K-Means Implementation (preparation for data processing):
Class in K – Means:
__init__: using for set up the data set for k-means
Init_random_centroids: randomly pick the centroid for clustering
_closest_centroid: finding the centroid that is close to the node
Create_clusters: building up the clusters.
Update_centroids: updating the cluster centroid to the mean of all the points assigned to that cluster
Get_cluster_labels: show the label of clusters
Predict: predict the next label of the k – means' clusters

Discussion:

1. By doing what description says, we try to build up the loop to check the change for each k-means to see the difference:

```
#discuss1
num = 20
for x in range(num):
    kmeans(datalist, k =5)pca=PCA(n_components=3)
    pca.fit(data_list)
    X_train_reduction=pca.transform(data_list)
    print(kmeans(datalist, k =5))
```

Based on the function, based on SEE, although the given data is word not numbers, by translate into math weight, we can make sure the output of the clusters is much accurate to the graph that shows in description.

2. Seeing that the given data is mostly string value, it cannot be easily count as numbers. Thus, I decide to use the label categories, math weight similarity, and the distance to determine the best clusters. However, I think the math weight is the best way to make this dataset. As what I talk in part 1, it is easier to calculate SSE by value. By doing the PCA, we can actually see the difference by repeating doing the K – meas kluster.

3. None ans
4.

```
#discuss4
#5
for x in range(num):
    kmeans(datalist, k =5)pca=PCA(n_components=3)
    pca.fit(data_list)
    X_train_reduction=pca.transform(data_list)
    print(kmeans(datalist, k =5))
#10
for x in range(num):
    kmeans(datalist, k =5)pca=PCA(n_components=3)
    pca.fit(data_list)
    X_train_reduction=pca.transform(data_list)
    print(kmeans(datalist, k =10))
#15
for x in range(num):
    kmeans(datalist, k =5)pca=PCA(n_components=3)
    pca.fit(data_list)
    X_train_reduction=pca.transform(data_list)
    print(kmeans(datalist, k =15))
#20
for x in range(num):
    kmeans(datalist, k =5)pca=PCA(n_components=3)
    pca.fit(data_list)
    X_train_reduction=pca.transform(data_list)
    print(kmeans(datalist, k =20))
#25
for x in range(num):
    kmeans(datalist, k =5)pca=PCA(n_components=3)
    pca.fit(data_list)
    X_train_reduction=pca.transform(data_list)
    print(kmeans(datalist, k =25))
```

We repeating doing the part1, but with different groups of k-means kluster(5,10,15,20,25). By checking each kluster, although I do not do the discussion 3, I would say the best choice for k should be k = 10, which can specific clearly see the PCA graph.