BF 550 Project(Fall 2022)

Library Import

In [1]:
```
import matplotlib.pyplot as plt
import matplotlib
import seaborn as sns
import pandas as pd

from sklearn.decomposition import PCA

import scanpy as sc
import scipy.stats as sps

import scipy.spatial
import scipy.cluster.hierarchy as sch
from scipy.io import mmread
import numpy as np
```

Dendrogram Plotting

In [2]:
```
df = pd.read_csv('DCM_HCM_UMAP_V1.txt',delimiter="\t",low_memory=False)
#Reading the data file which is tab delimited
display(df)
```

	NAME	X	Y	Category
0	TYPE	numeric	numeric	group
1	TTCTTCCGTTCAACGT-1-0	-1.695459008	7.326742649	00.Cardiomyocyte_I
2	CATCCACCATCTAACG-1-0	-1.356442571	7.895590782	00.Cardiomyocyte_I
3	ACCCAAACAGCTAACT-1-0	-1.015783072	8.717870712	00.Cardiomyocyte_I
4	AAGGAATCAACTGGTT-1-0	0.769289255	5.811653137	00.Cardiomyocyte_I
...	...	...	...	...
592685	TTATTGCGTCGGTGTC-1-79	14.41378117	0.419484466	02.Endothelial_I
592686	GTCACGGGTTGTATGC-1-79	14.97959518	0.975733459	02.Endothelial_I
592687	GTCATGATCTTTCGAT-1-79	14.6039629	1.903696299	02.Endothelial_I
592688	GCGATCGTCAGAGTGG-1-79	15.17921162	1.361344814	02.Endothelial_I
592689	GCCCGAACAGAGAAAG-1-79	15.46115494	1.432379961	02.Endothelial_I

592690 rows × 4 columns

In [3]:
```
df_1 = df.drop(labels = [0],axis = 0)
# Cleaning the dataframe to remove the unnecessary components
df_1.drop('NAME', inplace=True, axis=1)
# This column contains teh barcodes which are used to identify the RNA seq expression data. Since it is not needed
#for our analysis, it is dropped along axis 1(rows)
display(df_1)
columns = list(df_1.columns.values)
#This stores the column header values in an array
print(columns)
```

	X	Y	Category
1	-1.695459008	7.326742649	00.Cardiomyocyte_I
2	-1.356442571	7.895590782	00.Cardiomyocyte_I
3	-1.015783072	8.717870712	00.Cardiomyocyte_I
4	0.769289255	5.811653137	00.Cardiomyocyte_I
5	-0.311597943	7.567400455	00.Cardiomyocyte_I
...	...	...	...
592685	14.41378117	0.419484466	02.Endothelial_I
592686	14.97959518	0.975733459	02.Endothelial_I
592687	14.6039629	1.903696299	02.Endothelial_I
592688	15.17921162	1.361344814	02.Endothelial_I
592689	15.46115494	1.432379961	02.Endothelial_I

592689 rows × 3 columns

```
['X', 'Y', 'Category']
```

In [4]:
```
unique = df_1['Category'].unique().tolist()
#Stores the unique values of the categories in a list. This will be used later to create the labels of the dendrogram
unique.sort()
unique_1 = []
for n in range(len(unique)):
    unique_1.append(unique[n][3:])
    #This for loop removes the numbers at the front of the labels for formatting.

print(unique_1)
```

```
['Cardiomyocyte_I', 'Fibroblast_I', 'Endothelial_I', 'Pericyte_I', 'Macrophage', 'VSMC', 'Endothelial_II', 'Lymphocyte', 'Endocardial', 'Adipocyte', 'Neuronal', 'Lymphatic_endot
helial', 'Cardiomyocyte_II', 'Activated_fibroblast', 'Mast_cell', 'Endothelial_III', 'Cardiomyocyte_III', 'Proliferating_macrophage', 'Pericyte_II', 'Epicardial', 'Fibroblast_I
I']
```

In [5]:
```
df_columns = df_1['Category'].copy()
display(df_columns)
df_2 = df_1.apply(pd.to_numeric, errors='coerce')
#This is done to convert numbers which are represented as strings to float values so that we can find the mean of
#the points later
df_2.pop('Category')
#Since the column category only contains strings which could not be converted into float, they are removed.
display(df_2)
df_3 = pd.concat([df_2,df_columns],axis = 1)
display(df_3)
#This is the dataframe upon which we are going to find the center of the clusters.
```

```
1         00.Cardiomyocyte_I
2         00.Cardiomyocyte_I
3         00.Cardiomyocyte_I
4         00.Cardiomyocyte_I
5         00.Cardiomyocyte_I
                 ...
592685      02.Endothelial_I
592686      02.Endothelial_I
592687      02.Endothelial_I
592688      02.Endothelial_I
592689      02.Endothelial_I
Name: Category, Length: 592689, dtype: object
```

	X	Y
1	-1.695459	7.326743
2	-1.356443	7.895591
3	-1.015783	8.717871
4	0.769289	5.811653
5	-0.311598	7.567400
...	...	...
592685	14.413781	0.419484
592686	14.979595	0.975733
592687	14.603963	1.903696
592688	15.179212	1.361345
592689	15.461155	1.432380

592689 rows × 2 columns

	X	Y	Category
1	-1.695459	7.326743	00.Cardiomyocyte_I
2	-1.356443	7.895591	00.Cardiomyocyte_I
3	-1.015783	8.717871	00.Cardiomyocyte_I
4	0.769289	5.811653	00.Cardiomyocyte_I
5	-0.311598	7.567400	00.Cardiomyocyte_I
...	...	...	...
592685	14.413781	0.419484	02.Endothelial_I
592686	14.979595	0.975733	02.Endothelial_I
592687	14.603963	1.903696	02.Endothelial_I
592688	15.179212	1.361345	02.Endothelial_I
592689	15.461155	1.432380	02.Endothelial_I

592689 rows × 3 columns

In [6]:
```
# df_1.groupby(by='Category')[['X','Y']].mean()
# new_df['Category'] = ['00.Cardiomyocyte_I', '09.Adipocyte', '17.Proliferating_macrophage', '16.Cardiomyocyte_III', '04.Macrophage', '08.Endocardial', '01.Fibroblast_I', '15.En

new_df = pd.DataFrame(columns = ['X','Y','Category'])



for i in unique:
    #This entire for loop iterates through the list and gets the categories which are clustered together, extracts
    #the rows of the categories. It then finds the center of the clusters by finding out the mean of all the points
    #inside the columns. The resulting data is stored in a new dataframe.
    df_indv = df_3[df_3['Category'] == i]
    x_mean = df_indv['X'].mean()
    new_df.loc[i,['X']] = [x_mean]
    y_mean = df_indv['Y'].mean()
    new_df.loc[i,['Y']] = [y_mean]


new_df.pop('Category')
new_df.reset_index(inplace = True)
#Since the index contains the names of the categories we are resetting the index to numbers and moving the category
#names to a new column
new_df.rename(columns = {'index':'Category'},inplace = True)
#The column containing the category is renamed to our preference.
display(new_df)
```

	Category	X	Y
0	00.Cardiomyocyte_I	-0.83819	6.505634
1	01.Fibroblast_I	7.939621	10.103026
2	02.Endothelial_I	13.471817	1.370661
3	03.Pericyte_I	13.407958	15.58621
4	04.Macrophage	5.162196	-2.97554
5	05.VSMC	14.777002	11.927519
6	06.Endothelial_II	11.648213	2.980964
7	07.Lymphocyte	10.79959	-4.380344
8	08.Endocardial	8.422043	3.468433
9	09.Adipocyte	1.42544	13.964581
10	10.Neuronal	5.104368	2.94464
11	11.Lymphatic_endothelial	17.099502	6.410905
12	12.Cardiomyocyte_II	-0.369741	3.411075
13	13.Activated_fibroblast	7.204062	7.015108
14	14.Mast_cell	14.075913	7.217169
15	15.Endothelial_III	10.311464	-0.596766
16	16.Cardiomyocyte_III	2.292647	7.242772
17	17.Proliferating_macrophage	3.436537	0.784914
18	18.Pericyte_II	10.974208	15.614097
19	19.Epicardial	4.721231	4.767997
20	20.Fibroblast_II	9.952259	11.409339

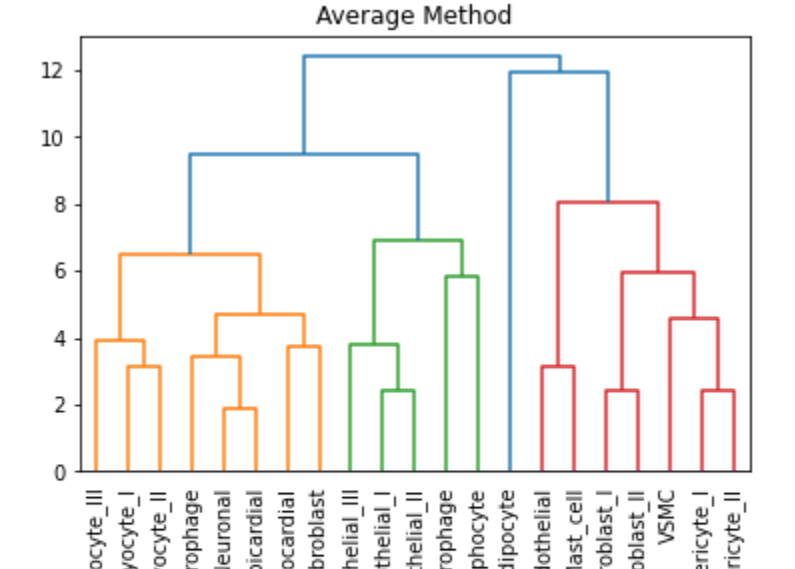
In [7]:
```
change_df = new_df.copy()
nparray = change_df.to_numpy(copy = True)
#A numpy array is created from the dataframe which is used to build the dendrogram. This is done because the scipy
#linkage and dendrogram functions only take arrays as inputs.
print(nparray)
indices = nparray[:,1:3]
#Slicing through the array is necessary to remove the category names from the arrays.
print("\n")
print(indices)
```

```
[['00.Cardiomyocyte_I' -0.8381899314133306 6.505633928124355]
 ['01.Fibroblast_I' 7.939620541154109 10.103026432792317]
 ['02.Endothelial_I' 13.471817428514083 1.3706608969093788]
 ['03.Pericyte_I' 13.407958481535859 15.586210278709537]
 ['04.Macrophage' 5.162196109078242 -2.975540299170163]
 ['05.VSMC' 14.777002080391615 11.927519387221905]
 ['06.Endothelial_II' 11.648213096675892 2.980963646883369]
 ['07.Lymphocyte' 10.799590225597 -4.380344053033414]
 ['08.Endocardial' 8.422042765554957 3.468433048983353]
 ['09.Adipocyte' 1.4254400901828979 13.964581056928452]
 ['10.Neuronal' 5.104368347484154 2.94463987031734]
 ['11.Lymphatic_endothelial' 17.09950207322814 6.41090470928083]
 ['12.Cardiomyocyte_II' -0.3697407701110705 3.411074691333036]
 ['13.Activated_fibroblast' 7.2040617696800435 7.015107623364509]
 ['14.Mast_cell' 14.075913464812736 7.2171685233926]
 ['15.Endothelial_III' 10.311463562417137 -0.5967655951172314]
 ['16.Cardiomyocyte_III' 2.292646860738793 7.24277197958837]
 ['17.Proliferating_macrophage' 3.4365368035282144 0.7849139442648899]
 ['18.Pericyte_II' 10.974207795897902 15.61409735788145]
 ['19.Epicardial' 4.721231384338293 4.76799744394052]
 ['20.Fibroblast_II' 9.952259340588029 11.40933890402464]]


[[-0.8381899314133306 6.505633928124355]
 [7.939620541154109 10.103026432792317]
 [13.471817428514083 1.3706608969093788]
 [13.407958481535859 15.586210278709537]
 [5.162196109078242 -2.975540299170163]
 [14.777002080391615 11.927519387221905]
 [11.648213096675892 2.980963646883369]
 [10.799590225597 -4.380344053033414]
 [8.422042765554957 3.468433048983353]
 [1.4254400901828979 13.964581056928452]
 [5.104368347484154 2.94463987031734]
 [17.09950207322814 6.41090470928083]
 [-0.3697407701110705 3.411074691333036]
 [7.2040617696800435 7.015107623364509]
 [14.075913464812736 7.2171685233926]
 [10.311463562417137 -0.5967655951172314]
 [2.292646860738793 7.24277197958837]
 [3.4365368035282144 0.7849139442648899]
 [10.974207795897902 15.61409735788145]
 [4.721231384338293 4.76799744394052]
 [9.952259340588029 11.40933890402464]]
```

Dendrogram

In [8]:
```
cluster = sch.linkage(indices, method = 'average', metric = 'euclidean')
#This creates the linkage matrix upon which the dendrogram is generated
#Method describes the clustering method ued to create the linkage matrix,
#Metric describes how the distance between the points is measured
dendrogram = sch.dendrogram(cluster,labels = unique_1, leaf_rotation = 90)
#This method takes the linkage matrix as the input
#Labels takes the cluster names as the input
#Leaf_rotation rotates the leaves(Clusternames) to 90 degrees to align the names properly
plt.title(label = "Average Method", loc = 'center')
plt.show()
```



Different methods to obtain Dendrogram Clustering

In [9]:
```
# methods = ['single','complete','average','weighted','centroid','median','ward']

# for met in methods:
#     cluster = sch.linkage(indices, method = met, metric = 'euclidean')
#     dendrogram = sch.dendrogram(cluster,labels = unique_1, leaf_rotation = 90)
#     plt.title(label = "%s method"%met , loc = 'center')
#     plt.show()
```

Heatmap with Hierarchial Clustering

In [10]:
```
heatmap_df = pd.read_excel('/Users/aravind/Documents/Bioinformatics/BU/SEMESTER 1/BF 550-Foundations of Programming, Data Analytics, and Machine Learning in Python /Project/Fina
#This creates a dataframe of the data provided for the image.
# heatmap_df[]
heatmap_df = heatmap_df.rename({'Unnamed: 0': 'Genes'}, axis=1)
#Renaming one column to our required name
Gene_names = heatmap_df['Genes']
#This stores the gene names in a list
column_headers = list(heatmap_df.columns.values)
column_headers = column_headers[1:]
#These 2 lines of code are used to store the names of the various cell clusters in a list
# print(column_headers)
# print(Gene_names)
display(heatmap_df)
```

	Genes	Cardiomyocyte_III	Cardiomyocyte_I	Cardiomyocyte_II	VSMC	Pericyte_I	Pericyte_II	Neuronal	Activated fibroblast	Fibroblast_I	...	Epicardial	Endothelial_III	Endothelial_I	Endothelial_II	Endocardial	Lymphatic endothelial
0	PALLD	2.081803	2.611033	1.868134	-0.286147	-0.552931	-0.551529	-0.163047	1.229281	-0.185865	...	-0.377395	-0.512821	-0.556436	-0.555582	-0.518936	-0.491756
1	FHL2	1.187615	3.001720	2.633896	-0.399789	-0.426421	-0.426459	-0.285226	-0.317562	-0.315525	...	-0.272084	-0.414071	-0.424022	-0.423723	-0.388428	-0.344860
2	RBM20	1.344881	3.088095	2.448901	-0.320980	-0.406693	-0.407267	-0.309930	-0.409847	-0.409030	...	-0.312532	-0.398238	-0.409299	-0.408639	-0.395725	-0.359980
3	MYBPC3	1.300055	3.067276	2.497157	-0.387826	-0.412629	-0.412943	-0.286762	-0.412612	-0.410966	...	-0.243416	-0.403109	-0.410218	-0.411299	-0.395334	-0.357953
4	PPP1R12B	1.763071	2.710385	2.156527	0.644354	-0.448378	-0.407939	-0.161711	-0.601028	-0.563182	...	-0.244047	-0.558405	-0.616489	-0.598375	-0.385101	-0.487072
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1995	MEG8	-0.071268	-0.560811	-0.390526	-0.534607	-0.568176	-0.577795	-0.128752	2.167660	2.480559	...	-0.069892	-0.523580	-0.586475	-0.576601	-0.477875	-0.251374
1996	ELN	-0.668504	-0.800136	-0.706989	0.788569	-0.744488	-0.761642	0.126333	-0.083010	1.355893	...	-0.537896	2.018803	-0.617193	-0.369297	1.334561	0.944773
1997	AKAP12	-0.739075	-0.905226	-0.785868	0.720659	-0.211477	-0.281758	0.560146	0.137753	0.736112	...	-0.674273	-0.513816	-0.177869	-0.111500	-0.248941	2.699206
1998	CHSY3	-0.202673	-1.168057	-0.937781	-0.493517	-0.584144	-0.426690	-0.442634	1.811530	1.406585	...	0.413412	-0.059866	1.336282	0.891337	-0.420059	2.292320
1999	EMP1	-0.956859	-1.088081	-0.996477	-1.024483	-0.994251	-1.001287	-0.519634	1.553497	0.434895	...	-0.026341	1.429614	0.617526	0.394702	0.409425	0.552970

2000 rows × 22 columns

In [11]:
```
heatmap_df.pop('Genes')
#In order to convert the given data into a numpy array which can be taken as an input by the clustemap function of
#sns.clustermap we have to remove the gene column
vectors = heatmap_df.values
#Here we are converting the data frame into a numpy array
vectors
```

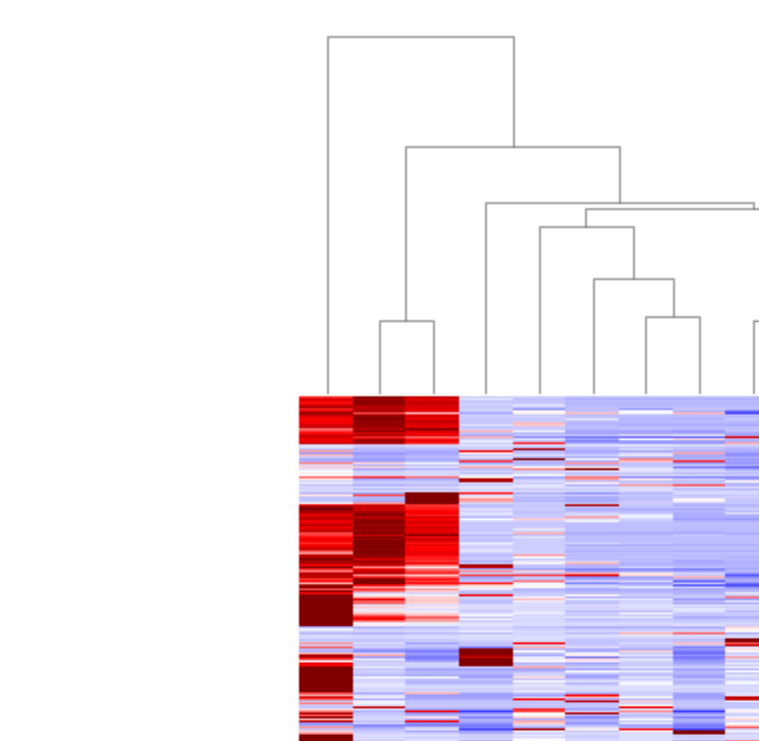
Out[11]:
```
array([[ 2.08180308,  2.6110332 ,  1.86813402, ..., -0.51022863,
        -0.55878013, -0.46063775],
       [ 1.18761468,  3.00172019,  2.63389564, ..., -0.40843776,
        -0.42640835, -0.4070957 ],
       [ 1.3448813 ,  3.08809543,  2.44890094, ..., -0.39262128,
        -0.41042536, -0.39840686],
       ...,
       [-0.73907477, -0.90522557, -0.78586763, ..., -0.81004292,
        -0.8947069 , -0.77949572],
       [-0.2026729 , -1.16805685, -0.93778145, ..., -0.46667305,
        -1.04746318, -0.98668885],
       [-0.95685887, -1.08808064, -0.99647665, ...,  0.41957691,
        -1.04981196, -0.96973634]])
```

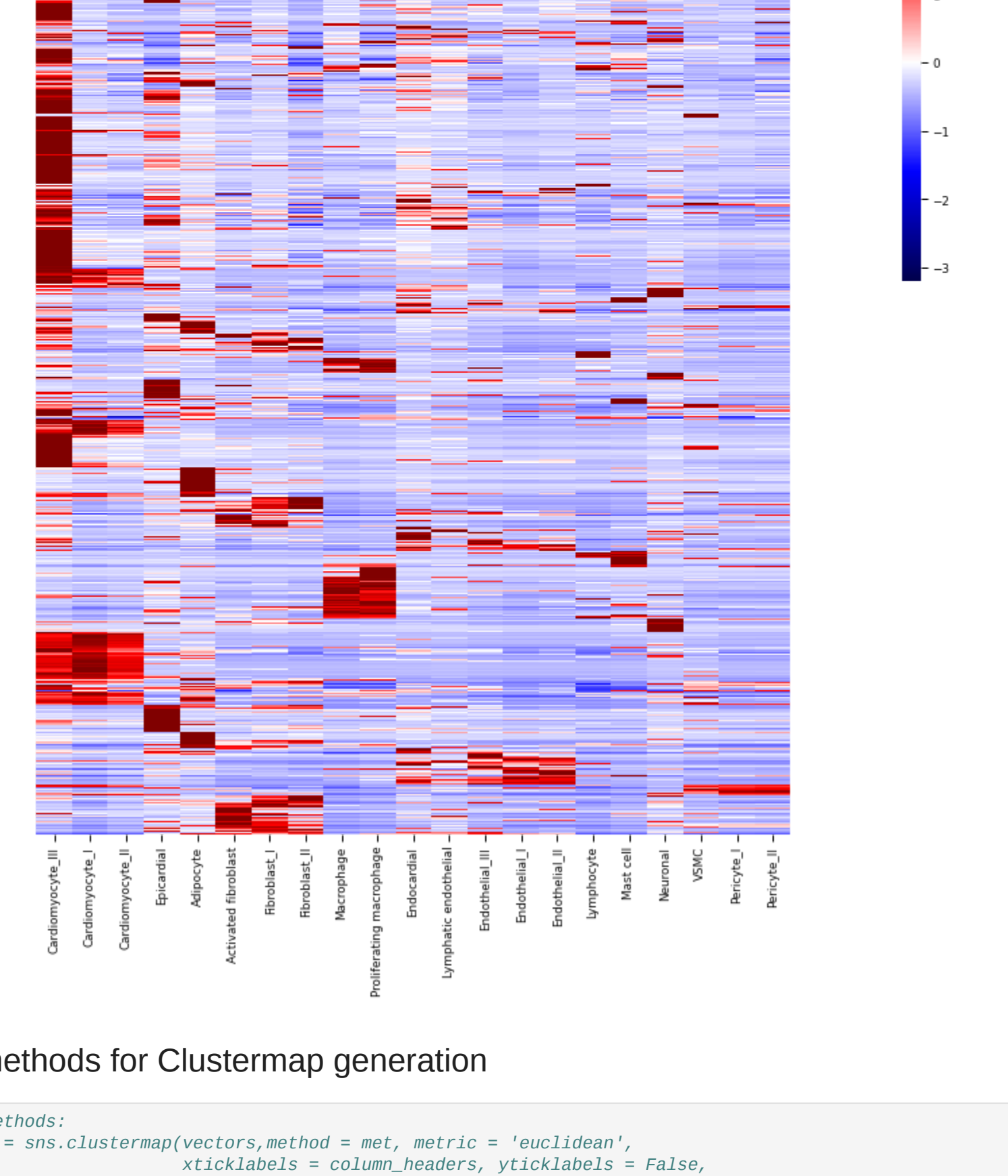
In [12]:
```
sns.set_context("paper",font_scale = 1)
#This sets the font size of the letterings on the image
```

Best Method for Clustermap Recreation

In [13]:
```
Z_columns = sch.linkage(np.transpose(vectors),method = 'ward' ,metric='euclidean')
#This is done to create a dendrogram which is identical to the one produced in the clustermap.
#The vector data has to be transposed in order to create a linkage matrix upon which the dendrogram is generated
dendrogram = sch.dendrogram(Z_columns,labels = column_headers, leaf_rotation = 90)
#This method generates the dendrograms.

sns_plot = sns.clustermap(vectors,method = 'ward', metric = 'euclidean',
                          xticklabels = column_headers, yticklabels = False,
                          row_cluster = False, col_linkage = Z_columns,
                          cmap="seismic", cbar_pos = [1.1, 0.5, 0.02, 0.3],
                          vmin = -3.2, vmax = 3.2, figsize = (10,15))
#This line of code generates the heatmap with the clustering of the variables. Each of the individual parameters
#allow us to modify the plot to generate an image similar to the one in the paper.
#xticklabels and yticklabels takes lists which contain the column and row headers as input and display them in the
#plot
#col_linkage allows us to use a linkage matrix which has already been generated and use it in the clustermap
#cmap allows us to use a predefined colour scale in the clustermap
#cbar_pos is used to define the position of the colorbar
#vmin and vmax defines the minimum and maximum limit of the colorbar
#figsize determines the size of the clustermap image
plt.title("Z-score Expression")
plt.show()
```





Different methods for Clustermap generation

In [14]:
```
# for met in methods:
#     sns_plot = sns.clustermap(vectors,method = met, metric = 'euclidean',
#                               xticklabels = column_headers, yticklabels = False,
#                               row_cluster = False, cmap="seismic", cbar_pos = [1.1, 0.5, 0.02, 0.3],
#                               vmin = -3, vmax = 3)
```

Clustermap with clustering done on Y axis(Gene Names)

In [15]:
```
# sns_plot = sns.clustermap(vectors,method = 'ward', metric = 'euclidean',
#                           xticklabels = column_headers, yticklabels = False,
#                           row_cluster = True, col_linkage = Z_columns,
#                           cmap="seismic", cbar_pos = [1.1, 0.5, 0.02, 0.3],
#                           vmin = -3.2, vmax = 3.2, figsize = (10,15))
# plt.title("Z-score Expression")
# plt.show()
```