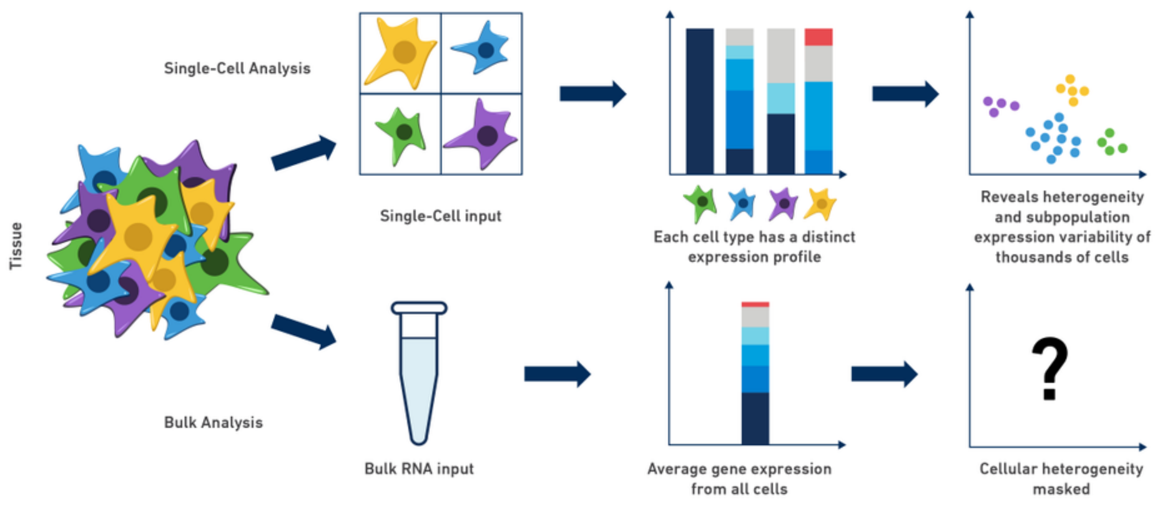
Homework 3: Unsupervised Learning

Due Wednesday 11/24 at 11:59 pm EST

In this notebook, we will be applying unsupervised learning approaches to a problem in computational biology. Specifically, we will be analyzing single-cell genomic sequencing data. Single-cell genomics is a set of revolutionary new technologies which can profile the genome of a specimen (tissue, blood, etc.) at the resolution of individual cells. This increased granularity can help capture intercellular heterogeneity, key to better understanding and treating complex genetic diseases such as cancer and Alzheimer's.


Single-Cell Analysis
Tissue
Single-Cell input
Each cell type has a distinct expression profile
Reveals heterogeneity and subpopulation expression variability of thousands of cells
Bulk Analysis
Bulk RNA input
Average gene expression from all cells
?
Cellular heterogeneity masked

Source: 10xgenomics.com/blog/single-cell-rna-seq-an-introductory-overview-and-tools-for-getting-started

A common challenge of genomic datasets is their high-dimensionality: a single observation (a cell, in the case of single-cell data) may have tens of thousands of gene expression features. Fortunately, biology offers a lot of structure - different genes work together in pathways and are co-regulated by gene regulatory networks. Unsupervised learning is widely used to discover this intrinsic structure and prepare the data for further analysis.

In [154]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
```

Dataset: single-cell RNASeq of mouse brain cells

We will be working with a single-cell RNASeq dataset of mouse brain cells. In the following gene expression matrix, each row represents a cell and each column represents a gene. Each entry in the matrix is a normalized gene expression count - a higher value means that the gene is expressed more in that cell. The dataset has been pre-processed using various quality control and normalization methods for single-cell data.

Data source: https://chanzuckerberg.github.io/scRNA-python-workshop/preprocessing/00-tabula-muris.html (https://chanzuckerberg.github.io/scRNA-python-workshop/preprocessing/00-tabula-muris.html)

```
In [156]: cell_gene_counts_df = pd.read_csv('data/mouse_brain_cells_gene_counts.csv',
          cell_gene_counts_df
```

Out[156]:

	0610005C13Rik	0610007C21Rik	0610007L01Rik	0610007N19Rik	0610007P
cell					
A1.B003290.3_38_F.1.1	-0.08093	0.7856	1.334	-0.2727	-C
A1.B003728.3_56_F.1.1	-0.08093	-1.4840	-0.576	-0.2727	-C
A1.MAA000560.3_10_M.1.1	-0.08093	0.6300	-0.576	-0.2727	-C
A1.MAA000564.3_10_M.1.1	-0.08093	0.3809	1.782	-0.2727	-C
A1.MAA000923.3_9_M.1.1	-0.08093	0.5654	-0.576	-0.2727	-C
...	...	...	...	...	
E2.MAA000902.3_11_M.1.1	14.98400	1.1550	-0.576	-0.2727	-C
E2.MAA000926.3_9_M.1.1	-0.08093	-1.4840	-0.576	-0.2727	-C
E2.MAA000932.3_11_M.1.1	-0.08093	0.5703	-0.576	-0.2727	-C
E2.MAA000944.3_9_M.1.1	-0.08093	0.3389	-0.576	-0.2727	-C
E2.MAA001894.3_39_F.1.1	-0.08093	0.3816	-0.576	-0.2727	-C

1000 rows × 18585 columns

Note the dimensionality - we have 1000 cells (observations) and 18,585 genes (features)!

We are also provided a metadata file with annotations for each cell (e.g. cell type, subtissue, mouse sex, etc.)

In [157]:
```
cell_metadata_df = pd.read_csv('data/mouse_brain_cells_metadata.csv')
cell_metadata_df
```

Out[157]:

	cell	cell_ontology_class	subtissue	mouse.sex	mouse.id	plate.barcod
0	A1.B003290.3_38_F.1.1	astrocyte	Striatum	F	3_38_F	B00329
1	A1.B003728.3_56_F.1.1	astrocyte	Striatum	F	3_56_F	B00372
2	A1.MAA000560.3_10_M.1.1	oligodendrocyte	Cortex	M	3_10_M	MAA00056
3	A1.MAA000564.3_10_M.1.1	endothelial cell	Striatum	M	3_10_M	MAA00056
4	A1.MAA000923.3_9_M.1.1	astrocyte	Hippocampus	M	3_9_M	MAA00092
...	...	...	...	...	...	.
995	E2.MAA000902.3_11_M.1.1	astrocyte	Striatum	M	3_11_M	MAA00090
996	E2.MAA000926.3_9_M.1.1	oligodendrocyte	Cortex	M	3_9_M	MAA00092
997	E2.MAA000932.3_11_M.1.1	endothelial cell	Hippocampus	M	3_11_M	MAA00093
998	E2.MAA000944.3_9_M.1.1	oligodendrocyte	Cortex	M	3_9_M	MAA00094
999	E2.MAA001894.3_39_F.1.1	oligodendrocyte	Cortex	F	3_39_F	MAA00189

1000 rows × 8 columns

Different cell types

In [158]:
```
cell_metadata_df['cell_ontology_class'].value_counts()
```

Out[158]:
```
oligodendrocyte                      385
endothelial cell                     264
astrocyte                            135
neuron                                94
brain pericyte                        58
oligodendrocyte precursor cell        54
Bergmann glial cell                   10
Name: cell_ontology_class, dtype: int64
```

Different subtissue types (parts of the brain)

In [159]:
```
cell_metadata_df['subtissue'].value_counts()
```

Out[159]:
```
Cortex         364
Hippocampus    273
Striatum       220
Cerebellum     143
Name: subtissue, dtype: int64
```

In [160]:
```
cell_metadata_df['mouse.id'].value_counts()
```

Out[160]:
```
3_10_M    273
3_9_M     226
3_38_F    178
3_8_M     171
3_11_M     72
3_39_F     57
3_56_F     23
Name: mouse.id, dtype: int64
```

Our goal in this exercise is to use dimensionality reduction and clustering to visualize and better understand the high-dimensional gene expression matrix. We will use the following pipeline, which is common in single-cell analysis:

1. Use PCA to project the gene expression matrix to a lower-dimensional linear subspace.
2. Cluster the data using K-means on the first 20 principal components.
3. Use t-SNE to project the first 20 principal components onto two dimensions. Visualize the points and color by their clusters from (2).

Part 1: PCA

Perform PCA and project the gene expression matrix onto its first 50 principal components. You may use `sklearn.decomposition.PCA`.

In [161]:
```
pca = PCA(n_components=50)
principalComponents = pca.fit_transform(cell_gene_counts_df)
principalDf = pd.DataFrame(data = principalComponents)
principalDf.head()
```

Out[161]:

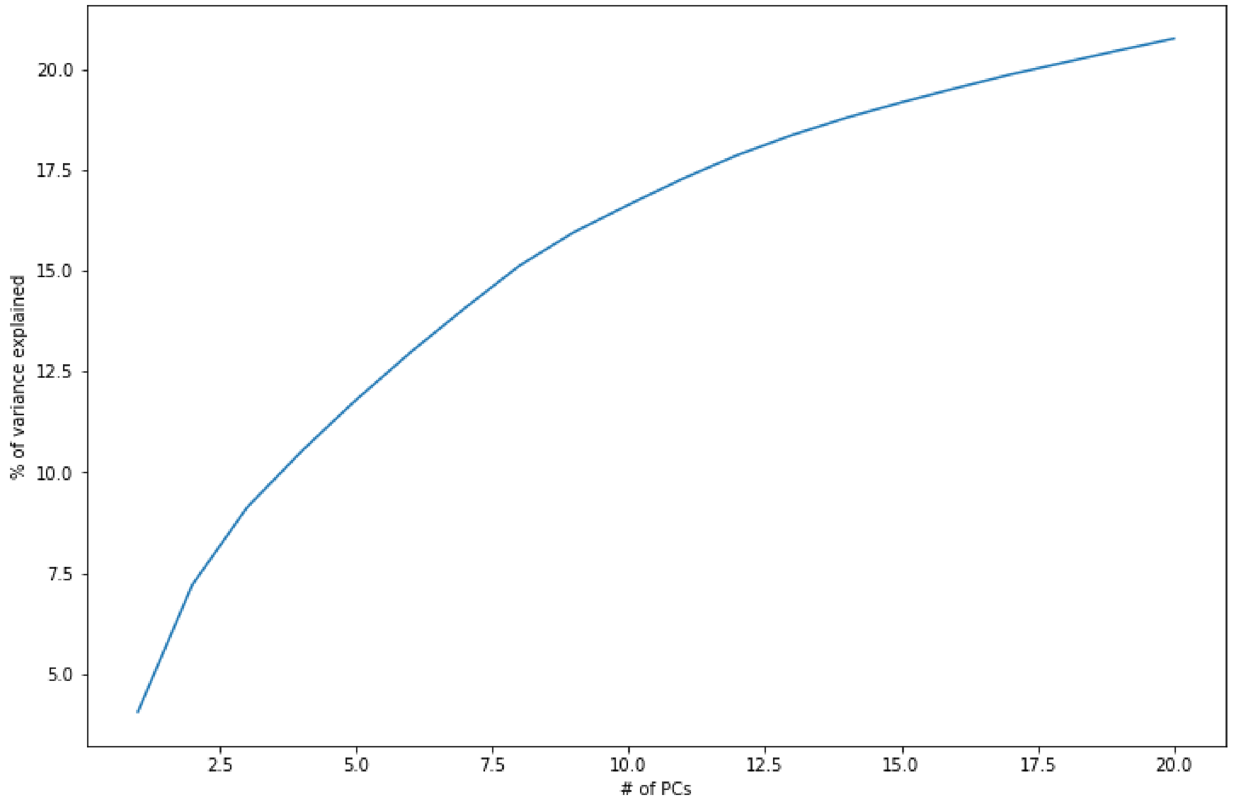
	0	1	2	3	4	5	6	7	
0	15.353967	22.551441	28.909568	18.160745	-63.669873	63.397364	22.120374	193.168096	5.0
1	-19.092789	-3.011189	37.073015	-7.781964	-0.324304	-5.520997	1.450257	-0.053576	-2.1
2	1.624026	-26.093832	-8.735882	1.431624	3.908803	-0.872088	-2.047059	2.420199	3.5
3	-15.469770	37.906454	-37.408305	5.952024	-10.229878	4.293262	15.286237	-4.262438	-6.7
4	-15.223271	-2.999145	38.531674	-6.379690	-6.113619	-4.637018	5.044909	-2.089756	-6.8

5 rows × 50 columns

Plot the cumulative proportion of variance explained as a function of the number of principal components. How much of the total variance in the dataset is explained by the first 20 principal components?

In [162]:
```
fig = plt.figure(figsize=(12,8))
top_20_pca_var = pca.explained_variance_ratio_[:20]
ax = fig.add_subplot(1,1,1)
plt.plot(np.arange(1,21), top_20_pca_var.cumsum()*100)
ax.set_xlabel("# of PCs")
ax.set_ylabel("% of variance explained")

#A little over 20% of the data is explained by the first 20 components
```

Out[162]:
```
Text(0, 0.5, '% of variance explained')
```



For the first principal component, report the top 10 loadings (weights) and their corresponding gene names. In other words, which 10 genes are weighted the most in the first principal component?

In [163]:

```
weights = pca.components_
first_component = abs(weights[0])
ind = np.argpartition(first_component, -10)[-10:]

col_names = cell_gene_counts_df.columns[ind]
col_names
print(col_names)
#'Erc2', 'Cpne5', 'Hpca', 'Nrsn2', 'Camkv', 'Nsg2', 'Rasgef1a', 'Kcnj4',
#       'Ptpn5', 'St8sia3'
```

```
Index(['Erc2', 'Cpne5', 'Hpca', 'Nrsn2', 'Camkv', 'Nsg2', 'Rasgef1a', 'Kc
nj4',
       'Ptpn5', 'St8sia3'],
      dtype='object')
```

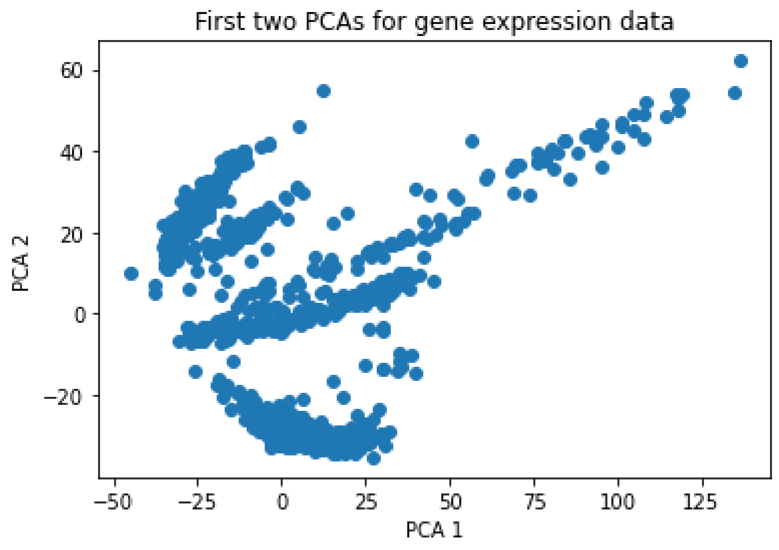
Plot the projection of the data onto the first two principal components using a scatter plot.

In [165]:

```
plt.scatter(x=principalDf[0], y=principalDf[1])
plt.xlabel("PCA 1")
plt.ylabel("PCA 2")
plt.title('First two PCAs for gene expression data')
```

Out[165]:

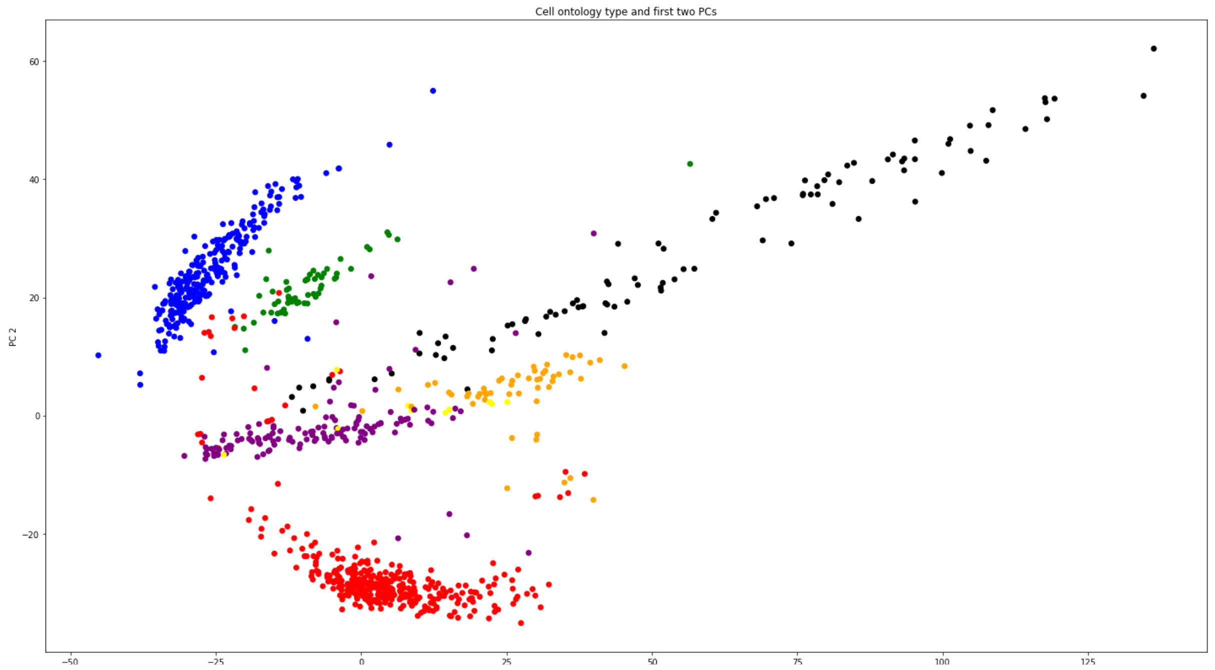
```
Text(0.5, 1.0, 'First two PCAs for gene expression data')
```



Now, use a small multiple of four scatter plots to make the same plot as above, but colored by four annotations in the metadata: cell_ontology_class, subtissue, mouse.sex, mouse.id. Include a legend for the labels. For example, one of the plots should have points projected onto PC 1 and PC 2, colored by their cell_ontology_class.

In [166]:
```
fig, (ax1, ax2, ax3, ax4) = plt.subplots(4, figsize=(20,45))
cell_ontology_colors = ['red', 'blue', 'yellow', 'purple', 'black', 'green'
cell_ontology_colors_map = {
  "oligodendrocyte": "red",
  "endothelial cell": "blue",
  "astrocyte": "purple",
  "neuron": "black",
  "brain pericyte": "green",
  "oligodendrocyte precursor cell": "orange",
  "Bergmann glial cell": "yellow",
}

mouse_sex_colors_map = {
  "F": "red",
  "M": "blue"
}

mouse_id_colors_map = {
  "3_10_M": "red",
  "3_9_M": "blue",
  "3_38_F": "purple",
  "3_8_M": "black",
  "3_11_M": "green",
  "3_39_F": "orange",
  "3_56_F": "yellow",
}

cell_subtissue_colors_map = {
  "Cortex": "red",
  "Hippocampus": "blue",
  "Striatum": "purple",
  "Cerebellum": "black",
}

cell_metadata_df['ontology_color'] = cell_metadata_df.apply(lambda row : ce
cell_metadata_df['sex_color'] = cell_metadata_df.apply(lambda row : mouse_s
cell_metadata_df['id_color'] = cell_metadata_df.apply(lambda row : mouse_id
cell_metadata_df['subtissue_color'] = cell_metadata_df.apply(lambda row : c

ax1.scatter(x=principalDf[0], y=principalDf[1], c=cell_metadata_df['ontolog
ax1.set_xlabel("PC 1")
ax1.set_ylabel("PC 2")
ax1.set_title("Cell ontology type and first two PCs")
ax2.scatter(x=principalDf[0], y=principalDf[1], c=cell_metadata_df['sex_col
ax2.set_xlabel("PC 1")
ax2.set_ylabel("PC 2")
ax2.set_title("Mouse.sex and first two PCs")
ax3.scatter(x=principalDf[0], y=principalDf[1], c=cell_metadata_df['id_colo
ax3.set_xlabel("PC 1")
ax3.set_ylabel("PC 2")
ax3.set_title("Mouse.id and first two PCs")
ax4.scatter(x=principalDf[0], y=principalDf[1], c=cell_metadata_df['subtiss
ax4.set_xlabel("PC 1")
ax4.set_ylabel("PC 2")
ax3.set_title("subtissue and first two PCs")
```

```
fig.tight_layout()


#plt.xlabel("PC 1")
#plt.ylabel("PC 2")
```



Based on the plots above, the first two principal components correspond to which aspect of the cells? What is the intrinsic dimension that they are describing?

In [167]:
```
#PC1 and PC2 are able to distinguish well between cell ontology type
#and M vs Female best, so I think that these two components correpond to
#these aspects of the cells
```

Part 2: K-means

While the annotations provide high-level information on cell type (e.g. cell_ontology_class has 7 categories), we may also be interested in finding more granular subtypes of cells. To achieve this, we will use K-means clustering to find a large number of clusters in the gene expression dataset. Note that the original gene expression matrix had over 18,000 noisy features, which is not ideal for clustering. So, we will perform K-means clustering on the first 20 principal components of the dataset.

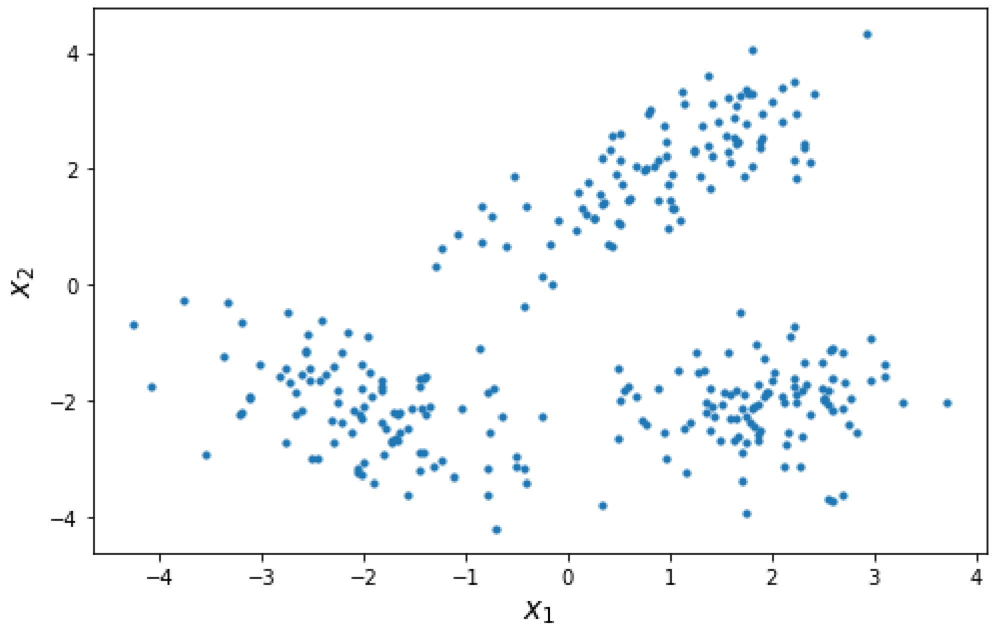
Implement a `kmeans` function which takes in a dataset `X` and a number of clusters `k`, and returns the cluster assignment for each point in `X`. You may NOT use sklearn for this implementation. Use lecture 6, slide 14 as a reference.

In [133]:
```
import random
from scipy.spatial import distance

def kmeans(X, k, iters=10):
    '''Groups the points in X into k clusters using the K-means algorithm.

    Parameters
    ----------
    X : (m x n) data matrix
    k: number of clusters
    iters: number of iterations to run k-means loop

    Returns
    -------
    y: (m x 1) cluster assignment for each point in X
    '''
    #random var between min and max of each col
    #k_vals: (k x n)


    #print(n)
    #should be 20


    #k_vals = np.zeros((k, n))
    #for i in range(0, k):
    #    for j in range (0, n):
    #        min_val_col = int(round(np.min(X[:, j])))
    #        max_val_col = int(round(np.max(X[:, j])))
    #        k_vals[i][j] = random.randint(min_val_col, max_val_col)

    count = 0
    m = len(X)
    idx = np.random.choice(m, k, replace=False)
    n = len(X[0])
    centroids = X[idx, :]
    distances = distance.cdist(X, centroids, 'euclidean')
    min_ks = np.array([np.argmin(i) for i in distances])

    while count < iters:
        centroids = []
        for idx in range(k):
            centroids.append(X[min_ks==idx].mean(axis=0))

        centroids = np.vstack(centroids)
        distances = distance.cdist(X, centroids ,'euclidean')
        min_ks = np.array([np.argmin(i) for i in distances])

        count = count +1

    return min_ks
```

Before applying K-means on the gene expression data, we will test it on the following synthetic dataset to make sure that the implementation is working.

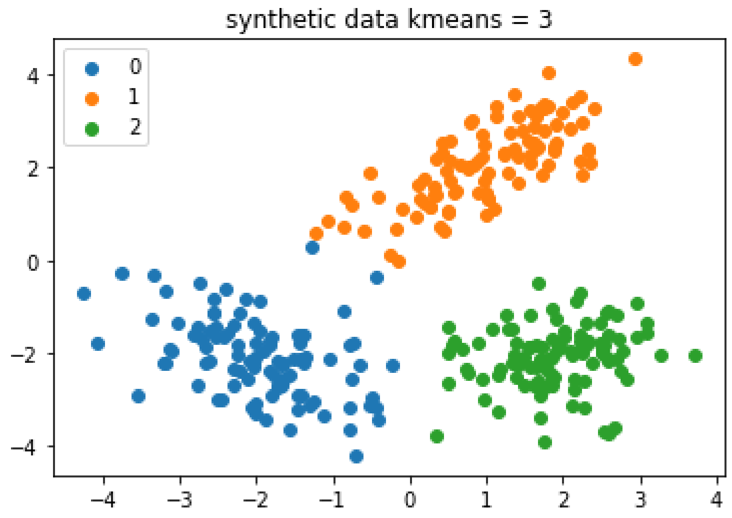
In [134]:
```
np.random.seed(0)
x_1 = np.random.multivariate_normal(mean=[1, 2], cov=np.array([[0.8, 0.6],
x_2 = np.random.multivariate_normal(mean=[-2, -2], cov=np.array([[0.8, -0.4
x_3 = np.random.multivariate_normal(mean=[2, -2], cov=np.array([[0.4, 0], [
X = np.vstack([x_1, x_2, x_3])
plt.figure(figsize=(8, 5))
plt.scatter(X[:, 0], X[:, 1], s=10)
plt.xlabel('$x_1$', fontsize=15)
plt.ylabel('$x_2$', fontsize=15)
```

Out[134]:
```
Text(0, 0.5, '$x_2$')
```



Apply K-means with k=3 to the synthetic dataset above. Plot the points colored by their K-means cluster assignments to verify that your implementation is working.

In [168]:
```
label = kmeans(X, 3, 25)

for i in np.unique(label):
    plt.scatter(X[label == i , 0] , X[label == i , 1] , label = i)
plt.legend()
plt.title('synthetic data kmeans = 3')
plt.show()
```



Use K-means with k=20 to cluster the first 20 principal components of the gene expression data.

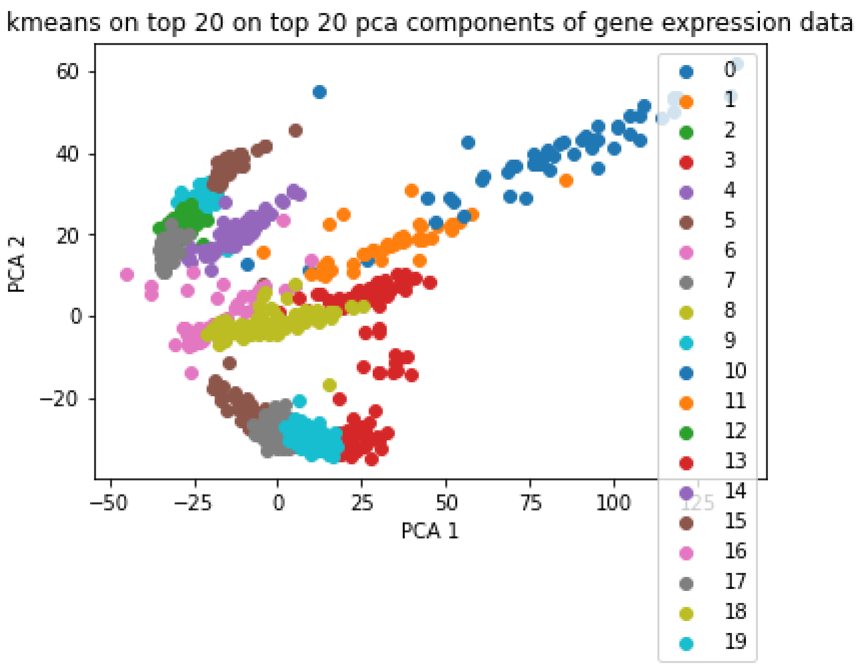
In [145]:
```
pca = PCA(n_components=20)
principalComponents = pca.fit_transform(cell_gene_counts_df)
principalDf_20 = pd.DataFrame(data = principalComponents)

principalDf_20.head()
```

Out[145]:

	0	1	2	3	4	5	6	7	
0	15.353967	22.551441	28.909571	18.160755	-63.669943	63.397297	22.120444	193.167733	5.(
1	-19.092789	-3.011189	37.073016	-7.781963	-0.324305	-5.521003	1.450280	-0.053563	-2.1
2	1.624026	-26.093832	-8.735882	1.431625	3.908805	-0.872093	-2.047048	2.420191	3.5
3	-15.469770	37.906453	-37.408305	5.952021	-10.229874	4.293272	15.286185	-4.262494	-6.7
4	-15.223271	-2.999145	38.531674	-6.379686	-6.113615	-4.637031	5.044940	-2.089725	-6.8

In [148]:
```
principalDf_20_numpy = principalDf_20.to_numpy()

pca_labels = kmeans(principalDf_20_numpy, 20, 15)
```

In [150]:
```
for i in np.unique(pca_labels):
    plt.scatter(principalDf_20_numpy[pca_labels == i , 0] , principalDf_20_

plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.title('kmeans on top 20 on top 20 pca components of gene expression dat
plt.legend()
plt.show()
```



Part 3: t-SNE

In this final section, we will visualize the data again using t-SNE - a non-linear dimensionality reduction algorithm. You can learn more about t-SNE in this interactive tutorial: https://distill.pub/2016/misread-tsne/ (https://distill.pub/2016/misread-tsne/).

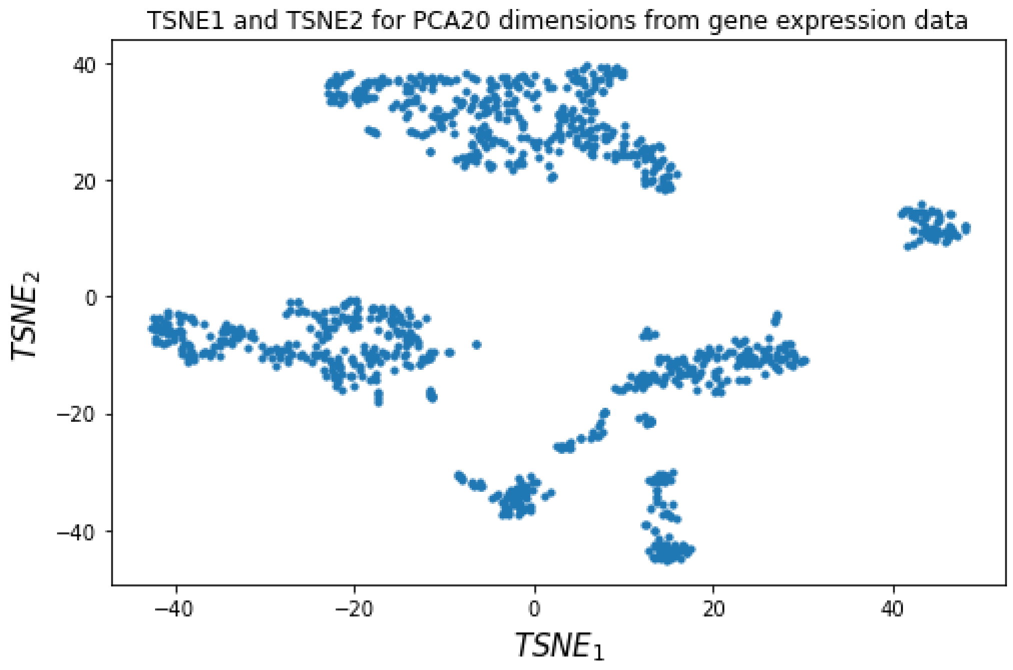
Use t-SNE to reduce the first 20 principal components of the gene expression dataset to two dimensions. You may use `sklearn.manifold.TSNE`. Note that it is recommended to first perform PCA before applying t-SNE to suppress noise and speed up computation.

In [113]:
```
tsne = TSNE()
tsne_pca_results = tsne.fit_transform(principalDf_20)
```

Plot the data (first 20 principal components) projected onto the first two t-SNE dimensions.

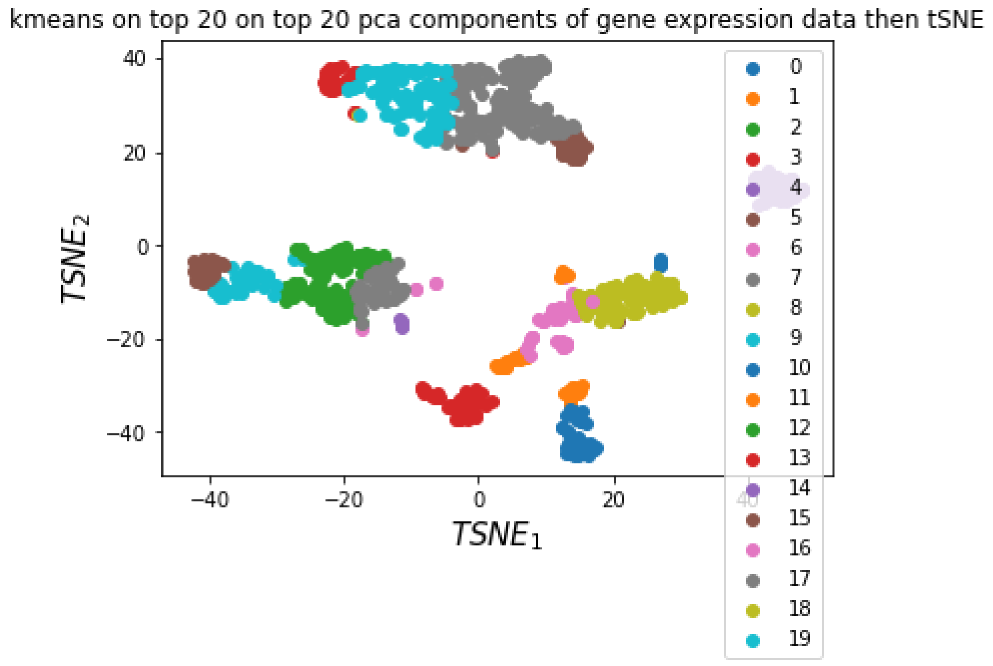
```
In [169]: plt.figure(figsize=(8, 5))
          plt.scatter(tsne_pca_results[:, 0], tsne_pca_results[:, 1], s=10)
          plt.xlabel('$TSNE_1$', fontsize=15)
          plt.ylabel('$TSNE_2$', fontsize=15)
          plt.title('TSNE1 and TSNE2 for PCA20 dimensions from gene expression data')
```

```
Out[169]: Text(0.5, 1.0, 'TSNE1 and TSNE2 for PCA20 dimensions from gene expression
          data')
```



Plot the data (first 20 principal components) projected onto the first two t-SNE dimensions, with points colored by their cluster assignments from part 2.

In [170]:
```
for i in np.unique(pca_labels):
    plt.scatter(tsne_pca_results[pca_labels == i , 0] , tsne_pca_results[pc
plt.xlabel('$TSNE_1$', fontsize=15)
plt.ylabel('$TSNE_2$', fontsize=15)
plt.title('kmeans on top 20 on top 20 pca components of gene expression dat
plt.legend()
plt.show()
```



Why is there overlap between points in different clusters in the t-SNE plot above?

In [19]:
```
### There is overlap because we are reducing an already reduced dimensional
### are unable to see the 'depth' or third/more dimensions which may be sep
### tSNE is a probabilisitic algorithm so it's possible that the overlap is
### which lends itself to non-clear cut /black and white slices.
```

These 20 clusters may correspond to various cell subtypes or cell states. They can be further investigated and mapped to known cell types based on their gene expressions (e.g. using the K-means cluster centers). The clusters may also be used in downstream analysis. For instance, we can monitor how the clusters evolve and interact with each other over time in response to a treatment.