02,  1.93619922e+02, ...,
         8.89863651e+01,  2.92565897e-02, -1.12945363e+02],
       [ 4.91985410e+02, -2.19317384e+02,  1.76595991e+02, ...,
         8.70908045e+01,  3.98547703e-02, -1.11782724e+02],
       ...,
       [ 2.10824774e+03, -2.76186452e+02,  2.68519032e+02, ...,
         8.52679620e+01, -2.74106294e-03, -1.14437377e+02],
       [ 2.34614499e+03, -3.72528193e+02,  4.39990898e+02, ...,
         8.63243304e+01, -7.52939940e-02, -1.13344794e+02],
       [ 3.11573744e+03, -1.55130642e+02,  6.22744741e+01, ...,
         7.48092146e+01,  7.44569707e-02, -1.07310232e+02]])
```

```
LDA_df = pd.DataFrame(Features_reduced)
LDA_df
```

	0	1	2	3	4	5	6

Split Data into Training/Testing

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

```
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(LDA_df, Label, test_size=0.3, random_stat
```

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred_train = clf.predict(X_train)
```

```
print("Training-set accuracy (in %):",metrics.accuracy_score(y_train, y_pred_train)*100)
```

```
Training-set accuracy (in %): 99.93279569892472
```

```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier(criterion="entropy", max_depth=8)

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

# Model Accuracy, how often is the classifier correct?
print("Accuracy (in %):",metrics.accuracy_score(y_test, y_pred)*100)
```

```
Accuracy (in %): 96.08150470219435
```

Naive Bayes

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, y_train)

y_pred_train = gnb.predict(X_train)

print('Training-set accuracy (in %):', metrics.accuracy_score(y_train, y_pred_train)*100)
```

```
Training-set accuracy (in %): 84.07258064516128
```

```
# making predictions on the testing set
y_pred = gnb.predict(X_test)

print("Gaussian Naive Bayes model accuracy (in %):", metrics.accuracy_score(y_test, y_pred)*1
```

```
Gaussian Naive Bayes model accuracy (in %): 81.50470219435736
```

Random Forest

```
# importing random forest classifier from assemble module
from sklearn.ensemble import RandomForestClassifier
```

```
# creating a RF classifier
rfcl = RandomForestClassifier(n_estimators = 100)

# Training the model on the training dataset
rfcl.fit(X_train, y_train)

y_pred_train = rfcl.predict(X_train)

print('Training-set accuracy (in %):', metrics.accuracy_score(y_train, y_pred_train)*100)
```

```
Training-set accuracy (in %): 99.93279569892472
```

```
# performing predictions on the test dataset
y_pred = rfcl.predict(X_test)

print('Training-set accuracy (in %):', metrics.accuracy_score(y_test, y_pred)*100)
```

```
Training-set accuracy (in %): 96.23824451410658
```

SVM

```
#Import svm model
from sklearn import svm

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

#Train the model using the training sets
svmclf.fit(X_train, y_train)

y_pred_train = svmclf.predict(X_train)

print('Training-set accuracy (in %):', metrics.accuracy_score(y_train, y_pred_train)*100)
```

```
Training-set accuracy (in %): 98.18548387096774
```

```
#Predict the response for test dataset
y_pred = svmclf.predict(X_test)

# using metrics module for accuracy calculation
print("SVM model accuracy (in %): ", metrics.accuracy_score(y_test, y_pred)*100)
```

```
SVM model accuracy (in %):  96.08150470219435
```