ME-793 Assignment 2: PCA

In [1]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
data = pd.read_csv("data.csv", delimiter=",")
data_arr = data.values # converting data into a numpy array
data.head()
```

Out[2]:

	length	width	thickness
0	7	4	3
1	4	1	8
2	6	3	5
3	8	6	1
4	8	5	7

(a) Write a function for determining PCs of the above dataset X. Standardize your data i.e. use zero mean and normalized data using the ”Standardize” function shown in Tutorial.

In [3]:
```
def standardize(X):
    mu = sum(X)/len(X)
    var = sum((X-mu)**2)/len(X)
    z = (X-mu)/var**0.5
    return z
```

In [4]:
```
def pca(data):
    data_norm = standardize(data)
    A = np.dot(data_norm.transpose(), data_norm) / (data_norm.shape[0]-1)
    w,v = np.linalg.eig(A)
    proj_data = np.dot(data_norm, v)
    return v.transpose(), proj_data, w
```

In [5]:
```
P, proj_data, eigvals = pca(data_arr)
```

(b) Show the principal vectors i.e. matrix P.

In [6]:
```
print(P.shape)
print("Principal vector matrix: ", P)
```

```
(3, 3)
Principal vector matrix:  [[-0.64200458 -0.68636164  0.34166917]
 [-0.66321742  0.72074503  0.20166619]
 [ 0.38467229  0.09713033  0.91792861]]
```

In [7]:
```
print(len(eigvals))
print("Eigenvalues: ", eigvals)
```

```
3
Eigenvalues:  [1.9653046  0.33794439 1.03008435]
```

(c) Show the transformed data Y.

In [8]:
```
print(proj_data.shape)
print("Projected Data: ", proj_data)
```

```
(10, 3)
Projected Data:  [[-0.54266035  0.03532425 -0.66495907]
 [ 2.80389723  0.34879208  0.06620673]
 [ 0.61563102  0.1650593  -0.30632536]
 [-2.15852616  0.38608588 -0.95883922]
 [-0.93105243  0.36001316  1.04481917]
 [ 1.1423878  -0.47124516  1.27394577]
 [ 0.80308178  0.47234173 -1.2618794 ]
 [-1.24681973 -0.0230066   1.65563789]
 [-0.28602725  0.1867989   0.02451158]
 [-0.1999119  -1.46016354 -0.87311811]]
```

In [9]:
```
print(np.cov(proj_data.T)) # covariance matrix of the projected data is a diagonal matrix!
```

```
[[ 1.96530460e+00 -1.03023311e-16 -4.30189222e-17]
 [-1.03023311e-16  3.37944386e-01  4.64517494e-17]
 [-4.30189222e-17  4.64517494e-17  1.03008435e+00]]
```

(d) Determine the variances along the principal directions.

In [10]:
```
for i, val in enumerate(eigvals):
    print("% Variance explained by PC {}: ".format(i+1), val/sum(eigvals))
```

```
% Variance explained by PC 1:  0.5895913787011411
% Variance explained by PC 2:  0.10138331566589406
% Variance explained by PC 3:  0.3090253056329648
```

(e) Determine the principal axes using Scikit learn and compare with your solution. Does your solution compare well with that from the python library functions ? Why or why not ?

In [11]:
```
from sklearn.decomposition import PCA

pca_sk = PCA(n_components = 3)
```

In [12]:
```
data_norm = standardize(data_arr)
pca_sk.fit(data_norm)
```

Out[12]:
```
PCA(n_components=3)
```

In [13]:
```
print(pca_sk.components_) # the eigenvectors (P matrix) calculated
```

```
[[-0.64200458 -0.68636164  0.34166917]
 [ 0.38467229  0.09713033  0.91792861]
 [ 0.66321742 -0.72074503 -0.20166619]]
```

In [14]:
```
print("Singular values obtained from sklearn: ", pca_sk.singular_values_)
```

```
Singular values obtained from sklearn:  [4.20567966 3.04479214 1.74398953]
```

In [15]:
```
cov_matrix = np.matmul(data_norm.transpose(), data_norm) / (data_norm.shape[0]-1)
eigvals_ = []
for eigenvector in pca_sk.components_:
    eigvals_.append(np.dot(eigenvector.T, np.dot(cov_matrix, eigenvector)))
print("Eigenvalues calculated from sklearn: ", eigvals_)
```

```
Eigenvalues calculated from sklearn:  [1.9653045956704716, 1.0300843521098824, 0.3379443855529801]
```

In [16]:
```
print(pca_sk.explained_variance_ratio_)
```

```
[0.58959138 0.30902531 0.10138332]
```

In [17]:
```
proj_data_sk = np.dot(data_norm, pca_sk.components_.T)
print(proj_data_sk.shape)
print(proj_data_sk)
```

```
(10, 3)
[[-0.54266035 -0.66495907 -0.03532425]
 [ 2.80389723  0.06620673 -0.34879208]
 [ 0.61563102 -0.30632536 -0.1650593 ]
 [-2.15852616 -0.95883922 -0.38608588]
 [-0.93105243  1.04481917 -0.36001316]
 [ 1.1423878   1.27394577  0.47124516]
 [ 0.80308178 -1.2618794  -0.47234173]
 [-1.24681973  1.65563789  0.0230066 ]
 [-0.28602725  0.02451158 -0.1867989 ]
 [-0.1999119  -0.87311811  1.46016354]]
```

In [18]:
```
print(np.cov(proj_data_sk.T)) # Covariance of the projected data matrix is diagonal!
```

```
[[ 1.96530460e+00 -7.72144097e-17  2.14378877e-16]
 [-7.72144097e-17  1.03008435e+00 -1.32620538e-16]
 [ 2.14378877e-16 -1.32620538e-16  3.37944386e-01]]
```

The results (explained variance, eigenvalues and P matrix) obtained from scikit learn library are very similar to the ones obtained from the python functions. The order of eigenvectors and hence eigenvalues obtained from the python function is not sorted according to the explained variance but the values are almost same (differing only after 8 decimal places). Also, there is one eigenvector (2nd in python function and 3rd in pca) that has differing signs. The values

(f) Determine the principal axes using numpy and compare with your solution. Does your solution compare well with that from the python library functions? Why or why not?

In [19]:
```
u, s, vt = np.linalg.svd(data_norm)
```

In [20]:
```
print(u.shape)
print(u) # U matrix in the svd factorisation of the standardised data matrix
```

```
(10, 10)
[[-0.12903036 -0.21839227  0.02025485 -0.11486499  0.01523355 -0.10749197
   0.42683596 -0.26415253  0.05492814 -0.80949526]
 [ 0.66669301  0.02174425  0.19999666  0.54703146  0.14989506 -0.39686657
   0.03199942  0.09473135  0.08301484 -0.13763891]
 [ 0.14638086 -0.10060633  0.09464466 -0.23565773  0.38826045  0.35442037
  -0.39344473  0.56546023  0.02839459 -0.39019043]
 [-0.51324075 -0.31491122  0.22138085  0.7147668   0.02147908  0.24432319
  -0.09351423  0.07964978 -0.02823629 -0.0383712 ]
 [-0.22137978  0.34314959  0.2064308  -0.02027774  0.82547525 -0.06306849
   0.16149905 -0.24337483 -0.02409298  0.13759994]
 [ 0.27162977  0.41840156 -0.270211    0.2652531  -0.0489155   0.71281627
   0.21997141 -0.18339279  0.03728132 -0.11778888]
 [ 0.19095172 -0.4144386   0.27083978 -0.14380081  0.06291575  0.2523406
   0.66563932  0.26795389 -0.04072629  0.33701252]
 [-0.29646094  0.54376056 -0.01319194  0.07910188 -0.17967015 -0.21550941
   0.34596521  0.63215821  0.00747188 -0.1091183 ]
 [-0.06800976  0.00805033  0.10711011 -0.04601466 -0.03476136  0.03660289
  -0.01160111 -0.01979099  0.98605644  0.0796131 ]
 [-0.04753379 -0.28675787 -0.83725476  0.14598955  0.32379262 -0.14862792
   0.12598353  0.17458768  0.11178445  0.08614862]]
```

In [21]:

```
print(s.shape)
print("Singular values obtained from numpy: ", s)
```

```
(3,)
Singular values obtained from numpy:  [4.20567966 3.04479214 1.74398953]
```

In [22]:

```
print(vt.shape)
print(vt) # eigenvector matrix of cov of data matrix: transformation matrix
```

```
(3, 3)
[[-0.64200458 -0.68636164  0.34166917]
 [ 0.38467229  0.09713033  0.91792861]
 [-0.66321742  0.72074503  0.20166619]]
```

In [23]:

```
proj_data_svd = np.dot(data_norm, vt.T) # Projected data: X(mxn).V(nxn)
print(np.cov(proj_data_svd.T)) # Covariance of the projected data is a diagonal matrix!
```

```
[[ 1.96530460e+00  9.52200081e-18 -4.52437690e-16]
 [ 9.52200081e-18  1.03008435e+00  1.67081863e-16]
 [-4.52437690e-16  1.67081863e-16  3.37944386e-01]]
```

The results obtained from numpy.linalg.svd function are again very similar to those obtained by both python and sklearn functions. The projected data has a diagonalised covariance matrix signifying almost 0 correlation between its columns.

(g) How many PCs are sufficient to represent the data in reduced dimensions with 95 % accuracy. Show how did you come up with you answer.

In [25]:
```
print(pca_sk.explained_variance_ratio_)
```

```
[0.58959138 0.30902531 0.10138332]
```

As can be seen above, the variance ratios of the 3 PCs obtained from the given data is arranged in a descending order. In order to represent the data with 95% accuracy, we would need all the 3 PCs because the largest 2 can explain 90% of data but the rest 10% can be represented after including the 3rd PC only.