clustering_lec_AB

April 4, 2022

1 CLUSTERING - Bioinformatics on Genomics 2022

1.1 THEMES

During this session we will work on: * PCA * hierarchical clustering * k-means

1.2 DATA

We will look at data from the publication: "RNAseq analysis of heart tissue from mice treated with atenolol and isoproterenol reveals a reciprocal transcriptional respons" (*2016, Prunotto et al.*)

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5015234/

1.3 PROGRAMMING

Python 3

1.3.1 STEP1: Libs

These are the libs we will be using.

[1]:
```
import numpy
from numpy import genfromtxt
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import scipy.stats as stats
from pandas import DataFrame
from sklearn.cluster import KMeans
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.cluster import AgglomerativeClustering
```

1.3.2 STEP2: data import

Import the following data (found under input/PGX_data). * tmm_data.tsv * genes_entrezid.tsv * samples.tsv

First import expression data and gene ids into data structures: numpy arrays are a good choice. Then import samples: here you need to open the file and read a list of strings.

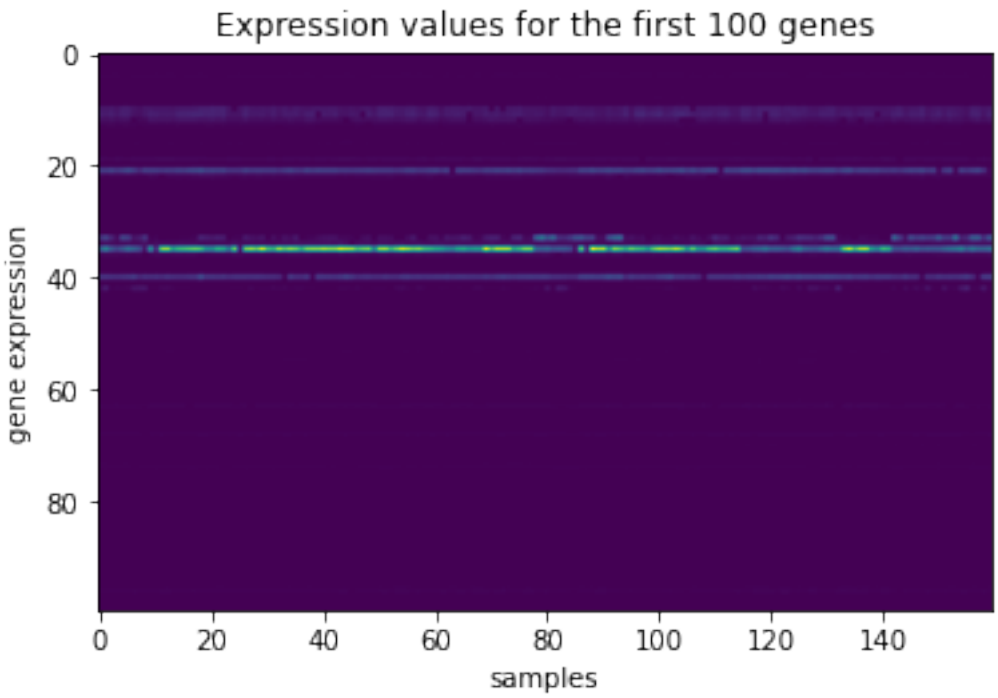
[2]:
```
# import expression data and gene ids
expression_data = genfromtxt("input/PGX_data/tmm_data.tsv",␣
 ↪delimiter='\t',dtype=None)

#export gene ids
genes = genfromtxt("input/PGX_data/genes_entrezid.tsv",encoding='latin1',␣
 ↪delimiter='\t',dtype=None)

# import samples
samples = genfromtxt("input/PGX_data/samples.tsv",encoding='latin1',␣
 ↪delimiter='\t',dtype=None)
```

1.3.3 STEP3: Inspect and prepare your data

What can you infer from the dimensions of the data? How many rows and columns in each data structure? Visualize a few elements to get an idea of the contents.

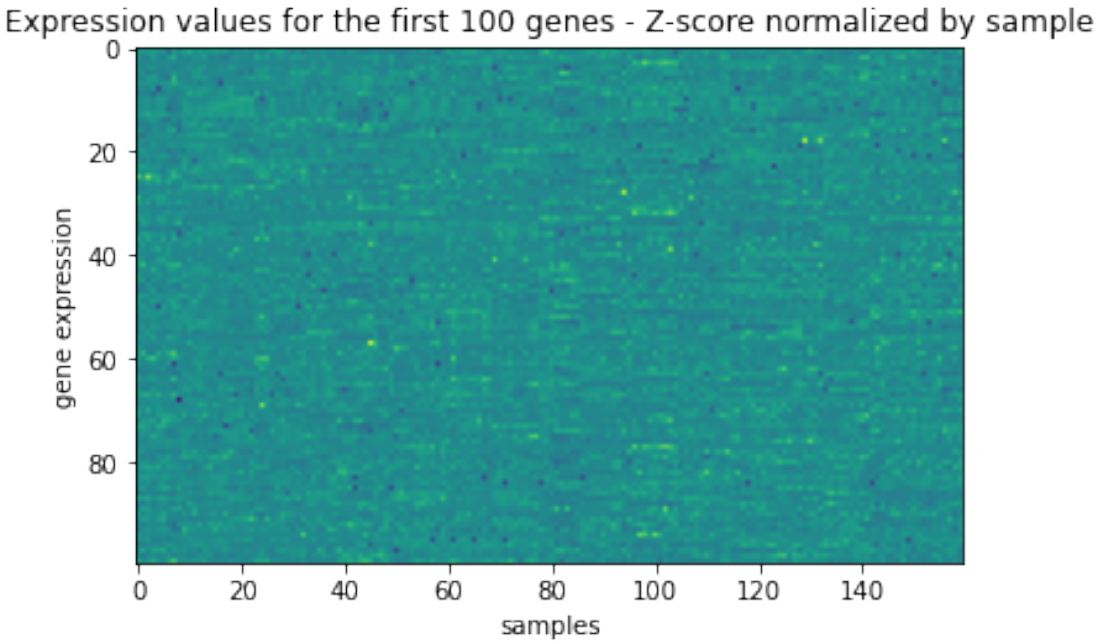
[3]:
```
print("Number of rows and columns for each data matrix:")
print(expression_data.shape)
print("Number of gene ids:")
print(genes.shape)
print("Number of samples:")
print(samples.shape,"\n")

print("Expression matrix (first sample, first 10 genes):")
print(expression_data[:,0][:10])

plt.imshow(expression_data[:100,:])
plt.title("Expression values for the first 100 genes")
plt.xlabel("samples")
plt.ylabel("gene expression")
plt.show()
```

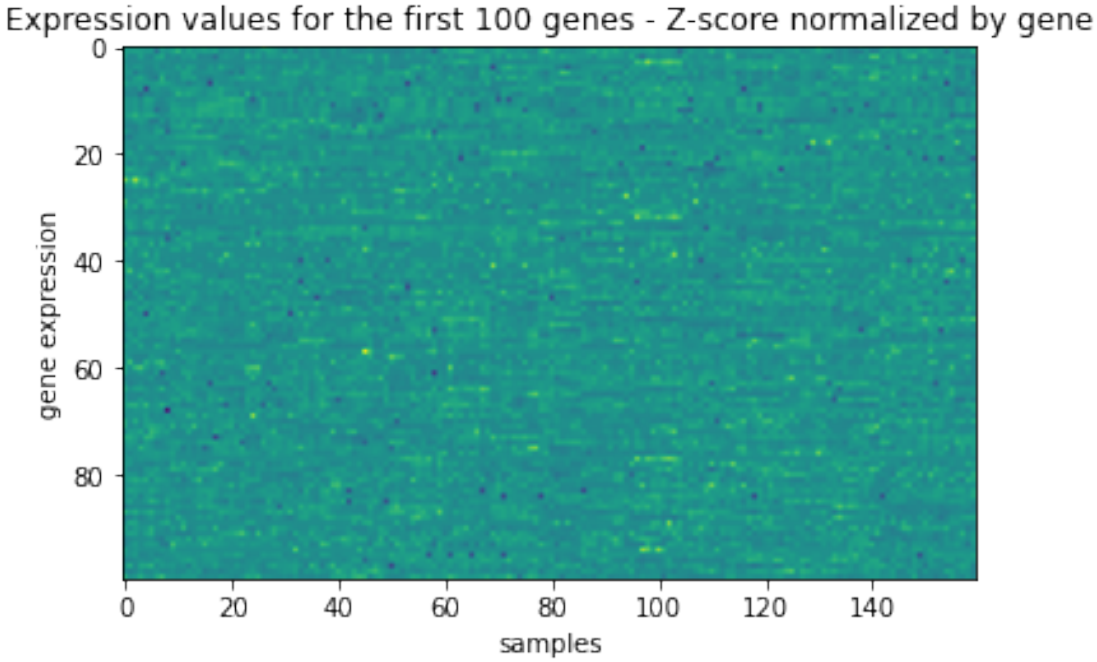
```
Number of rows and columns for each data matrix:
(16039, 160)
Number of gene ids:
(16040,)
Number of samples:
(160,)

Expression matrix (first sample, first 10 genes):
[   0.          131.8401538   716.2127274    50.77627545 1741.53716674
  650.2926505    84.62712575  475.6935279   939.80650174   67.7017006 ]
```



Now, prepare the expression date for our analysis by performing zscore normalization (first by sample, then by gene). Visualize a few elements before and after normalization.

[4]:
```
import numpy as np
from scipy.stats import zscore
# normalize the data by sample
expression_data_normalized = zscore(expression_data,axis=1)

plt.title("Expression values for the first 100 genes - Z-score normalized by␣
 ↪sample")
plt.imshow(expression_data_normalized[:100,:])
plt.xlabel("samples")
plt.ylabel("gene expression")
plt.show()

# normalize the data by sample
expression_data_normalized = zscore(expression_data_normalized,axis=0)

plt.title("Expression values for the first 100 genes - Z-score normalized by␣
 ↪gene")
plt.imshow(expression_data_normalized[:100,:])
plt.xlabel("samples")
plt.ylabel("gene expression")
plt.show()
```



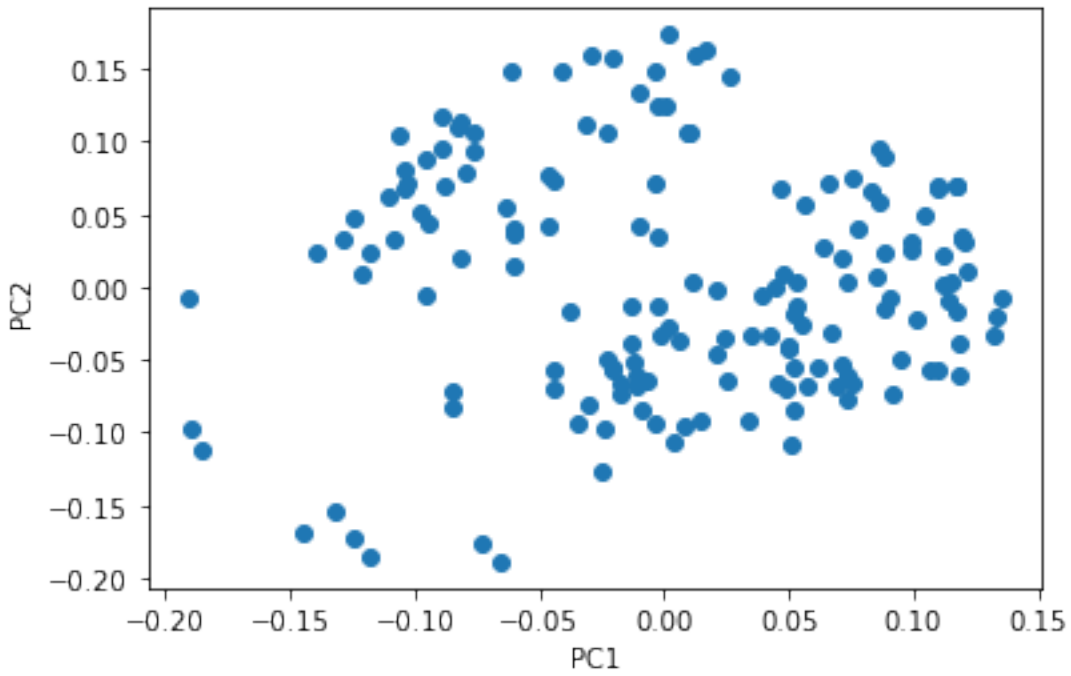


1.3.4 STEP4: perform PCA dimensionality reduction

Run PCA on expression data. Capture the loadings corresponding to the first two components.

[5]:
```
#fit three principal components to the normalized expression data
pca = PCA(n_components=3)
pca.fit(expression_data_normalized)

#print their dimensions
print("The proportion of variance explained by each principal component␣
 ↪analysis:")
print(pca.explained_variance_ratio_,"\n")
print("PCA dimensions:")
print(pca.components_.shape)

#plot the principal components
plt.scatter(pca.components_[0],pca.components_[1])
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.show()
```

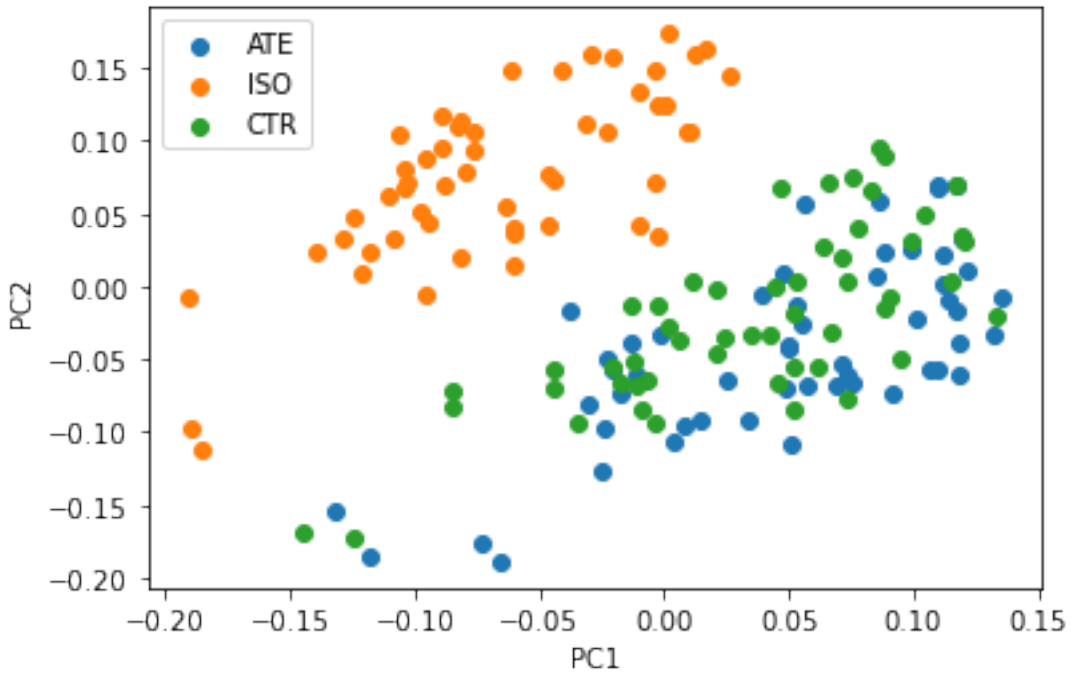
```
The proportion of variance explained by each principal component analysis:
[0.06198788 0.05199713 0.04577463]

PCA dimensions:
(3, 160)
```

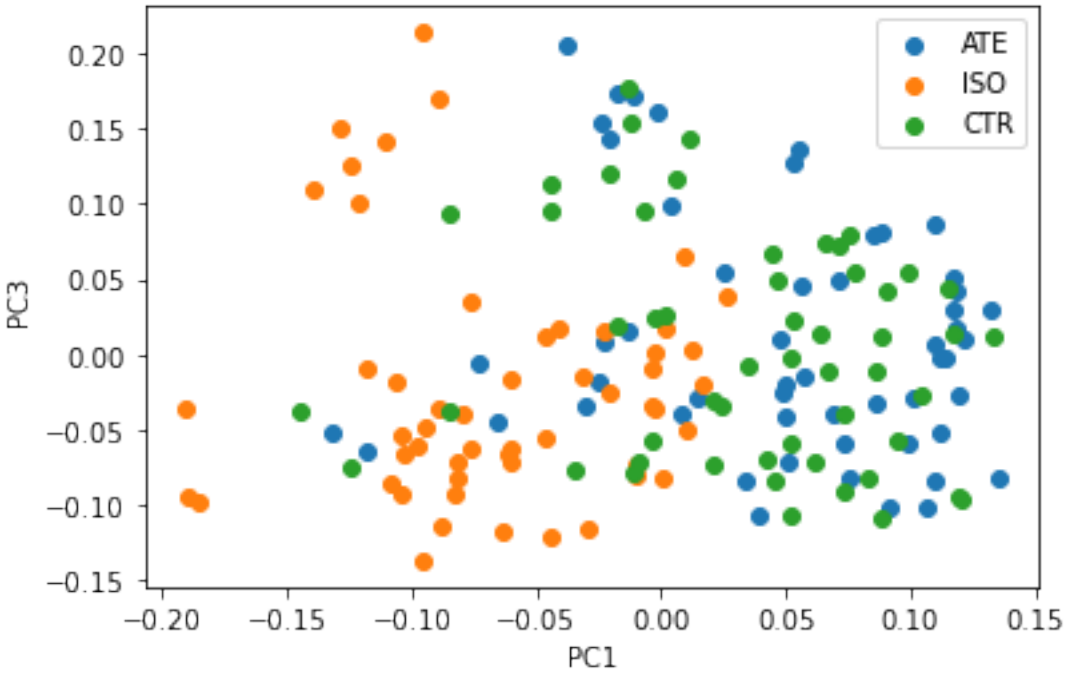


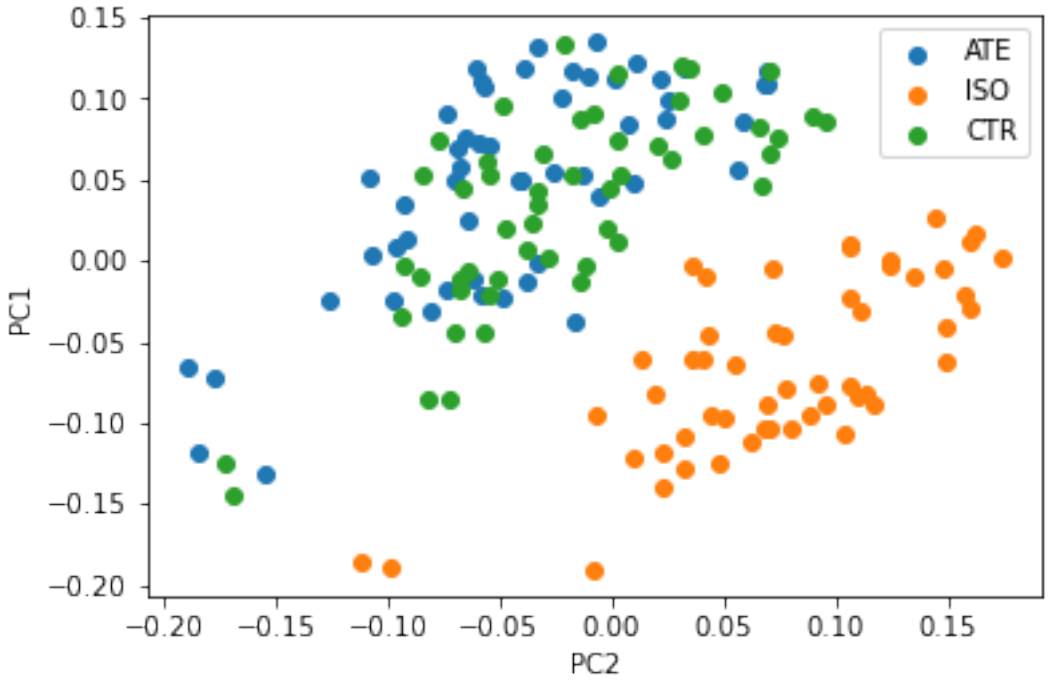
Plot the transformed data using colors for each treatment. Look at the sample name ending in ATE/ISO/CTR. * mice treated with the -blocker atenolol (sample name ending in ATE) * mice

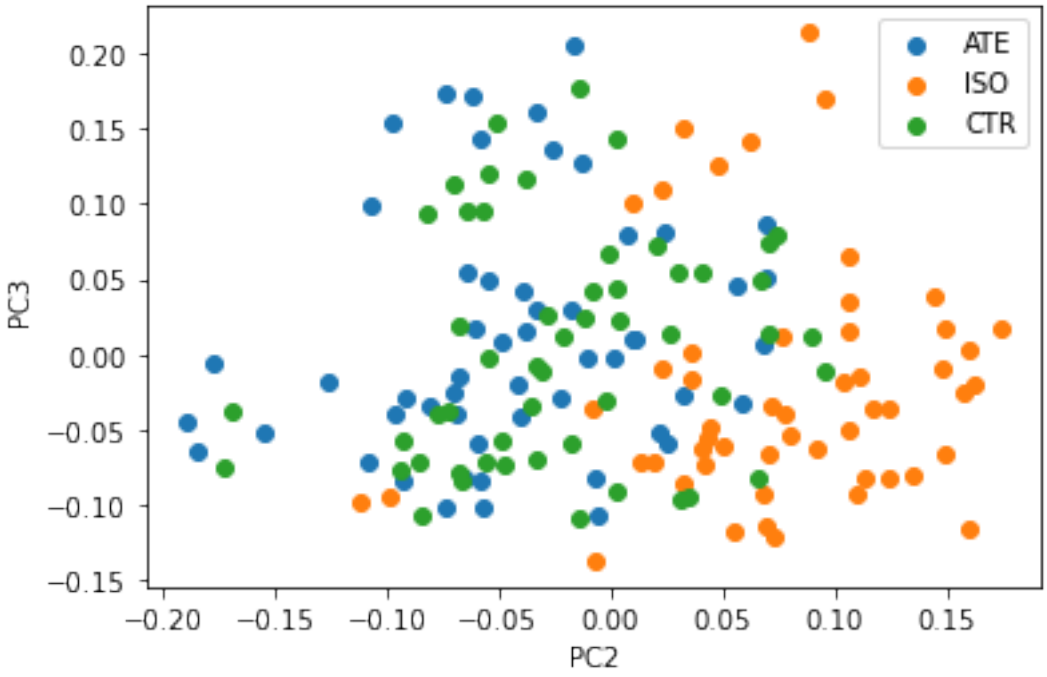
treated with the -agonist isoproterenol (sample name ending in ISO) * controls (sample name ending in CTR)

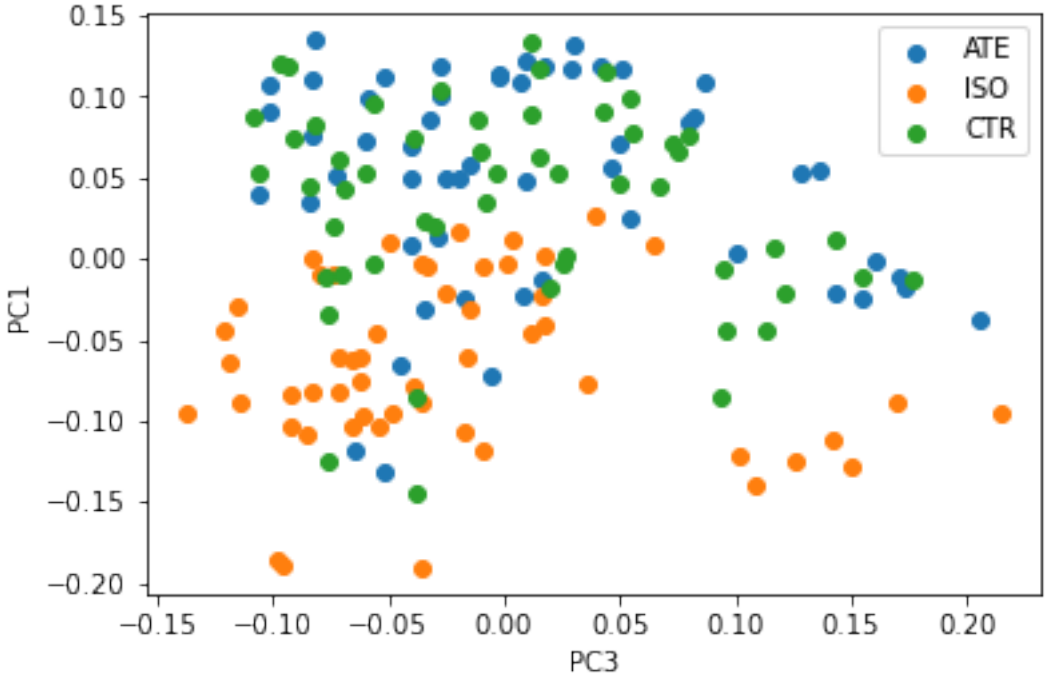
[6]:
```
from itertools import permutations
perm = permutations([0, 1, 2]) # Get all permutations
labels = ["ATE","ISO","CTR"]

for p in perm:
    P = list(p)
    for label in labels:
        idx_list = [i for i in  range(0,len(samples)) if label in samples[i]]
        plt.scatter(pca.components_[P[0]][idx_list],pca.
↪components_[P[1]][idx_list],label=label)

    plt.xlabel("PC%d"%(P[0]+1,))
    plt.ylabel("PC%d"%(P[1]+1,))
    plt.legend()
    plt.show()
    plt.close
```

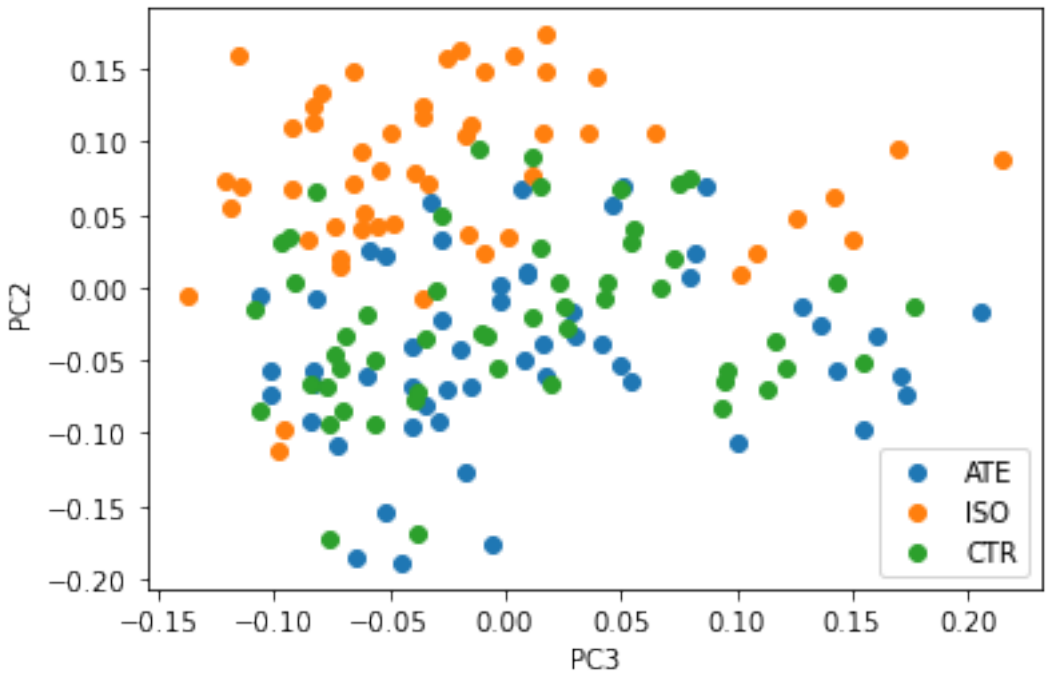



ATE
ISO
CTR
PC3
PC1
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−0.15
−0.10
−0.05
0.00
0.05
0.10
0.15


ATE
ISO
CTR
PC1
PC2
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−0.20
−0.15
−0.10
−0.05
0.00
0.05
0.10
0.15

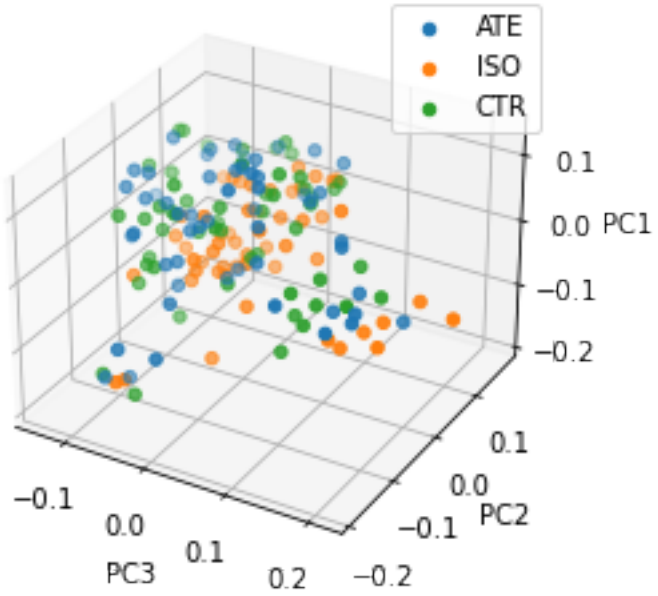

ATE
ISO
CTR
PC3
PC2
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−0.15
−0.10
−0.05
0.00
0.05
0.10
0.15


ATE
ISO
CTR
PC1
PC3
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−0.15
−0.10
−0.05
0.00
0.05
0.10
0.15
0.20



We can also make a 3D plot to visualize all of components at once

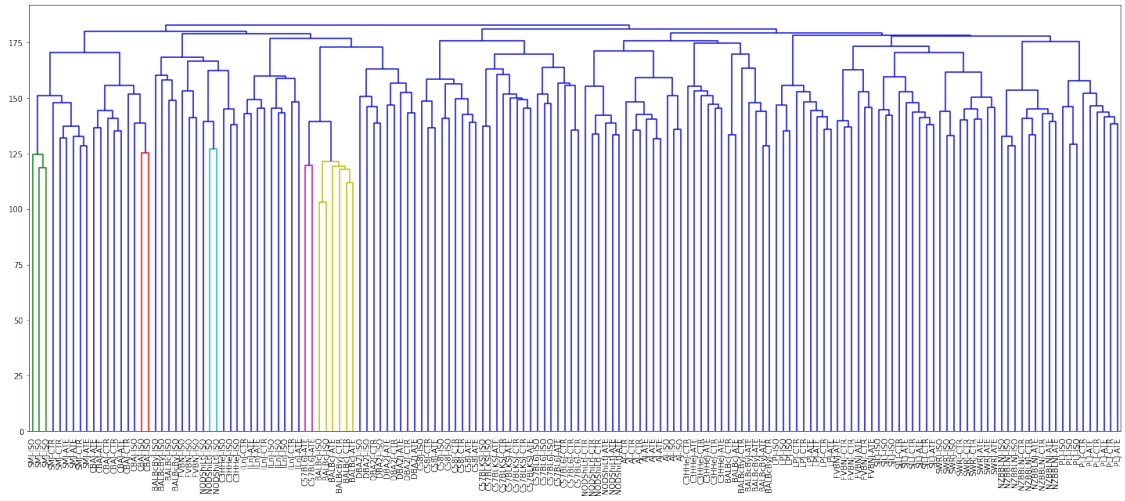
[7]:
```
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
ax = fig.gca(projection='3d')

for label in labels:
    idx_list = [i for i in  range(0,len(samples)) if label in samples[i]]
    ax.scatter(pca.components_[P[0]][idx_list],pca.
 ↪components_[P[1]][idx_list],pca.components_[P[2]][idx_list],label=label)

ax.set_xlabel("PC%d"%(P[0]+1,))
ax.set_ylabel("PC%d"%(P[1]+1,))
ax.set_zlabel("PC%d"%(P[2]+1,))
plt.legend()
plt.show()
```



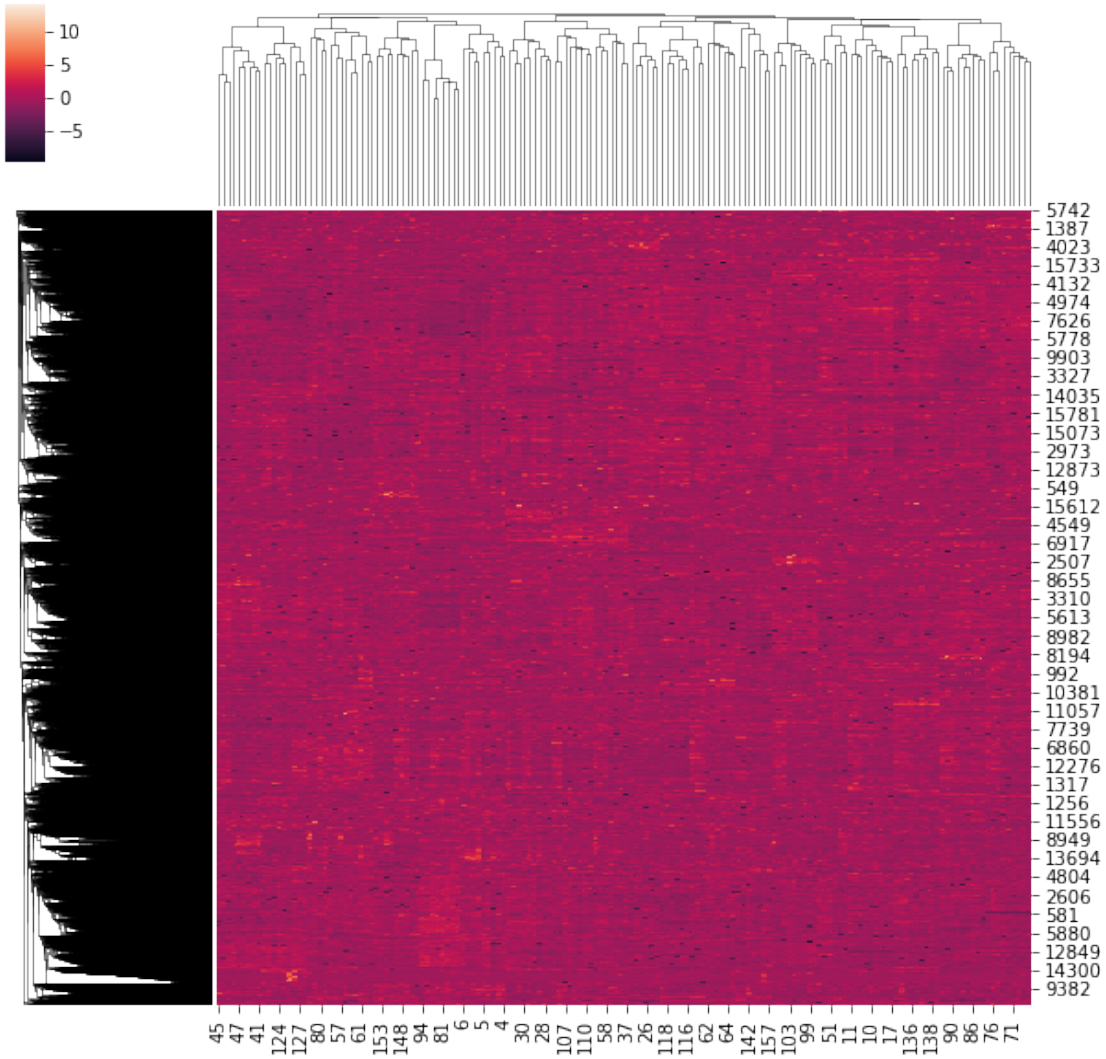
1.3.5 STEP5: perform h-clustering clustering

Perform hierarchical clustering: plot a dendrogram of our expression data. Label the leaves using the sample names: what do they mostly cluster by?

[8]:
```
# Calculate the distance between each sample
Z = linkage(np.transpose(expression_data_normalized), 'average')

# plot the dendogram
fig = plt.figure(figsize=(25, 10))
dn = dendrogram(Z,labels=samples)

ax = plt.gca()
ax.tick_params(axis = 'both', which = 'major', labelsize = 10)

plt.savefig("dendrogram.png",dpi=600)
plt.show()
```



[9]:
```
import seaborn as sns
import pandas as pd

pd.DataFrame(expression_data_normalized,columns=samples,index=genes[:-1])
ax =sns.clustermap(expression_data_normalized)
plt.savefig("dendrogram_heatmap.png",dpi=600)
plt.show()
```



1.3.6 STEP6: perform k-means clustering (Exercise for the students)

Run k-means clustering on expression data. Look again for 3 clusters. Note that, depending in the library you use, you might have to rotate the data to cluster by sample.

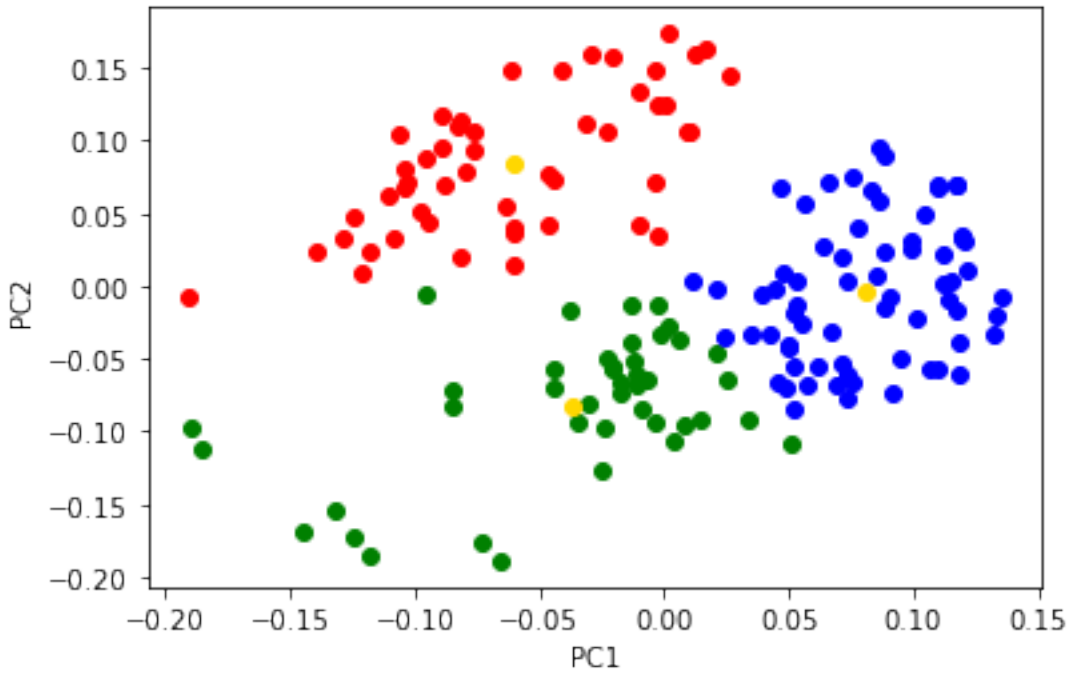
[10]:
```
# kmeans
#set the number of clusters to calculate
kmeans = KMeans(n_clusters=3)
#transform the expression data to their principal components
pca.fit(expression_data_normalized)
pca_data = np.transpose(np.vstack([pca.components_[0],pca.components_[1]]))
#perform the k-means clustering
kmeans.fit(pca_data)
```

[10]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

[ ]:

Repeat the PCA plot above, this time coloring the points according to the labels that were assigned by the clustering algorithm. Does it make sense, visually?

[11]:
```
#perform cluster assignment for the pca data
K = kmeans.predict(pca_data)
#colour the data by the assigned clusters
for i,cluster_value in enumerate(K):
    if cluster_value == 0:
        colour="red"
    elif cluster_value == 1:
        colour="blue"
    elif cluster_value == 2:
        colour="green"
    plt.scatter(pca_data[:,0][i],pca_data[:,1][i],c=colour)
plt.xlabel("PC%d"%(1,))
plt.ylabel("PC%d"%(2,))

#plot centers
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],c='gold')
plt.show()
```



[ ]:

[ ]:

[ ]:

[ ]:

[ ]: