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

```
ctg_df=pd.read_csv("ctg_data1.csv")
```

```
ctg_df.head()
```

	b	e	AC	FM	UC	DL	DS	DP	DR	LB	...	C	D	E	AD	DE	LD	FS	SUSP	CLASS	NSP
0	240	357	0	0	0	0	0	0	0	120	...	-1	-1	-1	-1	-1	-1	1	-1	9	2
1	5	632	4	0	4	2	0	0	0	132	...	-1	-1	-1	1	-1	-1	-1	-1	6	1
2	177	779	2	0	5	2	0	0	0	133	...	-1	-1	-1	1	-1	-1	-1	-1	6	1
3	411	1192	2	0	6	2	0	0	0	134	...	-1	-1	-1	1	-1	-1	-1	-1	6	1
4	533	1147	4	0	5	0	0	0	0	132	...	-1	-1	-1	-1	-1	-1	-1	-1	2	1

5 rows × 42 columns

```
ctg_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
```

```
#Checking for null values
ctg_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
```

```
ctg_df.dropna()
```

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

2126 rows × 42 columns

```
ctg_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=ctg_df.drop('NSP', axis=1)
Label=ctg_df['NSP']
```

PCA

```
# mean Centering the data
Features_meaned = Features - np.mean(Features , axis = 0)
Features_meaned
```

	b	e	AC	FM	UC	DL	DS	DP
0	-638.439793	-1345.877234	-2.722484	-7.241298	-3.659925	-1.570085	-0.003293	-0.126058
1	-873.439793	-1070.877234	1.277516	-7.241298	0.340075	0.429915	-0.003293	-0.126058
2	-701.439793	-923.877234	-0.722484	-7.241298	1.340075	0.429915	-0.003293	-0.126058
3	-467.439793	-510.877234	-0.722484	-7.241298	2.340075	0.429915	-0.003293	-0.126058
4	-345.439793	-555.877234	1.277516	-7.241298	1.340075	-1.570085	-0.003293	-0.126058
...	...	...	...	...	...	...	...	...
2121	1180.560207	1164.122766	-2.722484	-7.241298	2.340075	-1.570085	-0.003293	-0.126058
2122	697.560207	1164.122766	-1.722484	-7.241298	5.340075	-1.570085	-0.003293	-0.126058
2123	697.560207	893.122766	-1.722484	-7.241298	3.340075	-1.570085	-0.003293	-0.126058
2124	697.560207	1346.122766	-1.722484	-7.241298	5.340075	-1.570085	-0.003293	-0.126058
2125	1917.560207	1712.122766	-1.722484	-6.241298	1.340075	-1.570085	-0.003293	-0.126058

2126 rows × 41 columns

```
# Calculate the co-variance matrix of the mean-centered data.
cov_matrix = np.cov(Features_meaned , rowvar = False)
```

```
#Calculating Eigenvalues and Eigenvectors of the covariance matrix
eigen_values , eigen_vectors = np.linalg.eigh(cov_matrix)
```

```
#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([[ 6.91839404e-01,  7.21512407e-01,  1.75656916e-02, ...,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
       [ 7.22033857e-01, -6.91484578e-01, -1.52148770e-02, ...,
         2.42108531e-16, -1.19944670e-16,  9.70778671e-16],
       [ 5.35372127e-05, -5.53091262e-03,  1.35371709e-02, ...,
         3.08198656e-12, -4.15940569e-12, -7.86838897e-11],
       ...,
       [-3.88395017e-05,  1.12232968e-04, -1.40543871e-03, ...,
        -4.51674774e-01,  1.90457665e-01, -2.39697854e-03],
       [-7.85665562e-05,  4.22719070e-05, -3.33119538e-03, ...,
        -5.54477252e-01,  1.58982377e-01, -2.16854955e-03],
       [-1.95401290e-04, -1.98217218e-05,  9.55309887e-03, ...,
         2.05604956e-01,  6.29505765e-02, -4.56858038e-04]])
```

```
# 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([[ 6.91839404e-01,  7.21512407e-01,  1.75656916e-02, ...,
        -3.54733462e-05, -3.58086842e-05, -1.13214907e-04],
       [ 7.22033857e-01, -6.91484578e-01, -1.52148770e-02, ...,
         7.41245718e-05,  5.58820374e-05,  9.34508112e-05],
       [ 5.35372127e-05, -5.53091262e-03,  1.35371709e-02, ...,
         3.87838214e-03,  1.41788419e-02, -5.82796889e-03],
       ...,
       [-3.88395017e-05,  1.12232968e-04, -1.40543871e-03, ...,
         3.68522925e-01,  3.75361381e-01, -2.92472624e-01],
       [-7.85665562e-05,  4.22719070e-05, -3.33119538e-03, ...,
         1.43153772e-02,  3.86644752e-02, -2.58845884e-01],
       [-1.95401290e-04, -1.98217218e-05,  9.55309887e-03, ...,
         3.88068380e-02,  6.43857288e-02,  4.79739804e-02]])
```

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

```
array([[-1.41343472e+03,  4.70134580e+02,  2.84244658e+01, ...,
         4.15078967e-01,  9.06630922e-01, -2.25026432e-01],
       [-1.37755124e+03,  1.08605023e+02,  5.95305431e+01, ...,
        -4.15074824e-02, -4.22331452e-02,  1.84028598e-01],
       [-1.15241176e+03,  1.31088223e+02,  6.10420703e+01, ...,
         4.02103965e-02, -1.72973592e-01,  1.41992917e-01],
       ...,
       [ 1.12739797e+03, -1.14737308e+02, -2.45558023e+01, ...,
         3.12989095e-01, -3.28107836e-03, -1.13333542e-01],
       [ 1.45447463e+03, -4.27942448e+02, -3.29994207e+01, ...,
         3.70680658e-01, -4.59279370e-02, -8.61247641e-02],
       [ 2.56282951e+03,  2.00210681e+02, -4.24806990e+01, ...,
        -5.89768074e-02, -8.03310712e-02, -2.10165288e-02]])
```

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

	0	1	2	3	4	5	6	
0	-1413.434724	470.134580	28.424466	13.879196	40.820047	-46.972007	-34.539355	-31.88

```
from sklearn.model_selection import train_test_split # Import train_test_split function
from sklearn import metrics
```

```
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(PCA_df, Label, test_size=0.3, random_state=1)
```

DECISION TREE

```
# Import Decision Tree Classifier
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("Decision Tree Model Accuracy with training data (in %):",metrics.accuracy_score(y_train, y_pred_train)*100)
```

```
Decision Tree Model Accuracy with training data (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)
```

```
print("Decision Tree model accuracy(in %):",metrics.accuracy_score(y_test, y_pred)*100)
```

```
Decision Tree model accuracy(in %): 95.7680250783699
```

Naive Bayes

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

y_pred_train = gnb.predict(X_train)

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

```
Gaussian Naive Bayes Training-set accuracy(in %): 95.83333333333334
```

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

# comparing actual response values (y_test) with predicted response values
print("Gaussian Naive Bayes model accuracy(in %):", metrics.accuracy_score(y_test, y_pred)*100)
```

```
Gaussian Naive Bayes model accuracy(in %): 95.61128526645768
```

Random Forest

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

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

# Training the model on the training dataset
rfclf.fit(X_train, y_train)
y_pred_train = rfclf.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 = rfclf.predict(X_test)

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

```
Random Forest model accuracy(in %):  98.43260188087774
```

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 %): 99.32795698924731
```

```
#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 %):  98.90282131661442
```