UNIVERSITY OF TEXAS ARLINGTON

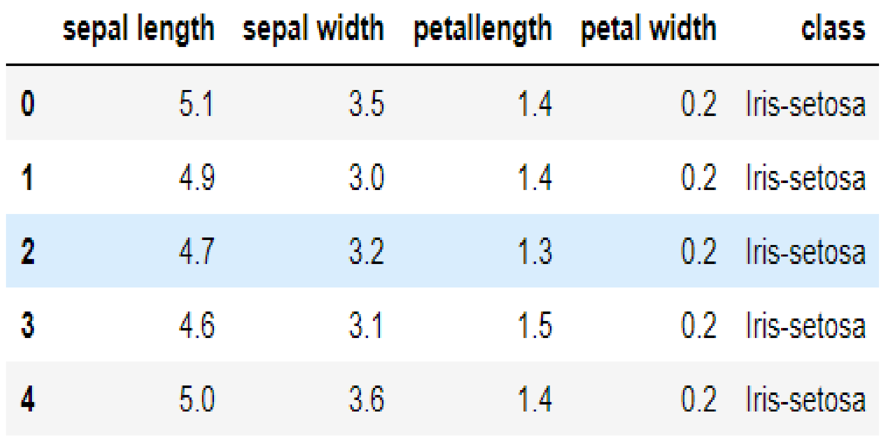
DATA MINING (IE 6318) HW3

Submittied by
ANUJ ANANDAKUMAR
1001746608
(10/16/2020)

Libraries used for the assignment

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.pyplot import figure
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns
%matplotlib inline
import yellowbrick as yb
from yellowbrick.features import ParallelCoordinates
from yellowbrick.datasets import load_occupancy
import plotly.graph_objects as go
from sklearn import preprocessing
import plotly.express as px
from math import *
from decimal import Decimal
from math import *
from decimal import Decimal
import numpy as np
import pandas as pd
from math import *
from decimal import Decimal
import scipy as stats
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

data=pd.read_csv('hw2_iris.csv')
data.head()
```

Out[18]:

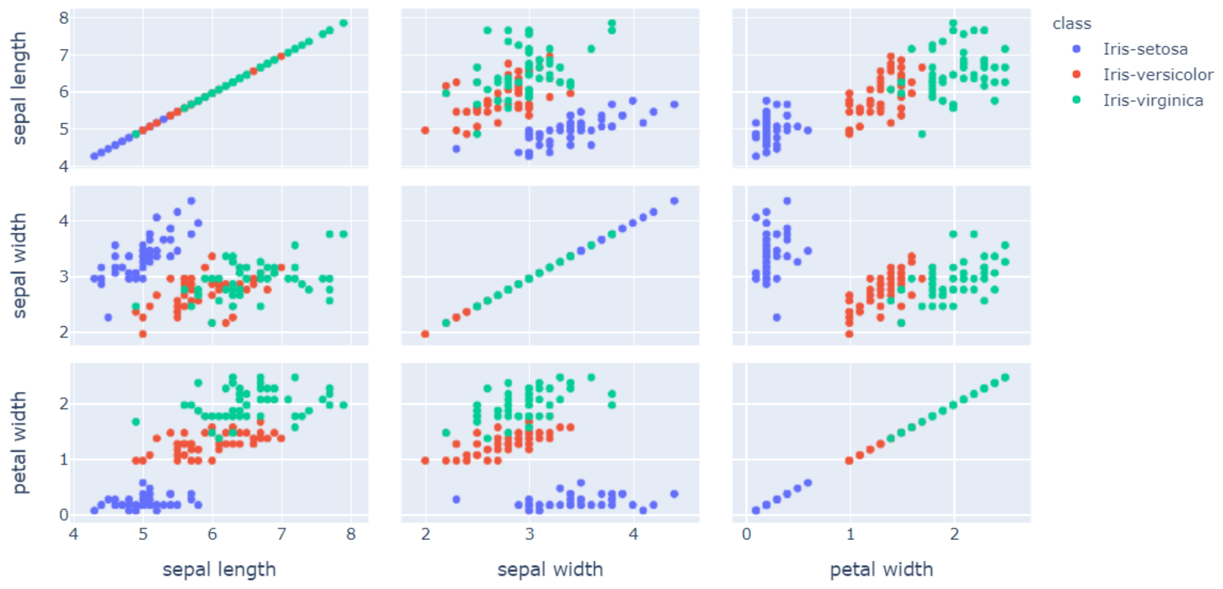


	sepal length	sepal width	petallength	petal width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Explore the Iris dataset and report the following:

1) 2D scatter plots of the four features (creates a matrix of scatter plots for each pair of the features)

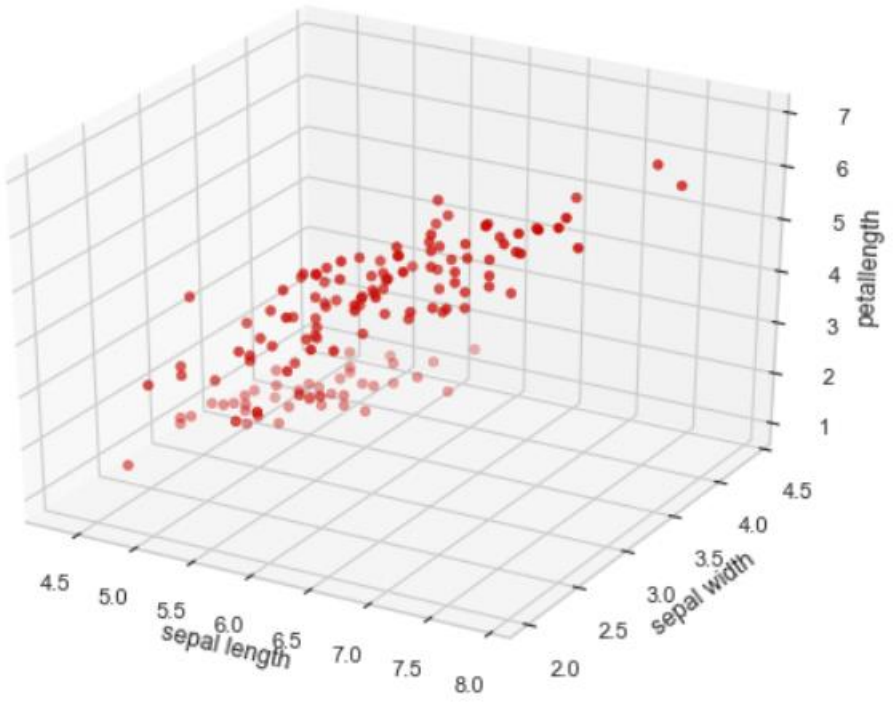
```
fig = px.scatter_matrix(data,
   dimensions=["sepal length","sepal width","petal width"],
   color="class")
fig.show()
```



2) 3D scatter plot of three features: sepal length, sepal width, petal width.

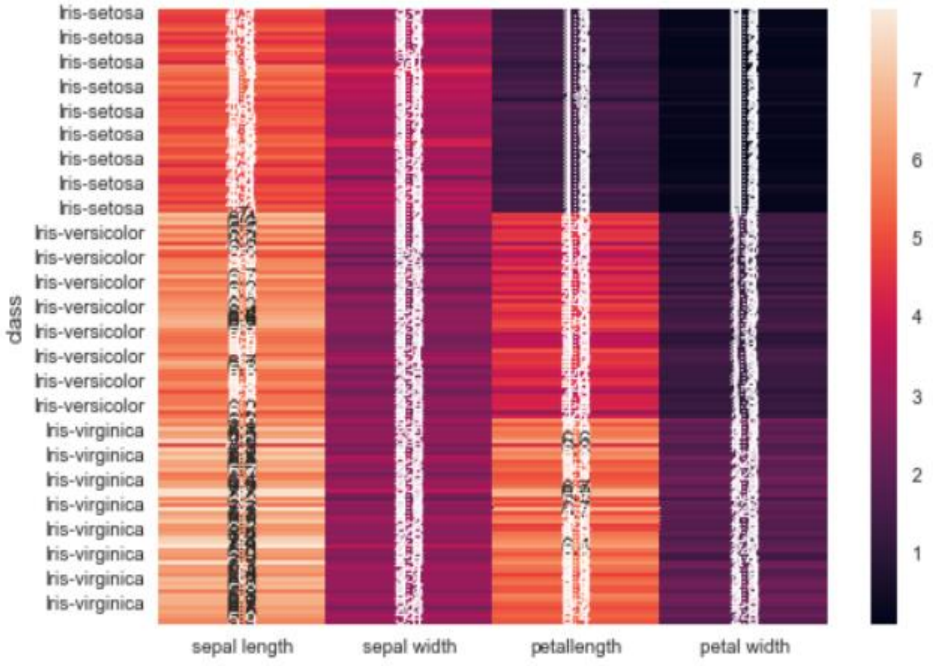
```
fig = plt.figure(num=None, figsize=(8, 6), dpi=80)
ax = fig.add_subplot(111, projection='3d')
x = data['sepal length']
y = data['sepal width']
z = data['petallength']
```

```
ax.scatter(x,y,z,c='r',marker = 'o')
ax.set_xlabel('sepal length')
ax.set_ylabel('sepal width')
ax.set_zlabel('petallength')
plt.show()
```



3) Visualization of the feature matrix (column 1-4) as an image.

```
irisdataset= pd.read_csv("hw2_iris.csv", index_col='class')
plt.pcolor(irisdataset)
sns.heatmap(irisdataset, annot=True)
```

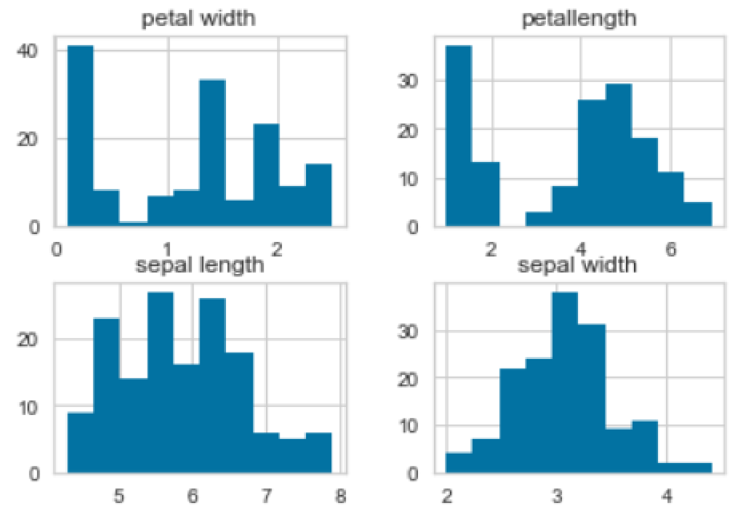


4) For each class, generate histograms for the four features.

plt.figure(num=None, figsize=(8, 6), dpi=80)

data.hist()

plt.show()



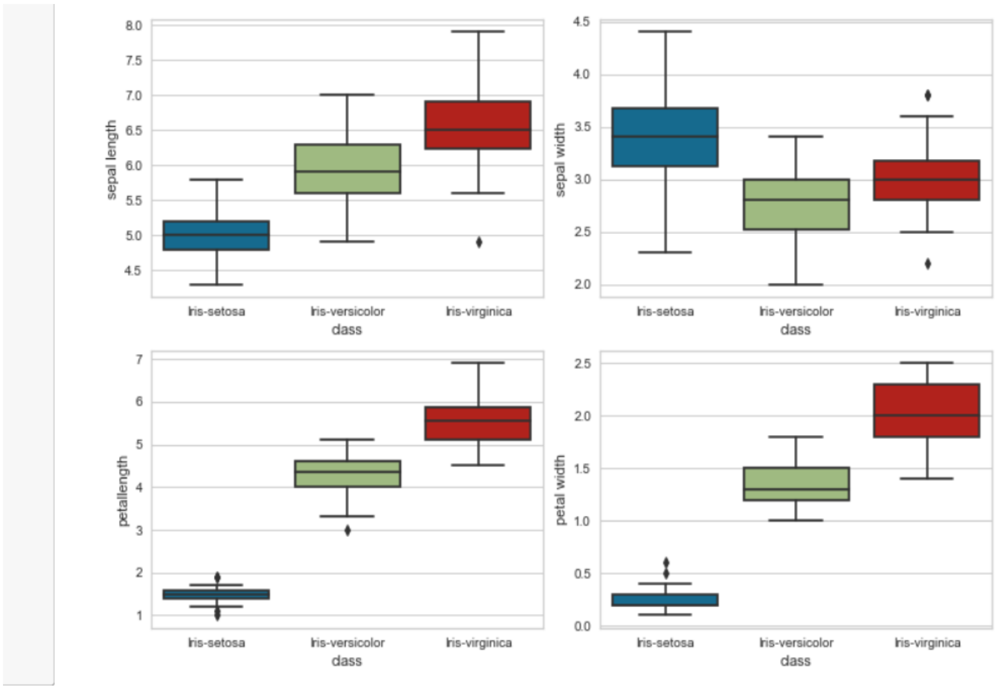
5) For each class, generate boxplots of the four features.

fig, axs = plt.subplots(2,2,figsize=(10,7))

x = data['class']

y = data['sepal length']

```
sns.boxplot(x = data['class'], y = data['sepal length'], ax = axs[0,0]);
sns.boxplot(x = data['class'], y = data['sepal width'],  ax = axs[0,1]);
sns.boxplot(x = data['class'], y = data['petallength'],  ax = axs[1,0]);
sns.boxplot(x = data['class'], y = data['petal width'],  ax = axs[1,1]);
fig.tight_layout(pad=1.0);
```



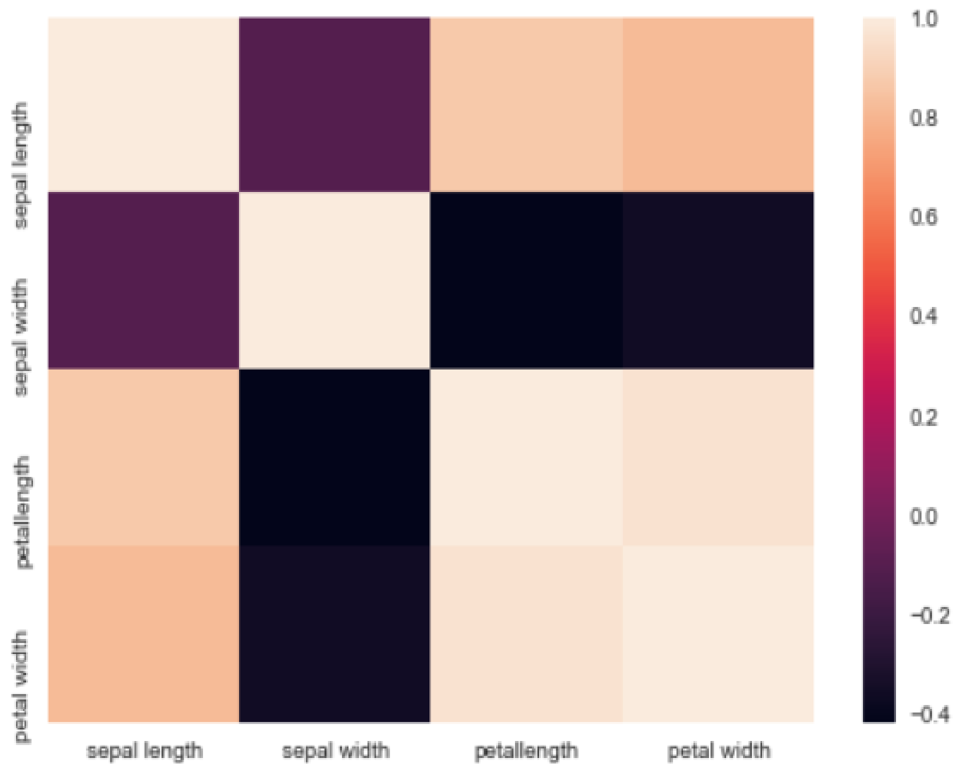
6) Calculate the correlation matrix of the four features

```
df = pd.DataFrame(data,columns=['sepal length','sepal width','petallength','petal width'])
corrMatrix = df.corr()
print (corrMatrix)
```

```
              sepal length  sepal width  petallength  petal width
sepal length      1.000000    -0.109369     0.871754     0.817954
sepal width      -0.109369     1.000000    -0.420516    -0.356544
petallength       0.871754    -0.420516     1.000000     0.962757
petal width       0.817954    -0.356544     0.962757     1.000000
```

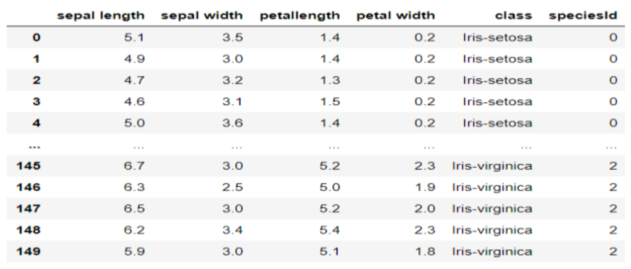
7) Visualize the correlation matrix as an image.

```
corr = data.corr()
sns.heatmap(corr,
        xticklabels=corr.columns.values,
        yticklabels=corr.columns.values)
plt.show()
```



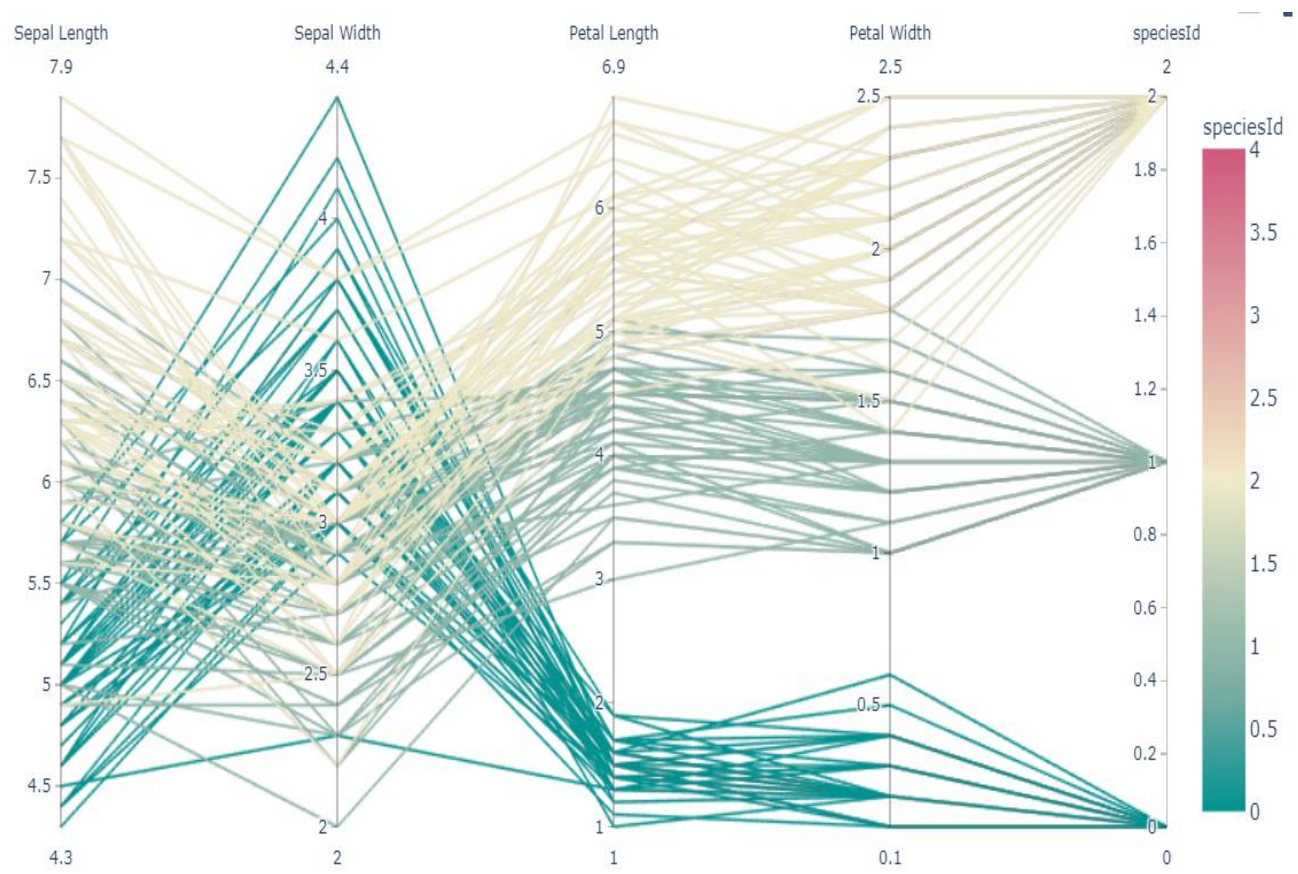
8) Create a parallel coordinate plot of the four features

```
data2=preprocessing.LabelEncoder()
speciesId=data2.fit_transform(list(data["class"]))
speciesId
data["speciesId"]=speciesId
data
```



	sepal length	sepal width	petallength	petal width	class	speciesId
0	5.1	3.5	1.4	0.2	Iris-setosa	0
1	4.9	3.0	1.4	0.2	Iris-setosa	0
2	4.7	3.2	1.3	0.2	Iris-setosa	0
3	4.6	3.1	1.5	0.2	Iris-setosa	0
4	5.0	3.6	1.4	0.2	Iris-setosa	0
...	...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica	2
146	6.3	2.5	5.0	1.9	Iris-virginica	2
147	6.5	3.0	5.2	2.0	Iris-virginica	2
148	6.2	3.4	5.4	2.3	Iris-virginica	2
149	5.9	3.0	5.1	1.8	Iris-virginica	2

```
fig = px.parallel_coordinates(data,color="speciesId", labels={"class": "Species","sepal width": "Sepal Width", "sepal length": "Sepal Length",
              "petal width": "Petal Width", "petallength": "Petal Length", },
                         color_continuous_scale=px.colors.diverging.Tealrose,
                         color_continuous_midpoint=2)
fig.show()
```


Sepal Length
7.9
7.5
7
6.5
6
5.5
5
4.5
4.3
Sepal Width
4.4
4
3.5
3
2.5
2
2
Petal Length
6.9
6
5
4
3
2
1
1
Petal Width
2.5
2.5
2
1.5
1
0.5
0.1
speciesId
2
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
speciesId
4
3.5
3
2.5
2
1.5
1
0.5
0

2. Practice Data Distance Measures

1) Make a function for Minkowski Distance. (3 function inputs: vector A, vector B, and order r)

```
def minkowski(A,B,r):
    vari1=np.array(A)
    vari2=np.array(B)
    vari3=np.array(vari1) - np.array(vari2)
    sum=0
    for i in range(4):
        sum+=pow(abs(vari3[i]),r)
    dist = pow(sum,1/r)
    return round(dist,3)
```

2) Make a function for T-statistics Distance. (3 function inputs: time series A, time series B)

```
def tstatistics(A,B):
    mean=abs(np.mean(tstat1)-np.mean(tstat2))
    tstat3=tstat1-tstat2
    variance=np.var(tstat3)
    tstatdist=mean/variance
    return tstatdist
```

3) Make a function for Mahalanobis Distance. (3 function inputs: vector A, vector B, and covariance matrix M)

```
def mahalanobis(X,Y,M):
    var1=np.array(X)
    var2=np.array(Y)
    var3=np.array(var1) - np.array(var2)
    var4=np.transpose(var3)
    M_inv=np.linalg.inv(M)
    a=np.matmul(var3,M_inv)
    b=np.matmul(a,var4)
    r = np.sqrt(b)
    return r
```

3. For a new iris data sample S with a feature vector of [5.0000, 3.5000, 1.4600, 0.2540], calculate the distances between the new sample and the 150 samples in the iris dataset using the distance functions you made:

1) Calculate Minkowski distances with r = 1, 2, 10, respectively, and plot the obtained distances.

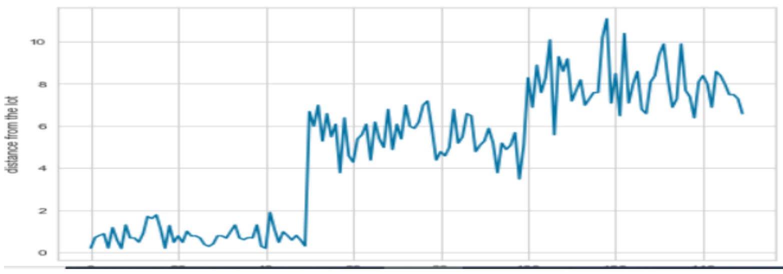
```
bluba = data.to_numpy()
duluba = bluba[:,:4]
x = [5.0000, 3.5000, 1.4600, 0.2540]
distance = np.zeros(shape=(150,))
for j in range(150):
   y=duluba[j]
   z=1
   distance[j]=minkowski(x,y,z)
print(distance)
plt.figure(figsize = (8, 6))
plt.plot(distance)
plt.xlabel('Sample Number')
plt.ylabel('Distance')
plt.show()
```

2) Calculate Mahalanobis distances and plot the obtained distances.

```
bluba = data.to_numpy()
duluba = bluba[:,:4]
x = [5.0000, 3.5000, 1.4600, 0.2540]
distance = np.zeros(shape=(150,))
for j in range(150):
   y=duluba[j]
   z=1/2/10   # can use any one.
   distance[j]=minkowski(x,y,z)
print(distance)
plt.figure(figsize = (8, 6))
plt.plot(distance)
plt.xlabel('lot Number')
plt.ylabel('distance from the lot')
plt.show()
```

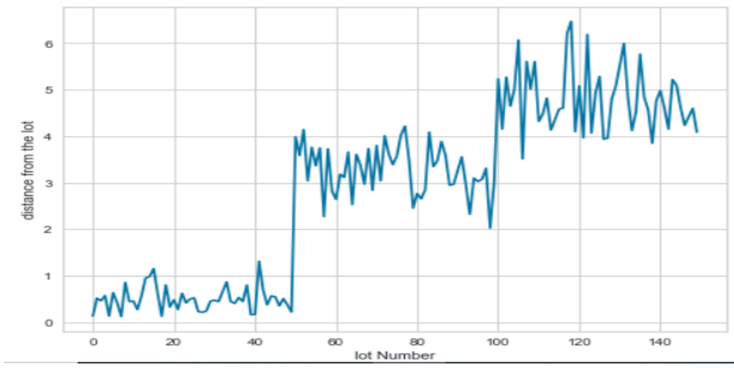
r=1

```
 0.214   0.714   0.814   0.894   0.214   1.186   0.606   0.194   1.314   0.694
 0.694   0.494   0.914   1.714   1.614   1.786   1.106   0.206   1.286   0.486
 0.794   0.486   1.014   0.786   0.794   0.694   0.386   0.294   0.414   0.794
 0.794   0.686   0.994   1.314   0.694   0.614   0.714   0.694   1.314   0.294
 0.206   1.906   1.114   0.486   0.986   0.806   0.594   0.814   0.594   0.314
 6.686   5.986   6.986   5.286   6.586   5.486   6.086   3.786   6.386   4.586
 4.286   5.386   5.586   6.086   4.386   6.186   5.386   4.986   6.786   4.886
 6.086   5.386   6.986   5.986   5.886   6.186   6.986   7.186   5.886   4.386
 4.786   4.586   4.986   6.786   5.186   5.486   6.586   6.486   4.786   5.086
 5.286   5.886   5.186   3.786   5.186   4.886   5.086   5.686   3.486   5.086
 8.286   6.886   8.886   7.586   8.286  10.086   5.586   9.286   8.586   9.186
 7.186   7.686   8.186   6.986   7.286   7.586   7.586  10.186  11.086   7.086
 8.486   6.486  10.386   7.086   7.986   8.586   6.786   6.586   8.086   8.386
 9.386   9.886   8.186   6.886   7.286   9.886   7.686   7.386   6.386   8.086
 8.386   7.986   6.886   8.586   8.386   7.986   7.486   7.486   7.286   6.586]
```



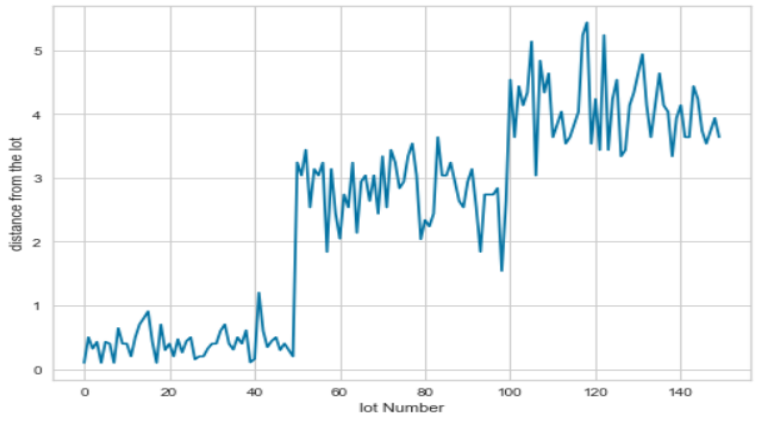
r=2

```
[0.129 0.516 0.457 0.57  0.129 0.632 0.419 0.12  0.852 0.442 0.452 0.269
 0.563 0.945 0.98  1.15  0.606 0.125 0.8   0.322 0.48  0.27  0.62  0.41
 0.497 0.522 0.226 0.211 0.238 0.45  0.472 0.439 0.652 0.864 0.442 0.401
 0.528 0.442 0.799 0.157 0.166 1.311 0.692 0.373 0.561 0.544 0.35  0.506
 0.367 0.216 3.988 3.584 4.142 3.039 3.762 3.364 3.747 2.272 3.725 2.819
 2.64  3.181 3.114 3.658 2.529 3.607 3.377 2.968 3.732 2.834 3.801 3.041
 4.01  3.618 3.388 3.573 4.025 4.215 3.486 2.453 2.766 2.652 2.851 4.087
 3.347 3.473 3.886 3.587 2.945 2.966 3.256 3.554 2.966 2.32  3.099 3.024
 3.074 3.311 2.024 3.007 5.233 4.152 5.271 4.645 5.012 6.069 3.513 5.609
 5.012 5.606 4.317 4.478 4.819 4.132 4.357 4.581 4.606 6.215 6.471 4.096
 5.086 3.966 6.188 4.069 4.931 5.285 3.934 3.96  4.796 5.073 5.522 5.996
 4.835 4.121 4.525 5.767 4.842 4.564 3.847 4.765 4.979 4.605 4.152 5.218
 5.094 4.617 4.234 4.42  4.6   4.087]
```



r=10

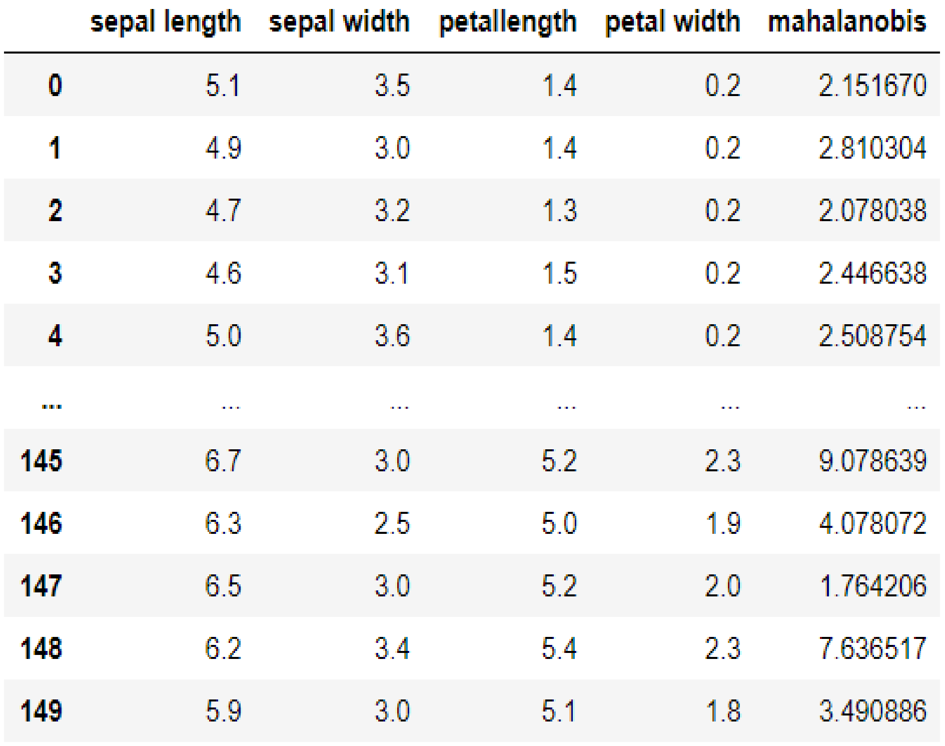
```
[0.1   0.5   0.322 0.429 0.1   0.429 0.4   0.1   0.643 0.4   0.4   0.201
 0.5   0.702 0.801 0.907 0.429 0.1   0.7   0.3   0.4   0.201 0.47  0.262
 0.44  0.5   0.154 0.2   0.2   0.322 0.4   0.4   0.6   0.702 0.4   0.307
 0.5   0.4   0.609 0.107 0.16  1.2   0.6   0.346 0.441 0.5   0.3   0.402
 0.301 0.2   3.243 3.04  3.441 2.54  3.14  3.04  3.24  1.841 3.14  2.44
 2.049 2.74  2.54  3.24  2.14  2.941 3.04  2.64  3.04  2.44  3.34  2.54
 3.44  3.24  2.84  2.941 3.341 3.54  3.04  2.04  2.34  2.24  2.44  3.64
 3.04  3.04  3.241 2.94  2.64  2.54  2.94  3.14  2.54  1.843 2.74  2.74
 2.74  2.84  1.542 2.64  4.54  3.64  4.44  4.14  4.34  5.141 3.04  4.84
 4.34  4.641 3.64  3.84  4.04  3.54  3.642 3.841 4.04  5.241 5.441 3.54
 4.24  3.44  5.241 3.44  4.24  4.54  3.34  3.44  4.14  4.34  4.641 4.942
 4.14  3.64  4.14  4.642 4.141 4.04  3.34  3.94  4.141 3.642 3.64  4.44
 4.241 3.741 3.54  3.74  3.941 3.64 ]
```



2) Calculate Mahalanobis distances and plot the obtained distances.

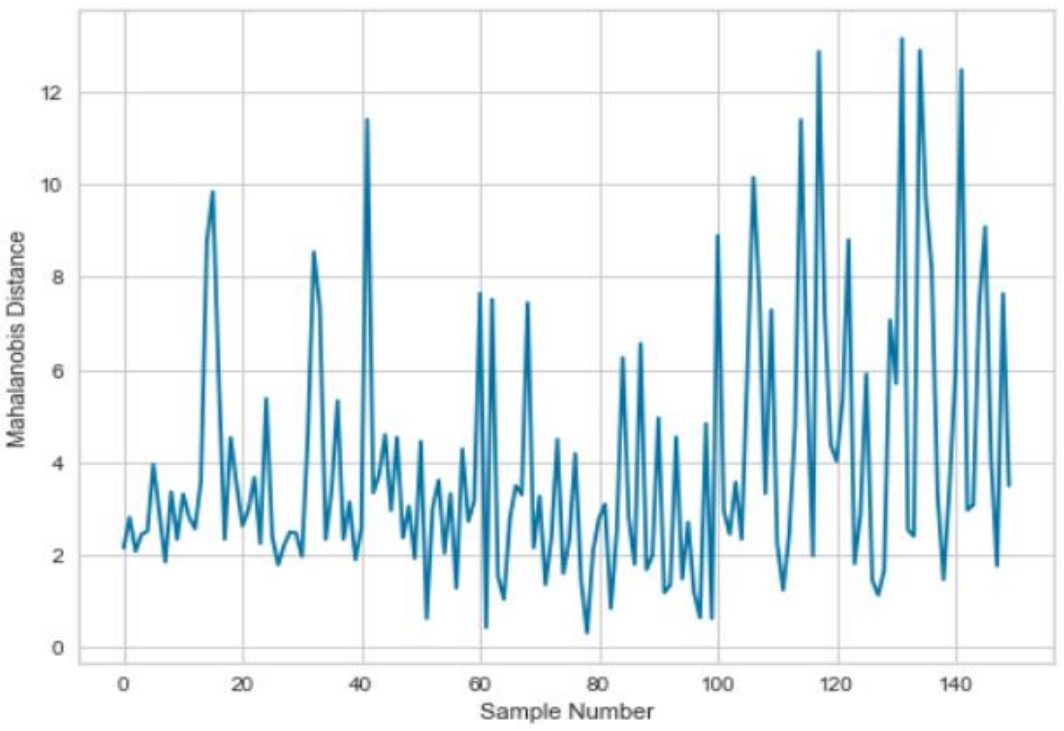
```
dataMahal=data2.copy()
def mahalanobis(x, datanew, cov=None):

    x_mu = x - np.mean(datanew)
    if not cov:
        cov = np.cov(datanew.values.T)
    inv_covmat = np.linalg.inv(cov)
    left = np.dot(x_mu, inv_covmat)
    mahal = np.dot(left, x_mu.T)
    return mahal.diagonal()

#create new column in dataframe that contains Mahalanobis distance for each row
x= [5.0000, 3.5000, 1.4600, 0.2540]
dataMahal['mahalanobis'] = mahalanobis(x=data2, datanew=data2[['sepal length', 'sepal width',
'petallength', 'petal width']])

#display first five rows of dataframe
```

dataMahal



	sepal length	sepal width	petallength	petal width	mahalanobis
0	5.1	3.5	1.4	0.2	2.151670
1	4.9	3.0	1.4	0.2	2.810304
2	4.7	3.2	1.3	0.2	2.078038
3	4.6	3.1	1.5	0.2	2.446638
4	5.0	3.6	1.4	0.2	2.508754
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	9.078639
146	6.3	2.5	5.0	1.9	4.078072
147	6.5	3.0	5.2	2.0	1.764206
148	6.2	3.4	5.4	2.3	7.636517
149	5.9	3.0	5.1	1.8	3.490886

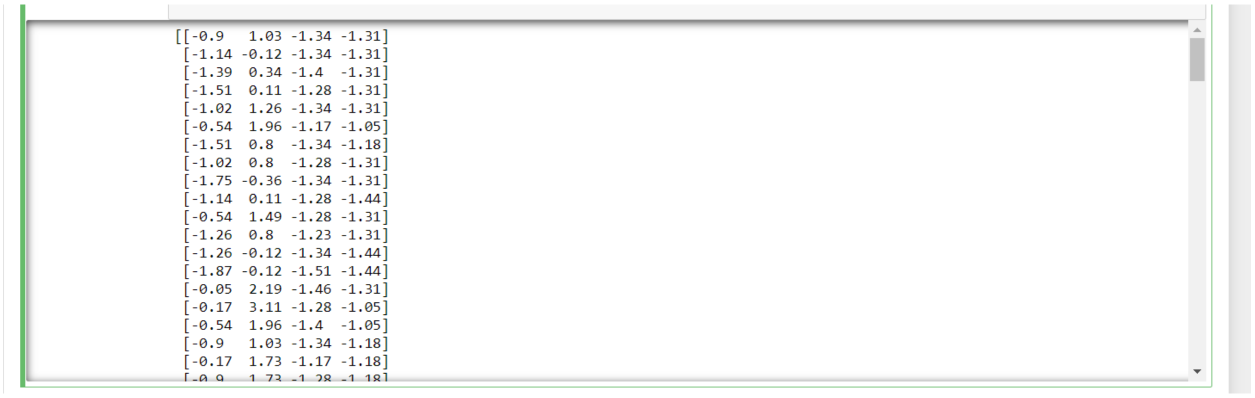
```
A=dataMahal['mahalanobis'].to_numpy()
plt.figure(figsize = (8, 6))
plt.plot(A)
plt.xlabel('lot no')
plt.ylabel('Distance')
plt.show()
```


Mahalanobis Distance
Sample Number
0
2
4
6
8
10
12
0
20
40
60
80
100
120
140

4. Feature matrix normalization

1) Normalize the feature matrix of the IRIS dataset such that each feature has a mean of 0 and a standard deviation of 1 after normalization.

```
datanorm=data.to_numpy()
features=datanorm[:,:4]
norm=preprocessing.scale(features)
print(norm.round(2))
```



```
[[-0.9   1.03 -1.34 -1.31]
 [-1.14 -0.12 -1.34 -1.31]
 [-1.39  0.34 -1.4  -1.31]
 [-1.51  0.11 -1.28 -1.31]
 [-1.02  1.26 -1.34 -1.31]
 [-0.54  1.96 -1.17 -1.05]
 [-1.51  0.8  -1.34 -1.18]
 [-1.02  0.8  -1.28 -1.31]
 [-1.75 -0.36 -1.34 -1.31]
 [-1.14  0.11 -1.28 -1.44]
 [-0.54  1.49 -1.28 -1.31]
 [-1.26  0.8  -1.23 -1.31]
 [-1.26 -0.12 -1.34 -1.44]
 [-1.87 -0.12 -1.51 -1.44]
 [-0.05  2.19 -1.46 -1.31]
 [-0.17  3.11 -1.28 -1.05]
 [-0.54  1.96 -1.4  -1.05]
 [-0.9   1.03 -1.34 -1.18]
 [-0.17  1.73 -1.17 -1.18]
 [-0.9   1.73 -1.28 -1.18]
```

2) Calculate the correlation matrix of the four features after normalization.

```
df = pd.DataFrame(norm)
corrMatrix = df.corr()
print (corrMatrix)
```

```
          0         1         2         3
0  1.000000 -0.109369  0.871754  0.817954
1 -0.109369  1.000000 -0.420516 -0.356544
2  0.871754 -0.420516  1.000000  0.962757
3  0.817954 -0.356544  0.962757  1.000000
```

3) Compare the correlation matrix before and after normalization. If they are same?

```
              sepal length  sepal width  petallength  petal width
sepal length      1.000000    -0.109369     0.871754     0.817954
sepal width      -0.109369     1.000000    -0.420516    -0.356544
petallength       0.871754    -0.420516     1.000000     0.962757
petal width       0.817954    -0.356544     0.962757     1.000000
```

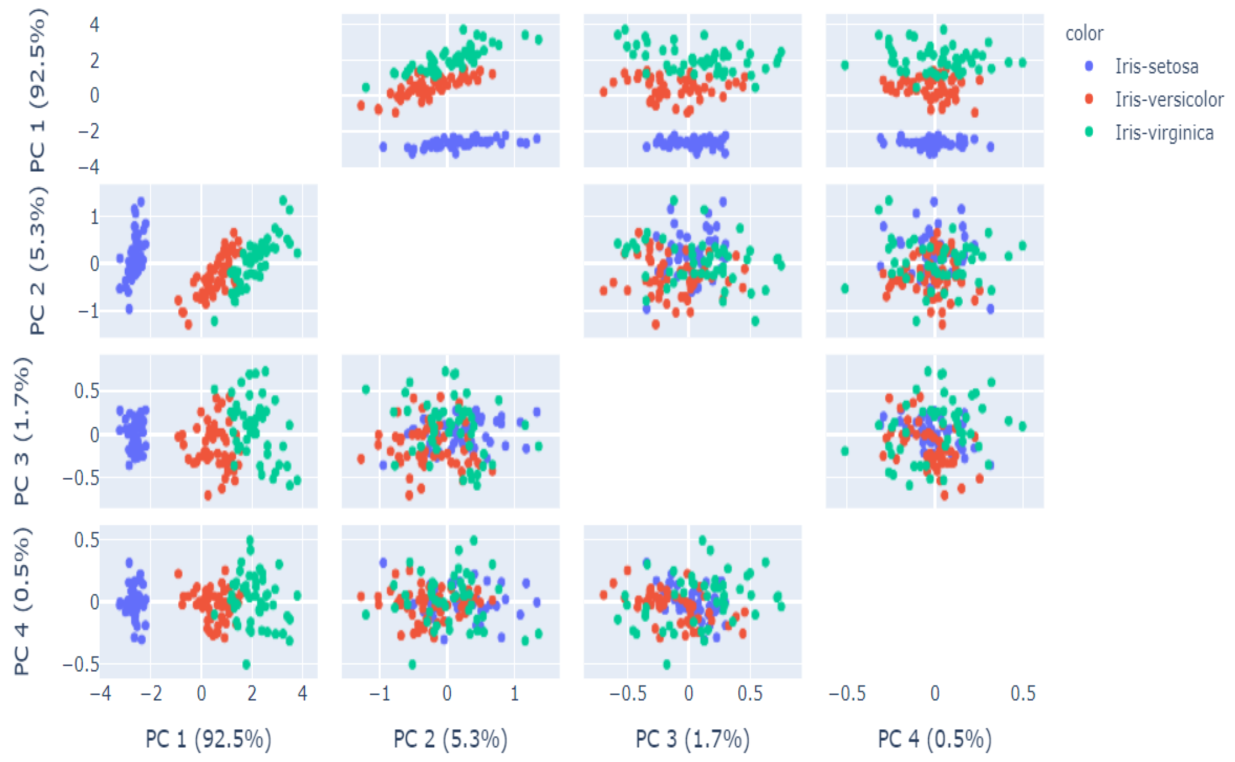
```
          0         1         2         3
0  1.000000 -0.109369  0.871754  0.817954
1 -0.109369  1.000000 -0.420516 -0.356544
2  0.871754 -0.420516  1.000000  0.962757
3  0.817954 -0.356544  0.962757  1.000000
```

Yes, they are same.

5. Principle Component Analysis (PCA) on the IRIS dataset

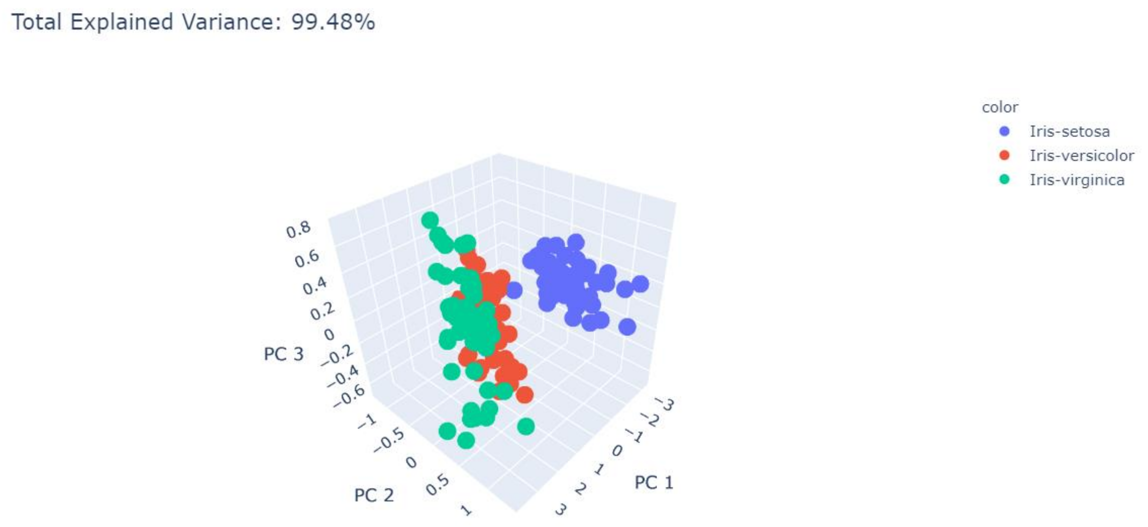
1) Create 2D scatter plots of each pair of the four components

```
features = ["sepal length","sepal width", "petallength","petal width"]

pca = PCA()
components = pca.fit_transform(data[features])
labels = {
   str(i): f"PC {i+1} ({var:.1f}%)"
   for i, var in enumerate(pca.explained_variance_ratio_ * 100)
}

fig = px.scatter_matrix(
   components,
   labels=labels,
   dimensions=range(4),
   color=data["class"]
)
fig.update_traces(diagonal_visible=False)
fig.show()
```



2) 3D scatter plot of the first three components

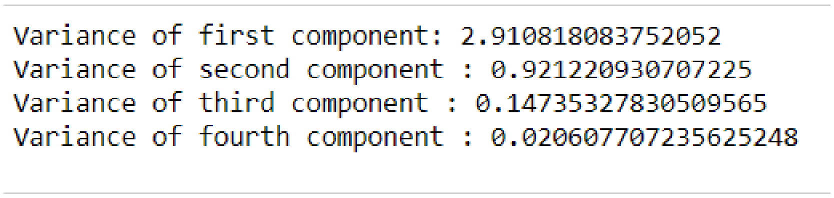
```
df = px.data.iris()
X = df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]

pca = PCA(n_components=3)
components = pca.fit_transform(X)

total_var = pca.explained_variance_ratio_.sum() * 100

fig = px.scatter_3d(
    components, x=0, y=1, z=2, color=df['species'],
    title=f'Total Explained Variance: {total_var:.2f}%',
    labels={'0': 'PC 1', '1': 'PC 2', '2': 'PC 3'}
)
fig.show()
```



3) Obtain the variance of each component and visualize in a figure plot.

```
iris_dataset= pd.read_csv("hw2_iris.csv")

iris_dataset.head()

classtypes = ["sepal length","sepal width", "petallength","petal width"]
a = iris_dataset.loc[:, classtypes].values
b = iris_dataset.loc[:,['class']].values
a = StandardScaler().fit_transform(x)
```

```
pd.DataFrame(data = a, columns = features).head()

pca = PCA(n_components=4)
principalComponents = pca.fit_transform(x)
irisnewvar = pd.DataFrame(data = principalComponents
            , columns = ['first component', 'second component', 'third component','fourth component'])

v1=np.array(irisnewvar['first component'])
v2=np.array(irisnewvar['second component'])
v3=np.array(irisnewvar['third component'])
v4=np.array(irisnewvar['fourth component'])

varianceone=np.var(v1)
variancetwo=np.var(v2)
variancethree=np.var(v3)
variancefour=np.var(v4)

print("Variance of first component:", varianceone, "\nVariance of second component :", variancetwo,
   "\nVariance of third component :", variancethree, "\nVariance of fourth component :", variancefour)
```

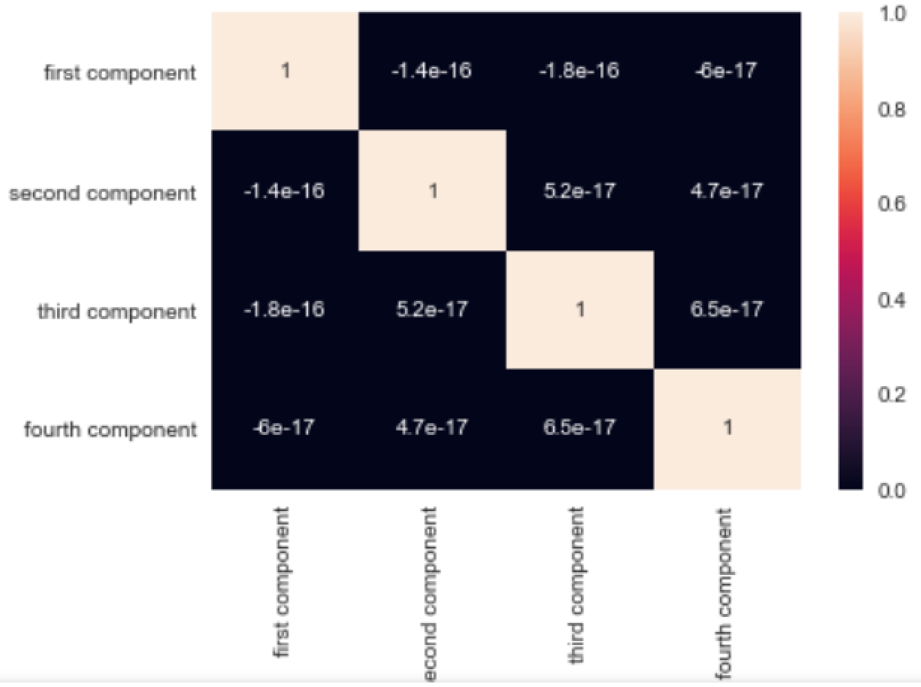


```
Variance of first component: 2.910818083752052
Variance of second component : 0.921220930707225
Variance of third component : 0.14735327830509565
Variance of fourth component : 0.020607707235625248
```

4) Calculate the correlation matrix of the four components

```
corrmatrix=irisnewvar.corr()
print(corrmatrix.round())
sns.heatmap(corrmatrix.round(), annot=True)
plt.show()
```

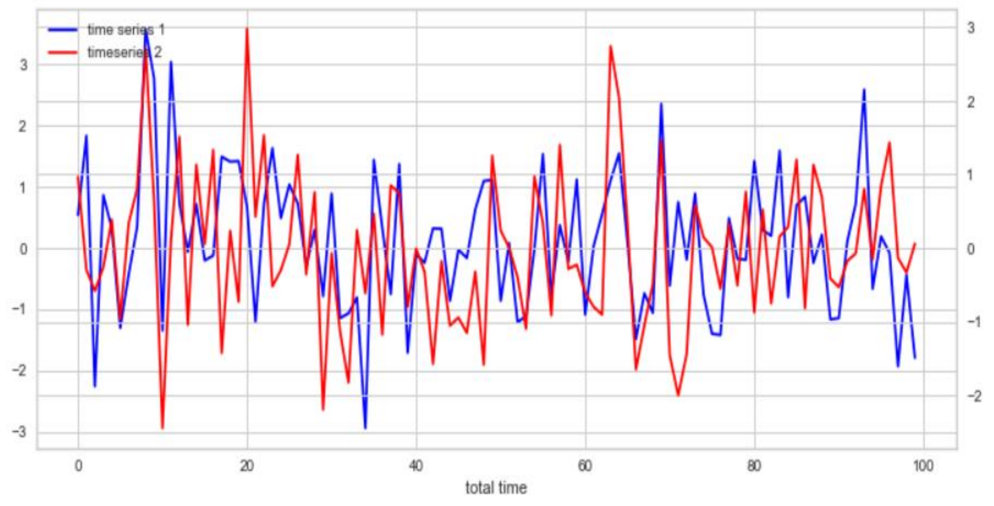
```
                   first component  second component  third component  \
first component       1.000000e+00     -1.378572e-16    -1.796920e-16
second component     -1.378572e-16      1.000000e+00     5.179182e-17
third component      -1.796920e-16      5.179182e-17     1.000000e+00
fourth component     -6.044030e-17      4.664679e-17     6.547852e-17

                   fourth component
first component       -6.044030e-17
second component       4.664679e-17
third component        6.547852e-17
fourth component       1.000000e+00
```



6. For the dataset with two time series in the “hw2_ts.txt” file, perform the following analysis:

1) Visualize the two time series in one figure plot.

```
df = pd.read_csv("hw2ts.csv")
plt.figure(figsize=(12,5))
plt.xlabel(' total time ')

ax1 = df.t1.plot(color='blue', grid=True, label='time series 1')
ax2 = df.t2.plot(color='red', grid=True, secondary_y=True, label='timeseries 2 ')

h1, l1 = ax1.get_legend_handles_labels()
h2, l2 = ax2.get_legend_handles_labels()


plt.legend(h1+h2, l1+l2, loc=2)
plt.show()
```



2) Calculate the T-statistics distance between the two time series using the function you made in 2.

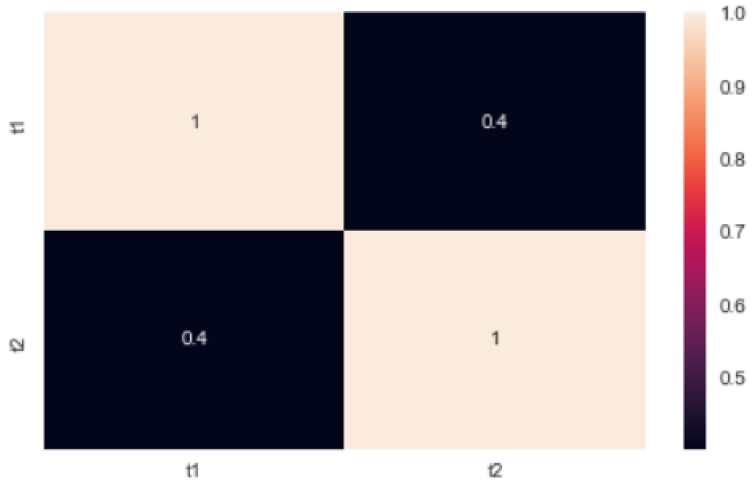
```
ts_data= pd.read_csv("hw2ts.csv")
ts1=np.array(ts_data["t1"])
ts2=np.array(ts_data["t2"])

def tstatistics(A,B):
   mean=abs(np.mean(ts1)-np.mean(ts2))
   ts3=ts1-ts2
   var=np.var(ts3)
   tdist=mean/var
   return tdist
tstatistics(ts1,ts2)
```

```
0.10731262850806823
```

3) Calculate the correlation of the two time series

```
corrmatrix=ts_data.corr()
print(corrmatrix)
sns.heatmap(corrmatrix, annot=True)
plt.show()
```

```
          t1        t2
t1  1.000000  0.403044
t2  0.403044  1.000000
```



1
0.4
0.4
1
t1
t2
t1
t2
1.0
0.9
0.8
0.7
0.6
0.5