```
import pandas as pd
df=pd.read_csv('/content/drive/MyDrive/StudentsPerformance.csv')
```

DATA PREPROCESSING

```
df.head(5)
```

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	female	group B	bachelor's degree	standard	none	72	72	74
1	female	group C	some college	standard	completed	69	90	88
2	female	group B	master's degree	standard	none	90	95	93
			associate's					

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

```
gender                         0
race/ethnicity                 0
parental level of education    0
lunch                          0
test preparation course        0
math score                     0
reading score                  0
writing score                  0
dtype: int64
```

```
num_duplicates = df.duplicated().sum()

# Print the result
print("Number of duplicate rows in the DataFrame:", num_duplicates)
```

```
Number of duplicate rows in the DataFrame: 0
```

```
df = df.drop_duplicates()
```

```
df.duplicated().sum()
```

```
0
```

```
df.head(5)
```

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	female	group B	bachelor's degree	standard	none	72	72	74
1	female	group C	some college	standard	completed	69	90	88
2	female	group B	master's degree	standard	none	90	95	93
			associate's					

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
df['gender']=le.fit_transform(df['gender'])
df['race/ethnicity']=le.fit_transform(df['race/ethnicity'])
df['parental level of education']=le.fit_transform(df['parental level of education'])
df['lunch']=le.fit_transform(df['lunch'])
df['test preparation course']=le.fit_transform(df['test preparation course'])
```

```
df['math score']=df['math score']/100
df['reading score']=df['reading score']/100
df['writing score']=df['writing score']/100
```

```
df.head(5)
```

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	0	1	1	1	1	0.72	0.72	0.74
1	0	2	4	1	0	0.69	0.90	0.88
2	0	1	3	1	1	0.90	0.95	0.93
3	1	0	0	0	1	0.47	0.57	0.44

```
X=df.drop('test preparation course',axis=1)
Y=df['test preparation course']
```

DIMENSIONALITY REDUCTION USING PCA

```
import numpy as np
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_std =  sc.fit_transform(X)
```

```
X_std

```

```
array([[-0.96462528, -1.01504393, -0.81264039, ...,  0.39002351,
         0.19399858,  0.39149181],
       [-0.96462528, -0.15044092,  0.82795259, ...,  0.19207553,
         1.42747598,  1.31326868],
       [-0.96462528, -1.01504393,  0.28108826, ...,  1.57771141,
         1.77010859,  1.64247471],
       ...,
       [-0.96462528, -0.15044092, -0.26577606, ..., -0.46775108,
         0.12547206, -0.20107904],
       [-0.96462528,  0.71416208,  0.82795259, ...,  0.12609287,
         0.60515772,  0.58901542],
       [-0.96462528,  0.71416208,  0.82795259, ...,  0.71993682,
         1.15336989,  1.18158627]])
```

```
cov_matrix = np.cov(X_std.T)
print("cov_matrix shape:",cov_matrix.shape)
print("Covariance_matrix",cov_matrix)
```

```
cov_matrix shape: (7, 7)
Covariance_matrix [[ 1.001001   -0.00150343  0.00191499  0.02139306  0.16815039 -0.24455717
  -0.30152646]
 [-0.00150343  1.001001   -0.03197762  0.0466092   0.21663208  0.14539802
   0.16585637]
 [ 0.00191499 -0.03197762  1.001001    0.00632623 -0.06834761 -0.07251615
  -0.0843837 ]
 [ 0.02139306  0.0466092   0.00632623  1.001001    0.35122787  0.22979011
   0.24601469]
 [ 0.16815039  0.21663208 -0.06834761  0.35122787  1.001001    0.81839806
   0.80344549]
 [-0.24455717  0.14539802 -0.07251615  0.22979011  0.81839806  1.001001
   0.95555363]
 [-0.30152646  0.16585637 -0.0843837   0.24601469  0.80344549  0.95555363
   1.001001  ]]
```

```
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)
print('Eigen Vectors \n%s', eigenvectors)
print('\n Eigen Values \n%s', eigenvalues)
```

```
Eigen Vectors
%s [[ 0.11055514 -0.35256598 -0.0767744   0.84349394  0.36767125  0.08481382
  -0.06101581]
 [-0.1566525  -0.05390091  0.01272697  0.18166885 -0.2791068  -0.85866449
  -0.35239726]
 [ 0.06625722 -0.00532822 -0.01028224  0.01154301  0.19184466 -0.43232905
   0.87842464]
 [-0.23670835 -0.09296822  0.00965638  0.34256411 -0.80842483  0.2551301
   0.31502474]
 [-0.52541435  0.76415685  0.06501425  0.29940741  0.21095127  0.03799594
   0.0137216 ]
 [-0.55871156 -0.44174035  0.66103016 -0.13505269  0.18781749  0.03874318
   0.02702428]
 [-0.56089022 -0.2916337  -0.7433418  -0.17362882  0.12956205  0.02360946
```

```
   0.01744189]]

 Eigen Values
%s [2.92501837 0.08616153 0.04264133 1.17030244 0.82139857 0.95049366
 1.01099111]
```

```
eig_pairs = [(eigenvalues[index], eigenvectors[:,index]) for index in range(len(eigenvalues))]
eig_pairs.sort()
eig_pairs.reverse()
print(eig_pairs)
eigvalues_sorted = [eig_pairs[index][0] for index in range(len(eigenvalues))]
eigvectors_sorted = [eig_pairs[index][1] for index in range(len(eigenvalues))]
print('Eigenvalues in descending order: \n%s' %eigvalues_sorted)
```

```
[(2.925018365739937, array([ 0.11055514, -0.1566525 ,  0.06625722, -0.23670835, -0.52541435,
       -0.55871156, -0.56089022])), (1.1703024360164427, array([ 0.84349394,  0.18166885,  0.01154301,  0.34256411,  0.29940741,
       -0.13505269, -0.17362882])), (1.0109911080812108, array([-0.06101581, -0.35239726,  0.87842464,  0.31502474,  0.0137216 ,
        0.02702428,  0.01744189])), (0.9504936647843585, array([ 0.08481382, -0.85866449, -0.43232905,  0.2551301 ,  0.03799594,
        0.03874318,  0.02360946])), (0.8213985721885256, array([ 0.36767125, -0.2791068 ,  0.19184466, -0.80842483,  0.21095127,
        0.18781749,  0.12956205])), (0.08616152856889082, array([-0.35256598, -0.05390091, -0.00532822, -0.09296822,  0.76415685,
       -0.44174035, -0.2916337 ])), (0.0426413316276395, array([-0.0767744 ,  0.01272697, -0.01028224,  0.00965638,  0.06501425,
        0.66103016, -0.7433418 ]))]
Eigenvalues in descending order:
[2.925018365739937, 1.1703024360164427, 1.0109911080812108, 0.9504936647843585, 0.8213985721885256, 0.08616152856889082, 0.042641331627
```

```
P_reduce = np.array(eigvectors_sorted[0:3])

X_std_3D = np.dot(X_std,P_reduce.T)

reduced_pca = pd.DataFrame(X_std_3D)

reduced_pca
```

	0	1	2
0	-0.709985	-0.730694	-0.046154
1	-1.838895	-0.940584	1.136995
2	-2.843801	-0.792531	0.995314
3	2.654047	0.065997	-1.087165
4	-0.920794	1.145459	0.984070
...	...	...	...
995	-3.407797	-0.420711	0.089157
996	1.605953	0.581859	-0.713265
997	0.506822	-1.427885	-0.552756
998	-1.074004	-0.566461	0.796551
999	-1.530003	-1.281477	0.171510

1000 rows × 3 columns

SUPPORT VECTOR MACHINE

```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(reduced_pca, Y, test_size=0.4, random_state=0)

# Initialize and fit the SVM mode
model = SVC()
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Evaluate the model
print(classification_report(y_test,y_pred))

```

```
              precision    recall  f1-score   support

           0       0.61      0.40      0.49       134
           1       0.74      0.87      0.80       266

    accuracy                           0.71       400
   macro avg       0.68      0.64      0.64       400
weighted avg       0.70      0.71      0.70       400
```

✓ 0s completed at 12:27 PM