**Written Report – 6.419x Module 2**

**Name:** 4sowmya

* **Problem 2: Larger unlabeled subset**

**Part 1: Visualization**

A scientist tells you that cells in the brain are either excitatory neurons, inhibitory neurons, or non-neuronal cells. Cells from each of these three groups serve different functions within the brain. Within each of these three types, there are numerous distinct sub-types that a cell can be, and sub-types of the same larger class can serve similar functions. Your goal is to produce visualizations which show how the scientist's knowledge reflects in the data.

As in Problem 1, we recommend using PCA before running T-SNE or clustering algorithms, for quality and computational reasons.

1. **(3 points) Provide at least one visualization which clearly shows the existence of three main brain cell types as described by the scientist and explain how it shows this. Your visualization should support the idea that cells from different groups can differ greatly.**

To visualize the three main brain cell, PCA and T-SNE along with K-means methods can be used. PCA (Principal Component Analysis) uses orthogonal transformations to convert correlates features into a set of values of linearly uncorrelated features. Then rank the dimensions according to the variance of the data along them. Because of the low interpretability of PCA, we will use PCA as preprocessing here and then use T-SNE for visualizing the main brain cell similarities. T-SNE uses local relationships between points to create low-dimension mapping which results in capturing non-linear relationship. It expands dense clusters and contracts sparse ones, evening out cluster size.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.manifold import MDS

from sklearn.manifold import TSNE

from sklearn.cluster import KMeans

X = np.load("data\p2\_unsupervised\X.npy")

# Due to the presence of high magnitude of data in only few cells in the genes, we will normalize the data.

Xlog = np.log2(X+1)

pca = PCA ()

pca\_comps = pca.fit\_transform(Xlog)

# Variance explained

variances = pca.explained\_variance\_ratio\_

comps = [i+1 for i in range (Xlog.shape[0])]

# Calculate the total variability and look for 85% threshold

cdf = np.cumsum(variances)

fig,ax=plt.subplots()

ax.plot(comps,cdf,label="cdf")

ax.axhline(y=0.85,color='r',linestyle='dashed')

ax.axvline(x=1250, color='r',linestyle='dashed')

plt.xlabel("Components")

plt.ylabel("Variance Explained")

plt.title("Variance Explained vs Number of Component  Data")

plt.show()

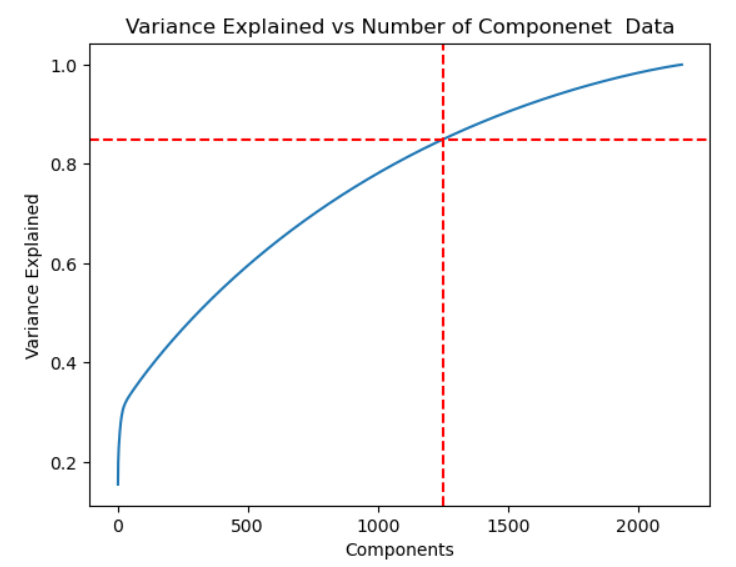


Figure 1 - 85% variance can be achieved with 1250 components

Based on the above diagram, we can infer that 85% variance can be explained by 1250 components. Hence, we can consider the 1250 components of PCA for further analysis.

# Consider 1250 component of PCA

transfrm\_pca = pca.transform(Xlog)[:,:1250]

plt.scatter(transfrm\_pca[:,0],transfrm\_pca[:,1])

plt.show()

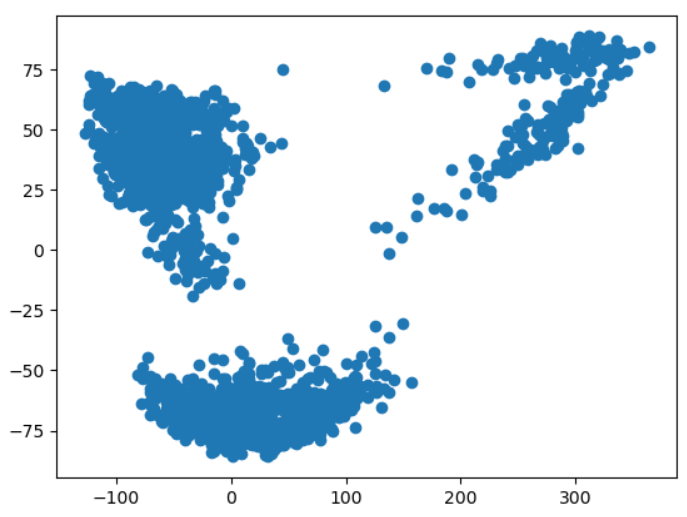


Figure 2 – PCA1 Vs PCA2 showing 3 distinct cluster of features

The above image shows 3 main clusters in the data which can be interpreted as 3 main cell types in the brain cells. But this is only visual analysis. Since we are attempting to find similar cells as meaningful cluster for the data where we do not have any prior labeling, we will compute silhoute\_score as metrics to measure the best number of clusters. (There is also another method using inertia value. I just picked this method for this analysis. There is no special reason behind it).

# use silhoute score to estimate the number of clusters needed

def get\_bestCluster(data):

    from sklearn.metrics import  silhouette\_score

    from sklearn.model\_selection import ParameterGrid

    # candidate values for our number of cluster

    parameters = [2, 3, 4, 5, 10, 15, 20, 25, 30, 35, 40]

    # initializing ParameterGrid, pass number of clusters as input

    parameter\_grid = ParameterGrid({'n\_clusters': parameters})

    best\_score = -1

    kmeans\_model = KMeans()     # instantiating KMeans model

    silhouette\_scores = []

    # evaluation based on silhouette\_score

    for p in parameter\_grid:

        kmeans\_model.set\_params(\*\*p) # set current hyper parameter

        kmeans\_model.fit(data) # fit model on wine dataset, this will find clusters based on parameter p

        ss = silhouette\_score(data, kmeans\_model.labels\_)   # calculate silhouette\_score

        silhouette\_scores += [ss]       # store all the scores

        # Check p which has the best score

        if ss > best\_score:

            best\_score = ss

            best\_grid = p

    return best\_grid

# TSNE with PCA initialization

tsne\_50 = TSNE (init='pca', perplexity=50)

tsne\_trns\_50 = tsne\_50.fit\_transform(transfrm\_pca)

n\_clusters = get\_bestCluster(tsne\_trns\_50)

kmeans = KMeans(n\_clusters= n\_clusters, random\_state=0).fit(tsne\_trns\_50[:,:1250])

plt.scatter(tsne\_trns\_50[:,0], tsne\_trns\_50[:,1], c=kmeans.labels\_)

plt.title("T-SNE with init=PCA, perplexity=50")

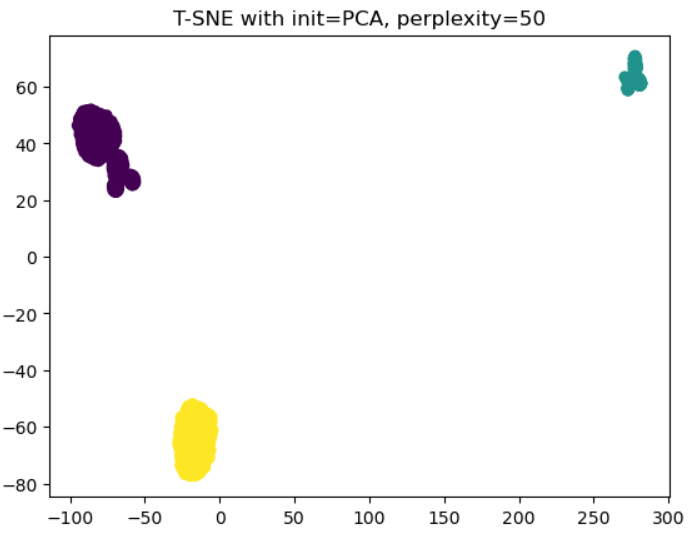


Figure 3 TSNE with perplexity=50 and cluster count obtained from silhoute score

The above visual shows that the best result is 3 which matches with our visual interpretation.

1. **(4 points) Provide at least one visualization which supports the claim that within each of the three types, there are numerous possible sub-types for a cell. In your visualization, highlight which of the three main types these sub-types belong to. Again, explain how your visualization supports the claim.**

To create a visual on the subtypes for each cell, I am using hierarchical clustering method in SciPy. First, I create a cluster visual using MDS. I used the log transformed X value.

embedding = MDS(n\_components=2)

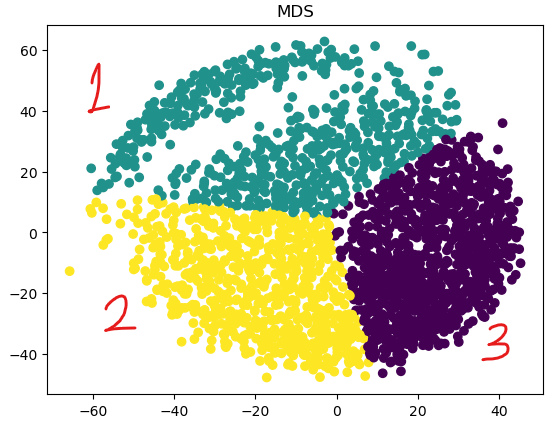
mds = embedding.fit\_transform(Xlog[:,:1250])

kmeans = KMeans(n\_clusters=3, random\_state=0, n\_init=10).fit(mds)

plt.scatter(mds[:,0],mds[:,1], c=kmeans.labels\_)

plt.title('MDS')

plt.show()



The image shows 3 distinct cluster which we can infer as 3 main cell type. Then I created dendrogram using the SciPy cluster library to create a hierarchical structure for cluster and its corresponding subgroups. A dendrogram is a tree like diagram that records the sequence of merges or splits calculated based on the distance between two clusters/ samples. The vertical line in the dendrogram represents the distance between the cluster. The larger the line means larger the distance between the cluster.

import scipy.cluster.hierarchy as shc

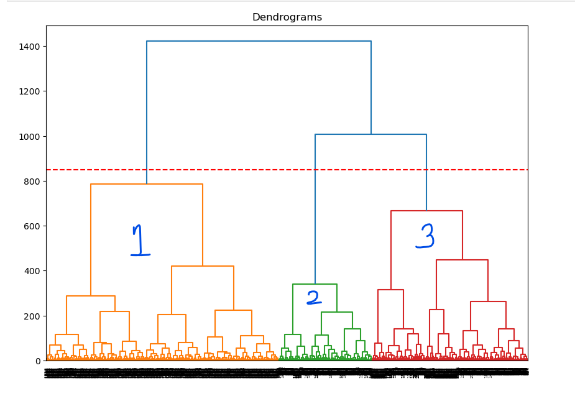
plt.figure(figsize=(10, 7))

plt.title("Dendrograms")

dend = shc.dendrogram(shc.linkage(mds, method='ward'))

plt.axhline(y=850,color='r',linestyle='dashed')

plt.show()

****

The x-axis in the dendrograms contains the samples and y-axis represents the distance between these samples. The vertical line with the maximum distance is the blue line and we can decide a threshold of 850 and cut the dendrogram. This provides 3 cluster of data that far apart in its features and each of them has multiple subsets of features that are closely connected within this main cluster.

**Part 2: Unsupervised Feature Selection**

Now we attempt to find informative genes which can help us differentiate between cells, using only unlabeled data. A genomics researcher would use specialized, domain-specific tools to select these genes. We will instead take a general approach using logistic regression in conjunction with clustering. Briefly speaking, we will use the p2\_unsupervised dataset to cluster the data. Treating those cluster labels as ground truth, we will fit a logistic regression model and use its coefficients to select features. Finally, to evaluate the quality of these features, we will fit another logistic regression model on the training set in p2\_evaluation and run it on the test set in the same folder.

1. (**4 points**) Using your clustering method(s) of choice, find a suitable clustering for the cells. Briefly explain how you chose the number of clusters by appropriate visualizations and/or numerical findings. (To cluster cells into the subcategories instead of categories)

First read the data

import numpy as np

import matplotlib.pyplot as plt

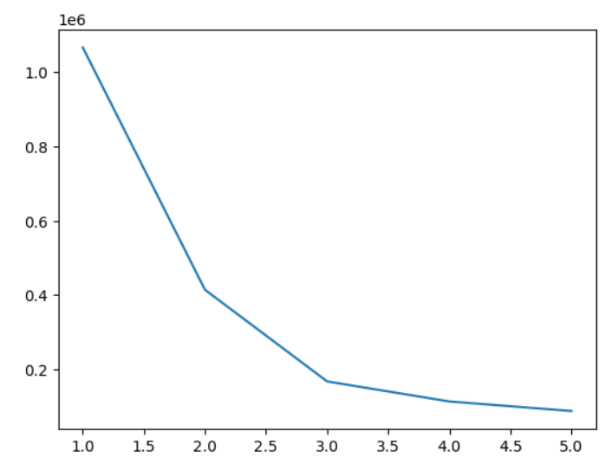
X = np.load('data/p2\_unsupervised/X.npy')

# normalize the training data

Xlog  = np.log2(X+1)

Xlog.max()

Let us use T-SNE transformation and WGSS for finding the number of possible clusters in the dataset



tsne = TSNE()

tsne\_trnf = tsne.fit\_transform(Xlog)

clusterCounter = [i for i in range(1,6)]

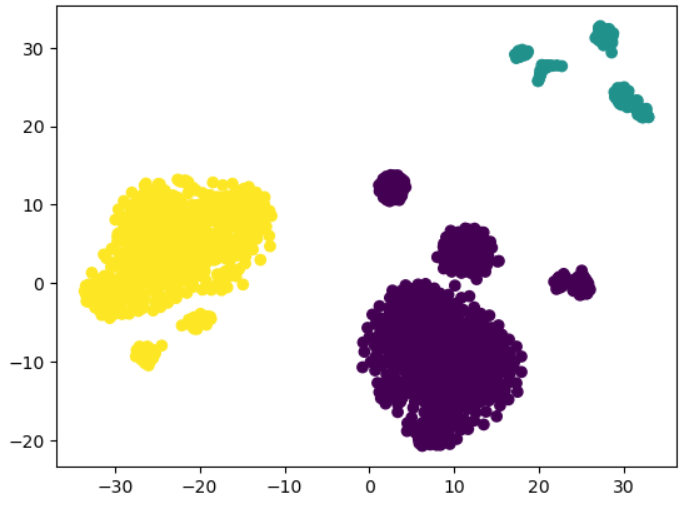
WGSS=[]

for \_c in clusterCounter:

    kmeans = KMeans(n\_clusters=\_c, random\_state=0).fit(tsne\_trnf)

    WGSS.append(kmeans.inertia\_)

plt.plot(clusterCounter, WGSS)



It looks like we have 3 cluster that gives best result. we will choose 3 for clustering. Now we will assign pseudo label to every data point.

kmeans = KMeans(n\_clusters=3, random\_state=0).fit(tsne\_trnf)

plt.scatter(tsne\_trnf[:,0], tsne\_trnf[:,1], c=kmeans.labels\_)

This is

1. (**6 points**) We will now treat your cluster assignments as labels for supervised learning. Fit a logistic regression model to the original data (not principal components), with your clustering as the target labels. Since the data is high-dimensional, make sure to regularize your model using your choice of l1, l2, or elastic net, and separate the data into training and validation or use cross-validation to select your model. Report your choice of regularization parameter and validation performance.

Continuing with the above code, we will use the output from the k-mean as labels and proceed to perform logistic regression.

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import train\_test\_split

penalties = ['l1','l2','elasticnet']

l1\_ratio = [0.1, 0.2,0.3,0.4,0.5,0.6,0.25,0.75,1]

Y = kmeans.predict(tsne\_trnf)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.33, random\_state=24)

Model\_details = []

classfier\_dict = {}

for i in penalties:

    if i == 'elasticnet':

        for r in l1\_ratio:

            clf = LogisticRegression(random\_state=0, penalty=i,l1\_ratio=r, multi\_class='ovr', solver='saga', max\_iter=200).fit(X\_train, y\_train)

            score = clf.score(X\_test,y\_test)

Model\_details.append({'score':score, 'ratio':r, 'model':clf,'penalty':i })

    else:

        clf = LogisticRegression(random\_state=0, penalty=i, multi\_class='ovr' , solver='liblinear', max\_iter=200).fit(X\_train, y\_train)

        score = clf.score(X\_test,y\_test)

Model\_details.append({'score':score, 'penalty':i, 'model':clf })

# Final classifier

lr\_clf = LogisticRegression(random\_state=10, penalty='l1', multi\_class='ovr', solver='liblinear').fit(X\_train, y\_train)

score = lr\_clf.score(X\_test,y\_test)

print(score)

After trying out multiple regularization parameters, all the three (l1,l2, elasticnet) and cross validated with multiple lr\_ratio, most parameters seems to provide the similar results. For the final model, I choose the l1 regularization parameter on a ‘one-vs-rest’ classification to get the linear regression prediction. Accuracy score is calculated used score() function. The result is 0.99

Here is one result for each regularization as example:

L1 result = {'score': 0.9986033519553073,

'penalty': 'l1',

'model': LogisticRegression(max\_iter=200, multi\_class='ovr', penalty='l1',

random\_state=0, solver='liblinear')}

l2 result = {'score': 0.9986033519553073,

'penalty': 'l2',

'model': LogisticRegression(max\_iter=200, multi\_class='ovr', random\_state=0,

solver='liblinear')}

elasticnet result = {'score': 0.9986033519553073,

'ratio': 0.25,

'model': LogisticRegression(l1\_ratio=0.25, max\_iter=200, multi\_class='ovr',

penalty='elasticnet', random\_state=0, solver='saga'),

'penalty': 'elasticnet'}

1. (**9 points**) Select the features with the top 100 corresponding coefficient values (since this is a multi-class model, you can rank the coefficients using the maximum absolute value over classes, or the sum of absolute values). Take the evaluation training data in p2\_evaluation and use a subset of the genes consisting of the features you selected. Train a logistic regression classifier on this training data, and evaluate its performance on the evaluation test data. Report your score. (Don't forget to take the log transform  before training and testing.)

Compare the obtained score with two baselines: random features (take a random selection of 100 genes), and high-variance features (take the 100 genes with highest variance). Finally, compare the variances of the features you selected with the highest variance features by plotting a histogram of the variances of features selected by both methods.

**Note:** The histogram should show the distribution of the variances of features selected by both methods. You could show the comparison by overlaying both histograms in the same plot.

import numpy as np

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.linear\_model import LogisticRegression

X\_train = np.load('data/p2\_evaluation\_reduced/X\_train.npy')

y\_train = np.load('data/p2\_evaluation\_reduced/y\_train.npy')

X\_test = np.load('data/p2\_evaluation\_reduced/X\_test.npy')

y\_test = np.load('data/p2\_evaluation\_reduced/y\_test.npy')

X\_train = np.log2(X\_train+1)

X\_test = np.log2(X\_test+1)

print(X\_train.shape, y\_train.shape, X\_test.shape, y\_test.shape)

# take 100 random  features

indexes = [i for i in range(X\_train.shape[1])]

rdm\_feats = np.random.choice(indexes, size=100)

rnd\_X\_train = X\_train[:,rdm\_feats]

rnd\_X\_test = X\_test[:,rdm\_feats]

top\_100\_clf = LogisticRegression(random\_state=10, penalty='l2', multi\_class='ovr', solver='liblinear').fit(rnd\_X\_train, y\_train)

accuracy = top\_100\_clf.score(rnd\_X\_test,y\_test)

print("Score for top 100 features = {}".format(accuracy))

output:

Score for top 100 features = 0.5388086642599278

Now, lets compare the above result with a model created from PCA based 100 features.

pca = PCA()

pca.fit(X\_train)

# Variance Explained by 1st Component

pca\_X\_train = pca.transform(X\_train)[:,:100]

pca\_X\_test= pca.transform(X\_test) [:,:100]

print("pca transformed Train Data shape {}".format(pca\_X\_train.shape))

print("pca transformed Test Data shape {}".format(pca\_X\_test.shape))

pca\_clf = LogisticRegression(random\_state=10, penalty='l1', multi\_class='ovr', solver='liblinear').fit(pca\_X\_train, y\_train)

accuracy = pca\_clf.score(pca\_X\_test,y\_test)

print("accuracy Score for top 100 high variance features: {}".format(accuracy))

output:

pca transformed Train Data shape (1077, 100)

pca transformed Test Data shape (1108, 100)

Validation Score for top 100 high variance features: 0.9259927797833934

Now we will create a histogram and compare the variance based on the trainingdata, top features selected, top 100 features from PCA.

import seaborn as sns

histog = np.var(X\_train[:,:100], axis=0)

histogramRandom = np.var(rnd\_X\_train[:,:100], axis=0)

histogram\_high\_variance = np.var(pca\_X\_train[:,:100], axis=0)

sns.displot(histog, kind='hist')

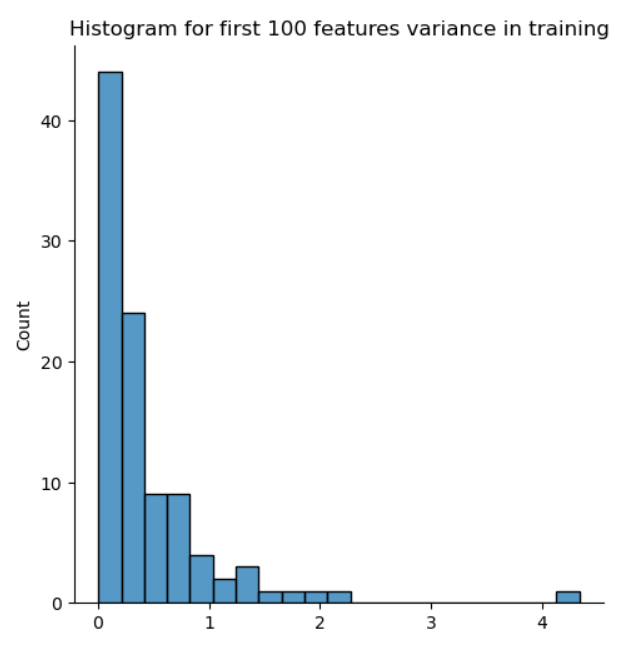
print("Histogram for first 100 features variance in training")

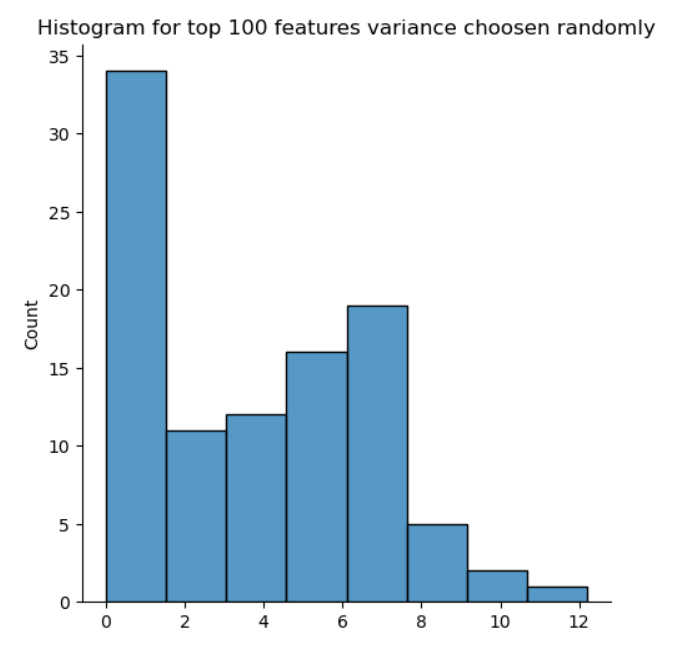
sns.displot(histogramRandom, kind='hist')

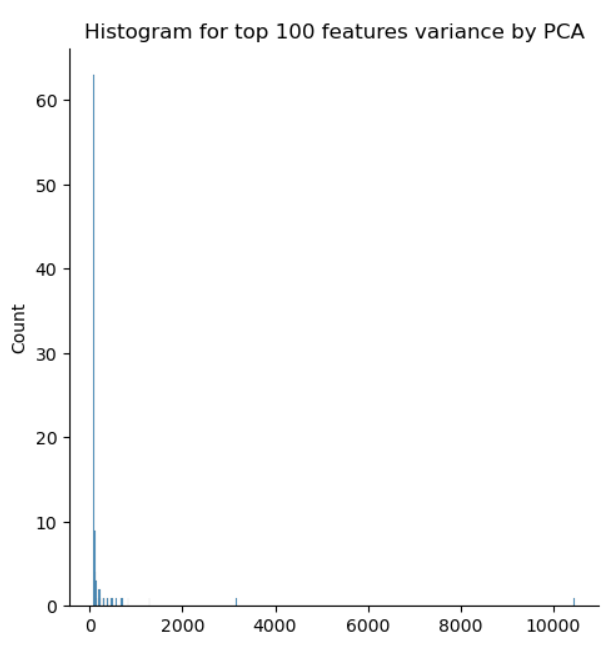
print("Histogram for top 100 features variance choosen randomly")

sns.displot(histogram\_high\_variance, kind='hist')

print("Histogram for top 100 features variance by PCA")







**Include your answers to all parts of problem 3 in your written report. This problem is worth 16 points.**

The hyper-parameter choices used in data analysis techniques can have a large impact on the inferences made. As you may have encountered, finding the best choice of parameter such as perplexity in T-SNE or the number of clusters can be an ambiguous problem. We will now investigate the sensitivity of your results to changes in these hyper-parameters, with the goal of understanding how your conclusions may vary depending on these choices.

1. (**3 points**) When we created the T-SNE plot in Problem 1, we ran T-SNE on the top 50 PC's of the data. But we could have easily chosen a different number of PC's to represent the data. Run T-SNE using 10, 50, 100, 250, and 500 PC's, and plot the resulting visualization for each. What do you observe as you increase the number of PC's used?

import time

Xpath = 'data/p1/X.npy'

yPath = 'data/p1/y.npy'

X = np.load(Xpath)

y = np.load(yPath)

X1 = np.log2(X+1)

PCS = [10,20,50,100,250,500]

Transformed\_top\_pca = TSNE()

(fig, subplots) = plt.subplots(3, 2, figsize=(15, 8))

subplots = subplots.flatten()

for idx, \_c in enumerate(PCS):

    pca = PCA(n\_components=\_c)

    pca\_trs = pca.fit\_transform(X1)

    ax = subplots[idx]

    t0 = time.time()

    tsne\_top\_pca = Transformed\_top\_pca.fit\_transform(pca\_trs[:,:\_c])

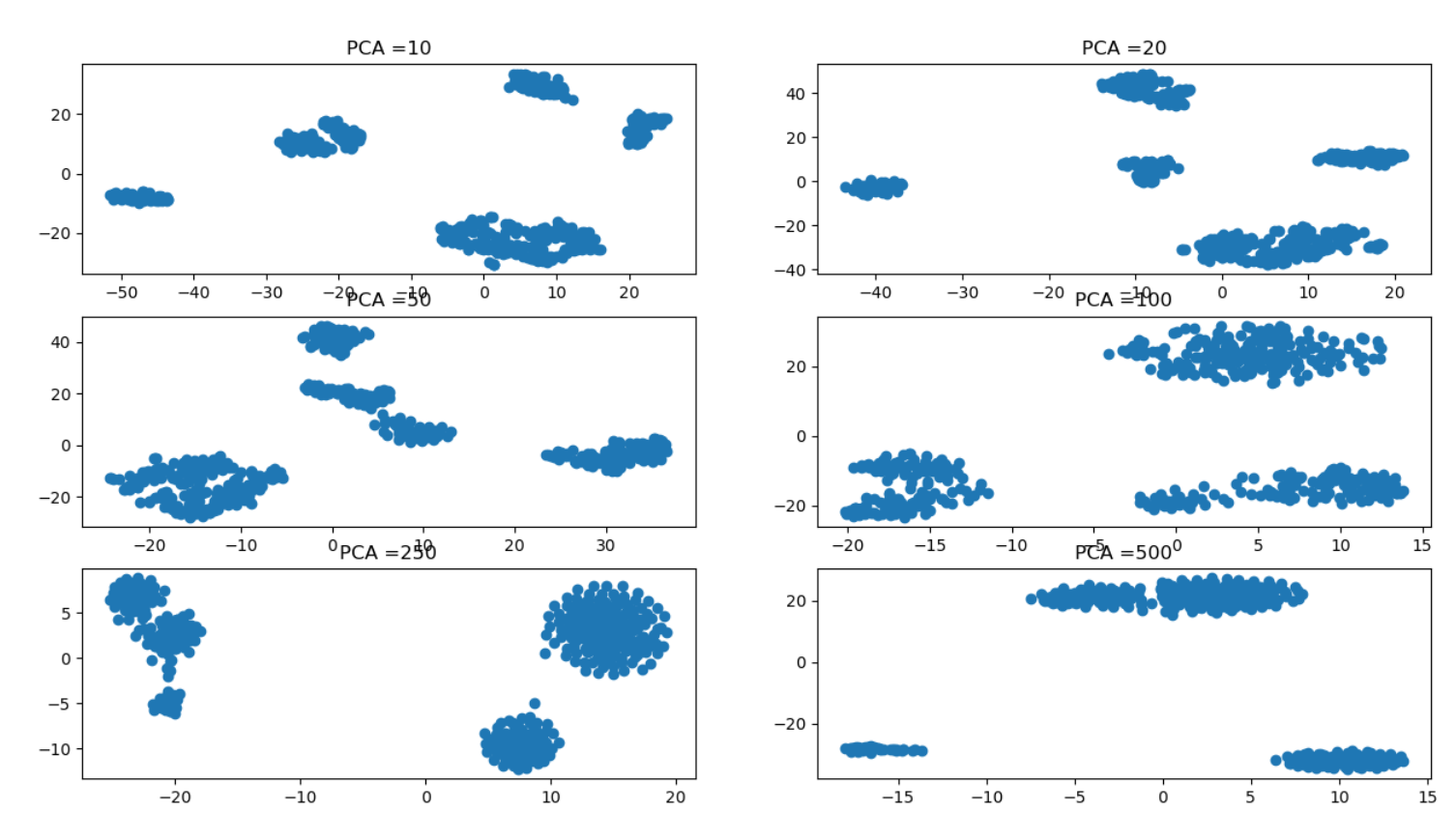
    t1 = time.time()

    ax.set\_title("PCA =%d" % \_c)

    ax.scatter(tsne\_top\_pca[:,0], tsne\_top\_pca[:,1])

    ax.axis('tight')

plt.show()



1. (**13 points**) Pick three hyper-parameters below (the 3 is the total number that a report needs to analyze. It can take a) 2 from A, 1 from B, or b) 1 from A, 2 from B.) and analyze how changing the hyper-parameters affect the conclusions that can be drawn from the data. Please choose at least one hyper-parameter from each of the two categories (visualization and clustering/feature selection). At minimum, evaluate the hyper-parameters individually, but you may also evaluate how joint changes in the hyper-parameters affect the results. You may use any of the datasets we have given you in this project. For visualization hyper-parameters, you may find it productive to augment your analysis with experiments on synthetic data, though we request that you use real data in at least one demonstration.

Some possible choices of hyper-parameters are:

**Category A (visualization):**

* + T-SNE perplexity
  + T-SNE learning rate
  + T-SNE early exaggeration
  + T-SNE initialization
  + T-SNE number of iterations/convergence tolerance

**Category B (clustering/feature selection):**

* + Effect of number of PC's chosen on clustering
  + Type of clustering criterion used in hierarchical clustering (single linkage vs ward, for example)
  + Number of clusters chosen for use in unsupervised feature selection and how it affects the quality of the chosen features
  + Magnitude of regularization and its relation to your feature selection (for example, does under or over-regularizing the model lead to bad features being selected?)
  + Type of regularization (, , elastic net) in the logistic regression step and how the resulting features selected differ

For visualization hyper-parameters, provide substantial visualizations and explanation on how the parameter affects the image.

For clustering/feature selection, provide visualizations and/or numerical results which demonstrate how different choices affect the downstream visualizations and feature selection quality.

Provide adequate explanations in words for each of these visualizations and numerical results.

From Category-A:, I picked Perplexity, Learning rate and from category -B we will check k-means clusters with number of PCS

T-SNE Perplexity:

pca = PCA(n\_components=50)

trn\_pca = pca.fit\_transform(X1)

perplexities = [2,5,10,30, 50, 100, 200,500]

(fig, subplots) = plt.subplots(4, 2, figsize=(15, 8))

subplots = subplots.flatten()

for idx, \_c in enumerate(perplexities):

    top50\_pca = TSNE(perplexity=\_c)

    ax = subplots[idx]

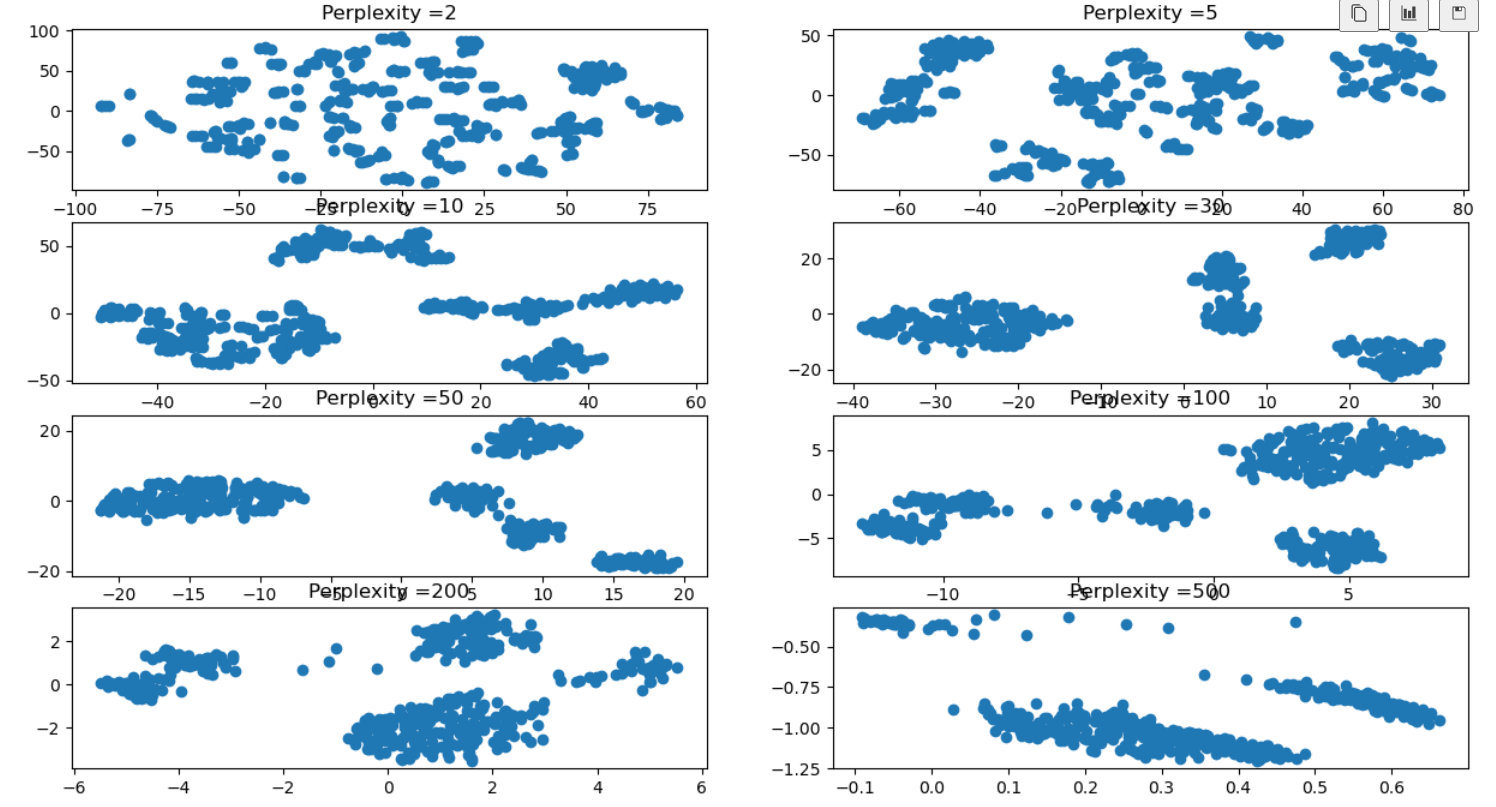
    tsnepcatoppcs = top50\_pca.fit\_transform(trn\_pca)

    ax.set\_title("Perplexity =%d" % \_c)

    ax.scatter(tsnepcatoppcs[:,0], tsnepcatoppcs[:,1])

    ax.axis('tight')

plt.show()



Perplexity tells about how to balance between global and local structure of data. The clustering of data changes significantly as the perplexity changes. From the image above, we can see that perplexity of 30,50 provides better grouping of similar data than other values.

Learning rate:

trn\_pca = pca.fit\_transform(X1)

learning\_rate = ['auto',10,25,50,100,1000]

(fig, subplots) = plt.subplots(3, 2, figsize=(15, 8))

subplots = subplots.flatten()

for idx, \_c in enumerate(learning\_rate):

    top50\_pca = TSNE(learning\_rate=\_c, perplexity=30)

    ax = subplots[idx]

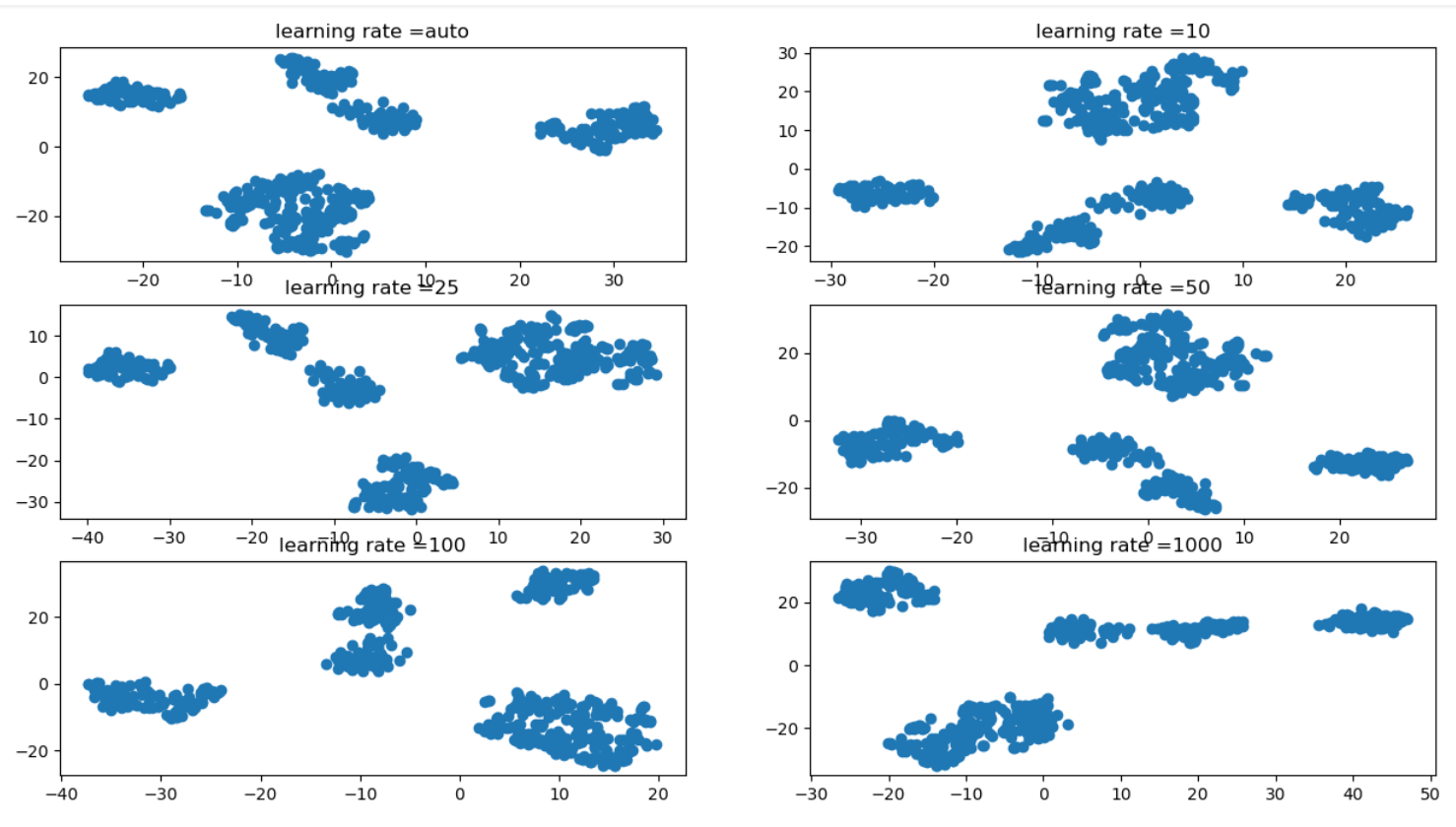
    tsne\_top\_pcs = top50\_pca.fit\_transform(trn\_pca)

    ax.set\_title("learning rate ={}".format(\_c))

    ax.scatter(tsne\_top\_pcs[:,0], tsne\_top\_pcs[:,1])

    ax.axis('tight')

plt.show()



The learning rate for t-SNE is usually in the range [10.0, 1000.0]. If the learning rate is too high, the data may look like a ‘ball’ with any point approximately equidistant from its nearest neighbours. If the learning rate is too low, most points may look compressed in a dense cloud with few outliers.

**K-mean cluster:**

PCS = [10,20,50,100,250,350,500]

(fig, subplots) = plt.subplots(4, 2, figsize=(15, 15))

subplots = subplots.flatten()

for idx, \_c in enumerate(PCS):

    pca = PCA(n\_components=\_c)

    trsn\_pca = pca.fit\_transform(X1)

    ax = subplots[idx]

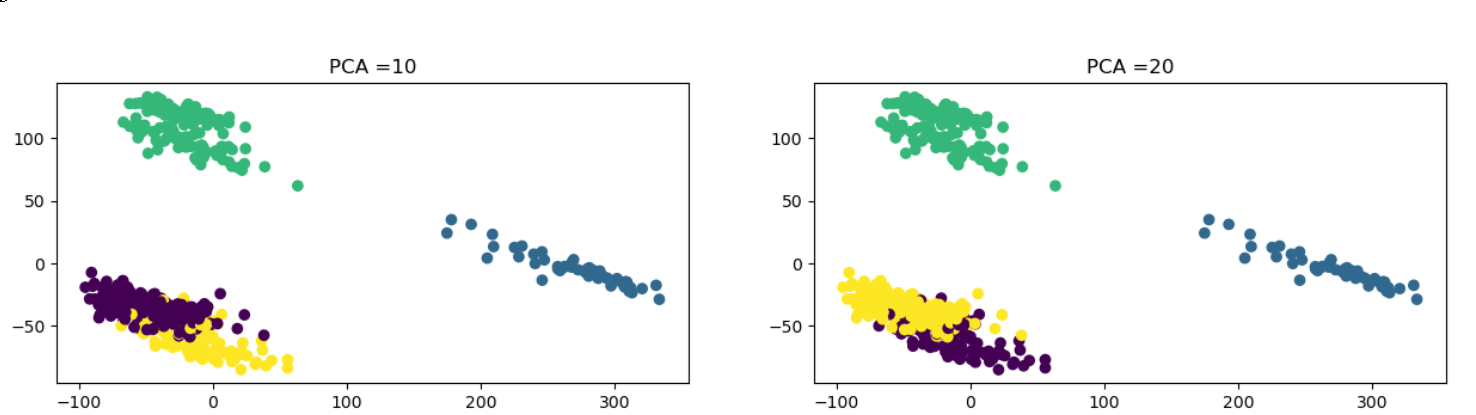
    kmeans = KMeans(n\_clusters=4, random\_state=0).fit(trsn\_pca)

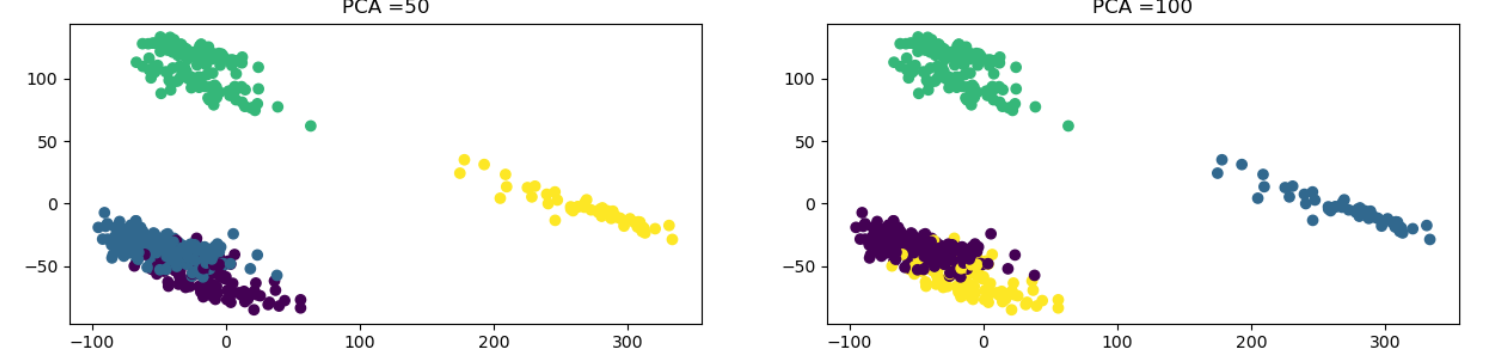
    ax.set\_title("PCA =%d" % \_c)

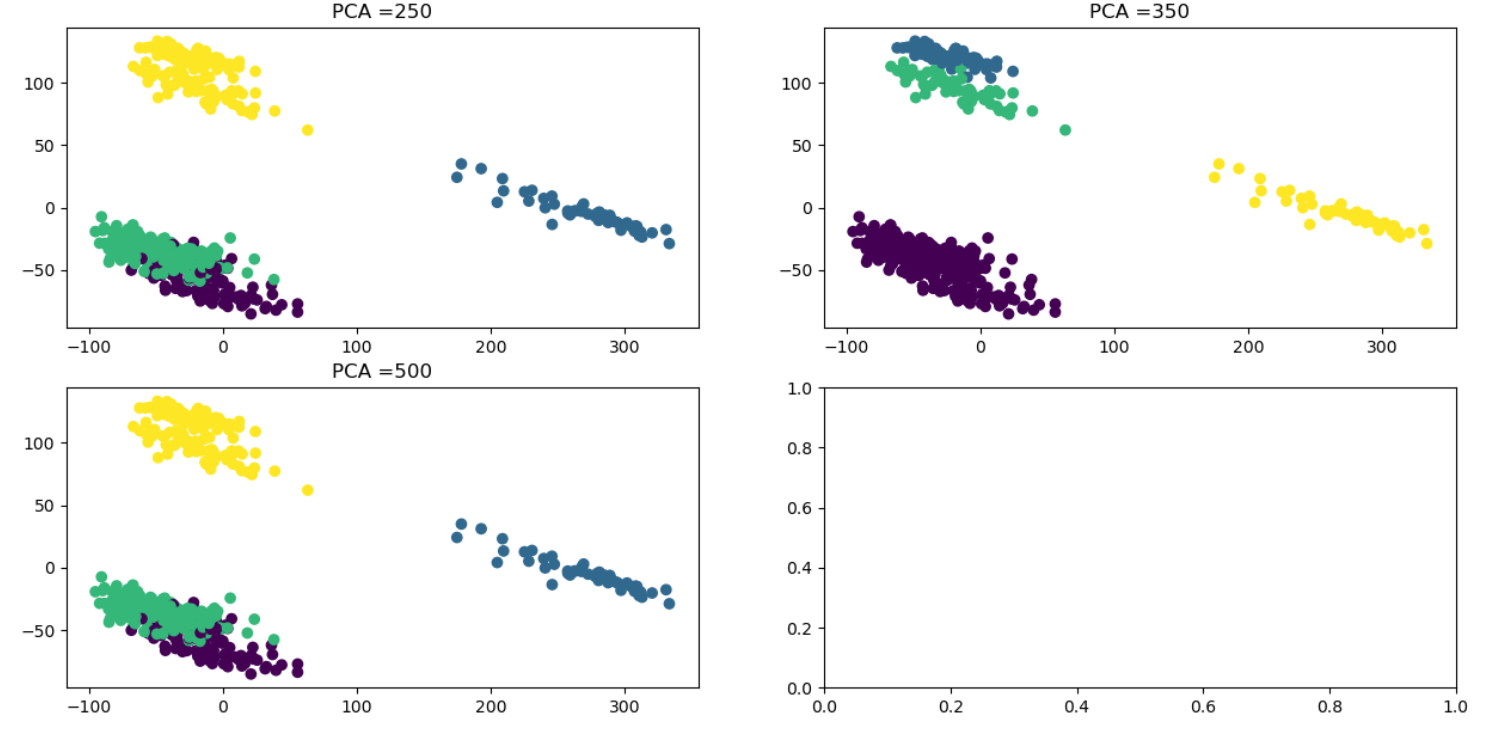
    ax.scatter(trsn\_pca[:,0], trsn\_pca[:,1],  c=kmeans.labels\_)

    ax.axis('tight')

plt.show()







K-means try to minimize the distance within a cluster for a given dimension. Hence the compressibility of PCA helps a lot.