Northeastern University, Khoury College of Computer Science

CS 6220 Data Mining — Assignment 4

Due: March 1, 2023(100 points)

Haoran Zhang
https://github.com/hrcheung
zhang.haoran1@northeastern.edu

K-Means

The normalized automobile distributor timing speed and ignition coil gaps for production F-150 trucks over the years of 1996, 1999, 2006, 2015, and 2022. We have stripped out the labels for the five years of data.

Each sample in the dataset is two-dimensional, i.e. $\mathbf{x}_i \in \mathbb{R}^2$ (one dimension for timing speed and the other for coil gaps), and there are $N = 5000$ instances in the data.

Question 1 [20 pts total]

[10 pts] Question 1a.) Implement a simple k-means algorithm in Python on Colab with the following initialization:

$$\mathbf{x}_1 = \begin{pmatrix} 10 \\ 10 \end{pmatrix}, \mathbf{x}_2 = \begin{pmatrix} -10 \\ -10 \end{pmatrix}, \mathbf{x}_3 = \begin{pmatrix} 2 \\ 2 \end{pmatrix}, \mathbf{x}_4 = \begin{pmatrix} 3 \\ 3 \end{pmatrix}, \mathbf{x}_5 = \begin{pmatrix} -3 \\ -3 \end{pmatrix},$$

You need only 100 iterations, maximum, and your algorithm should run very quickly to get the results.

```
def euclideanDistance(p1,p2):
    return math.sqrt(((p1[0]-p2[0])**2)+((p1[1]-p2[1])**2) )

def find_closest_centroid(point,centroids):
#     for i in range(len(centroids)):
```

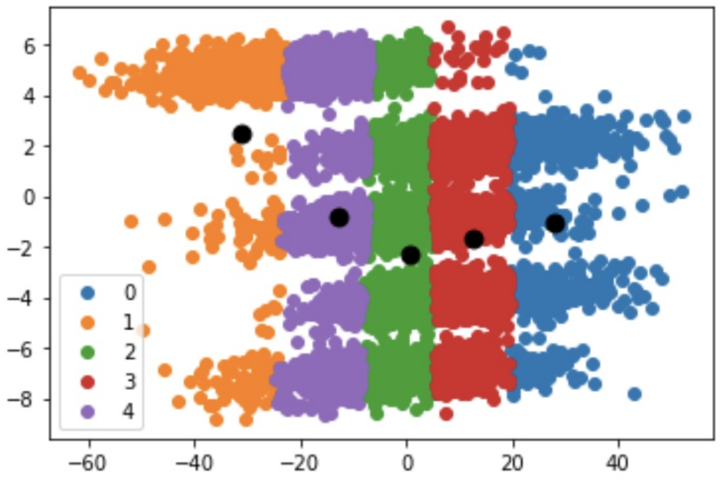
```
#         clusters[i]=list()
    minV=float("inf")
    label=-1
    for i, center in enumerate(centroids):
        if euclideanDistance(center,point)<minV:
            label=i
            minV=euclideanDistance(center,point)
    return label

#compute new centroid after assign
def computeNewCentroid(X):
    new=list()
    tmp=X.groupby(by='label').mean()
    for row in range(len(tmp)):
        new.append([tmp.at[row,'x'],tmp.at[row,'y']])
    return new


#check if centroids change
def centroidsChange(old,new):
    if euclideanDistance(old,new)>0.000000000001:
        return True
    else:
        return False

def fit(X,initialization,max_iter,n_cluster):
    centroids=initialization
#     print("initial centroids =",centroids)

    X['label']=X.apply(lambda row: find_closest_centroid((row[0],row[1]),centroi
    for i in range(max_iter):
        new_centroids=computeNewCentroid(X)
#         print("new centroids is",new_centroids)
        for i in range(len(centroids)):
            if not centroidsChange(centroids[i],new_centroids[i]):
                break
            else:
                centroids=new_centroids
                X['label']=X.apply(lambda row: find_closest_centroid((row[0],row
    return centroids

initialization=[[10,10],[-10,-10],[2,2],[3,3],[-3,-3]]
#         new_centroids=findNewCentroids(centroids)
clusters=fit(df,initialization,100,5)
clusters=np.asarray(clusters)
clusters
```

```
Out[22]: array([[ 28.09146831,  -1.0241717 ],
                [-30.93833893,   2.43436785],
                [  0.52511293,  -2.3158082 ],
                [ 12.86956124,  -1.68099097],
                [-12.88359021,  -0.80312447]])
```

[5 pts] **Question 1b.)** Scatter the results in two dimensions with different clusters as different colors. You can use **matplotlib**'s **pyplot** functionality:

```
>> import matplotlib.pyplot as plt
>> plt.scatter(<YOUR CODE HERE>)
```

```
u_labels=np.unique(df['label']) #allocat label to different clusters
for i in u_labels:
    plt.scatter(data[label==i,0],data[label==i,1],label=i)
plt.scatter(clusters[:,0] , clusters[:,1] , s = 80, color = 'k')
plt.legend()
plt.show()
```

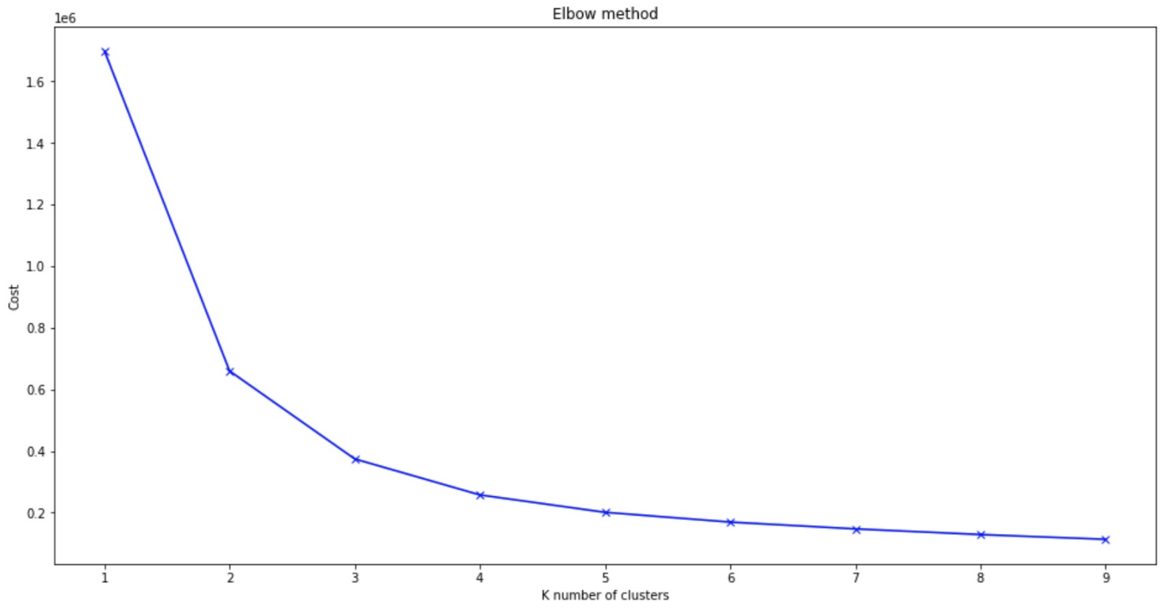


[5 pts] **Question 1c.)** You will notice that in the above, there are only five initialization clusters. Why is $k = 5$ a logical choice for this dataset? After plotting your resulting clusters, what do you notice? Did it cluster very well? Is there an initialization that would make it cluster well?

Based on the cluster figure above, the cluster result does not look good.
I am suspecting the initialization vectors and centroids are incorrect. They are all on the line of y=x, which disturbed the accurate clustering process.
Based on the picture, a better intialization would be y=c, where c in the constant, since apparently points can be clustered as blocks layered parallel to the x axis.

k=3 is a relatively good choice from the perspective of cost, according to Elbow method.

```
cost = []
K = range(1,10)
for k in K:
    kmeanModel = cluster.KMeans(n_clusters=k)
    kmeanModel.fit(data)
    cost.append(kmeanModel.inertia_) #sum of distances to cluster center

plt.figure(figsize=(16,8))
plt.plot(K, cost, 'bx-')
plt.xlabel('K number of clusters')
plt.ylabel('Cost')
plt.title('Elbow method')
plt.show()
```



Question 2)[30 pts total]

In the data from Question 1, let $\mathbf{x}$ and $\mathbf{y}$ be two instances, i.e., they are each truck with separate measurements. A common distance metric is the *Mahalanobis Distance* with a specialized matrix $P \in \mathbb{R}^{2\times 2}$ that is written as follows:

$$d(\mathbf{x}, \mathbf{y}) = (\mathbf{x} - \mathbf{y})^T P^{-1} (\mathbf{x} - \mathbf{y})$$

In scalar format (non-matrix format), the Mahalanobis Distance can be expressed as:

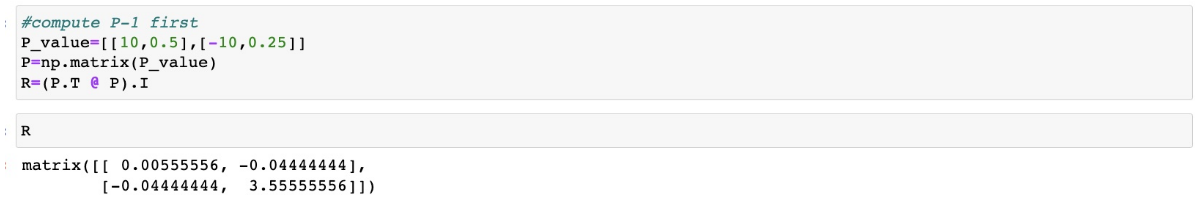
$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{2} \sum_{j=1}^{2} (x_i - y_i) \cdot P_{i,j}^{-1} \cdot (x_j - y_j)$$

where $\mathbf{x}$ and $\mathbf{y}$ are two instances of dimensionality 2, and $d(\mathbf{x}, \mathbf{y})$ is the distance between them. In the case of the F150 engine components, P is a known relationship through Ford's quality control analysis each year, where it is numerically shown as below:

$$P = \begin{pmatrix} 10 & 0.5 \\ -10 & 0.25 \end{pmatrix}$$

[15 pts] Question 2a.) Using the same data as **Question 1** and the same initialization instances $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5\}$ implement a specialized k-means with the above Mahalanobis Distance. Scatter the results with the different clusters as different colors.

What do you notice? You may want to pre-compute P^{-1} so that you aren't calculating an inverse every single loop of the the k-Means algorithm.



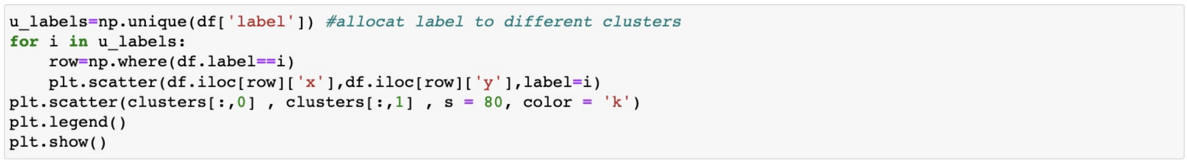
```
#compute P-1 first
P_value=[[10,0.5],[-10,0.25]]
P=np.matrix(P_value)
R=(P.T @ P).I
```

```
R
```

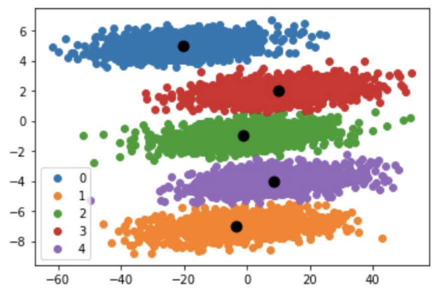
```
matrix([[ 0.00555556, -0.04444444],
        [-0.04444444,  3.55555556]])
```

```
def mahalaDistance(x,y):
    x=np.asarray(x)
    y=np.asarray(y)
    return (x-y).T @ R @ (x-y)

def find_closest_centroid(point,centroids):
#     for i in range(len(centroids)):
#         clusters[i]=list()
    minV=float("inf")
    label=-1
    for i, center in enumerate(centroids):
        if mahalaDistance(center,point)<minV:
            label=i
            minV=mahalaDistance(center,point)
    return label

#compute new centroid after assign
def computeNewCentroid(X):
    new=list()
    tmp=X.groupby(by='label').mean()
    for row in range(len(tmp)):
        new.append([tmp.at[row,'x'],tmp.at[row,'y']])
    return new


#check if centroids change
def centroidsChange(old,new):
    if mahalaDistance(old,new)>0.000000000001:
        return True
    else:
        return False
```

```
def fit(X,initialization,max_iter,n_cluster):
    centroids=initialization
#     print("initial centroids =",centroids)

    X['label']=X.apply(lambda row: find_closest_centroid((row[0],row[1]),centroi
    for i in range(max_iter):
        new_centroids=computeNewCentroid(X)
#         print("new centroids is",new_centroids)
        for i in range(len(centroids)):
            if not centroidsChange(centroids[i],new_centroids[i]):
                break
            else:
                centroids=new_centroids
                X['label']=X.apply(lambda row: find_closest_centroid((row[0],row
    return centroids
```



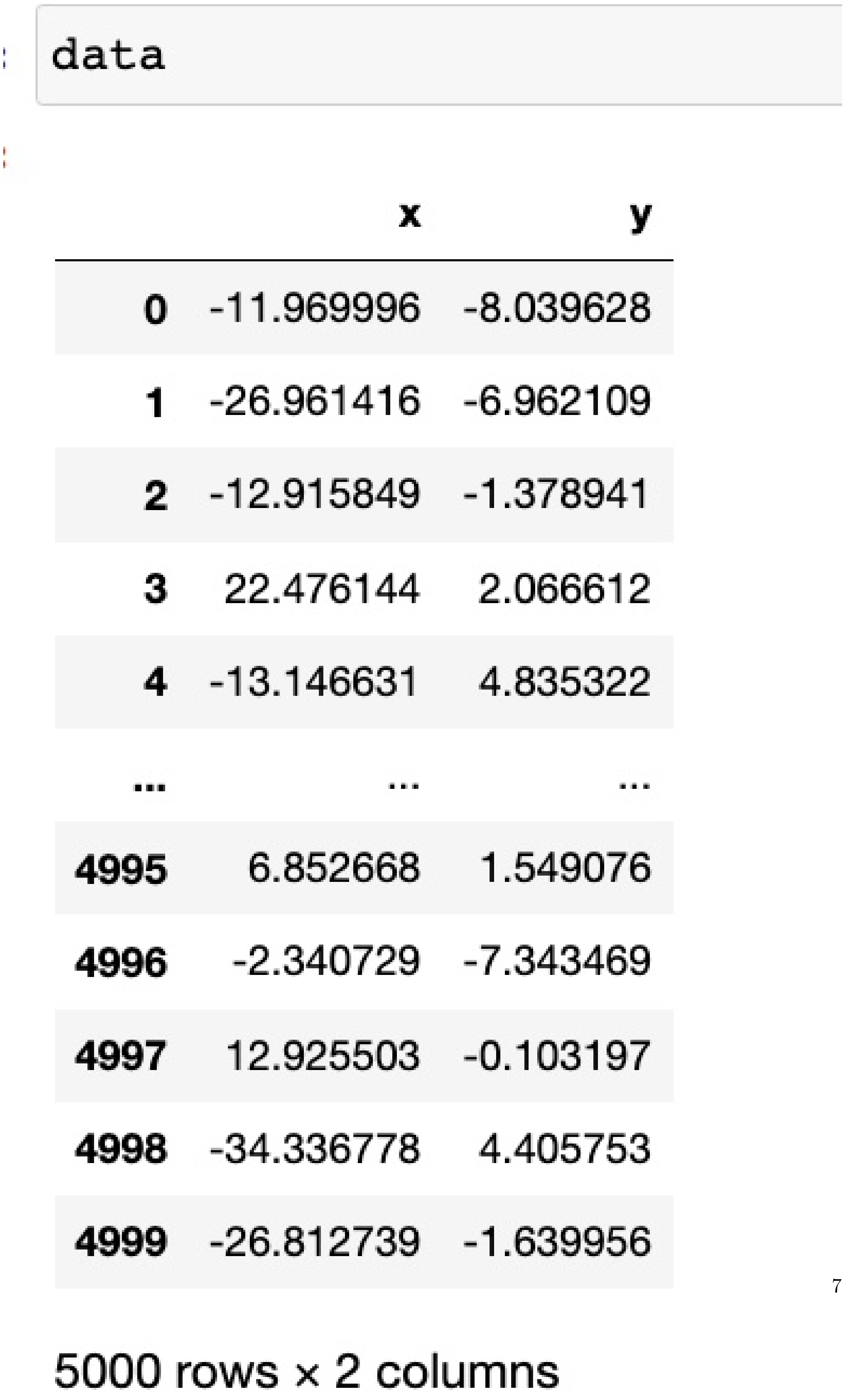
```
u_labels=np.unique(df['label']) #allocat label to different clusters
for i in u_labels:
    row=np.where(df.label==i)
    plt.scatter(df.iloc[row]['x'],df.iloc[row]['y'],label=i)
plt.scatter(clusters[:,0] , clusters[:,1] , s = 80, color = 'k')
plt.legend()
plt.show()
```



Points are clustered into a right way. Using Mahalanobis Distance allows a different way to calculate similarity between points.

[5 pts] Question 2b.) Calculate and print out the principle components of the aggregate data.

As requested by professor, if we don't do z-score scalar for PCA



```
data
```

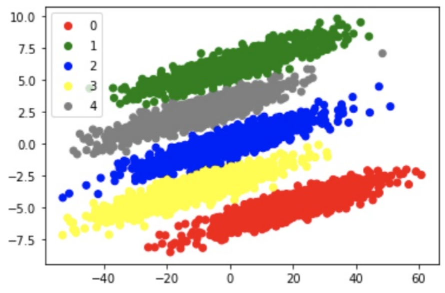
	x	y
0	-11.969996	-8.039628
1	-26.961416	-6.962109
2	-12.915849	-1.378941
3	22.476144	2.066612
4	-13.146631	4.835322
...	...	...
4995	6.852668	1.549076
4996	-2.340729	-7.343469
4997	12.925503	-0.103197
4998	-34.336778	4.405753
4999	-26.812739	-1.639956

5000 rows × 2 columns

```python
from sklearn.decomposition import PCA
# Two components of PCA
pca = PCA(2)

# Fit on data
pca.fit(data)

# Access values and vectors
print(pca.components_)
print(pca.explained_variance_)

# transform data
B = pca.transform(data)
print(B)
```

```
[[-0.99838317  0.05684225]
 [-0.05684225 -0.99838317]]
[322.50713273  17.38845582]
[[ 10.33254906   7.6356903 ]
 [ 25.36097916   7.41205973]
 [ 11.65548133   1.03953719]
 ...
 [-14.07157308  -1.70302537]
 [ 33.3705919   -3.51819072]
 [ 25.51506544   2.09006015]]
```

```python
import matplotlib.pyplot as plt
import matplotlib
colors=['red','green','blue','yellow','grey']
scatter=plt.scatter(B[:, 0], B[:, 1],c=target,\
                    cmap=matplotlib.colors.ListedColormap(colors),marker='o')
plt.legend(*scatter.legend_elements())
plt.show()
```



[5 pts] Question 2c.) Calculate and print out the principle components of *each cluster*. Are they the same as the aggregate data? Are they the same as each other?

Firstly I PCAed every cluster to 2-D. And then I drew the graph one by one onto the same figure.

Comparing 2b and 2c, the PCA results are not the same

The reason might be that if i PCA the aggregated data, data points' relative position is taken into consideration. However, when I do it seperately, only absolute position matters, and points are PCAed onto the same scalar.

```
: cluster_0=data_q2[data_q2['label'] == 0]
  cluster_1=data_q2[data_q2['label'] == 1]
  cluster_2=data_q2[data_q2['label'] == 2]
  cluster_3=data_q2[data_q2['label'] == 3]
  cluster_4=data_q2[data_q2['label'] == 4]
```

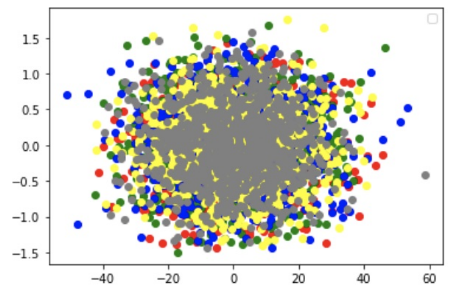
```
: cluster_0
```

:

	x	y	label
4	-13.146631	4.835322	0
12	-38.508282	4.112202	0
19	6.843345	4.550918	0
22	-36.768240	4.173526	0
25	-16.883207	4.664295	0
...	...	...	...
4985	-41.441343	5.164262	0
4987	-39.028493	5.153460	0
4991	-14.425117	5.286585	0
4994	-11.915634	5.320493	0
4998	-34.336778	4.405753	0

1000 rows × 3 columns

```
colors=['red','green','blue','yellow','grey']

for i in range(len(res)):
    scatter=plt.scatter(res[i][:, 0], res[i][:, 1],c=colors[i],
                cmap=matplotlib.colors.ListedColormap(colors),marker='o')
plt.legend(*scatter.legend_elements())
plt.show()
```



[5 pts] Question 2d.) Take the eigenvector / eigenvalue decomposition of P^T and subsequently, take their product. That is to say,

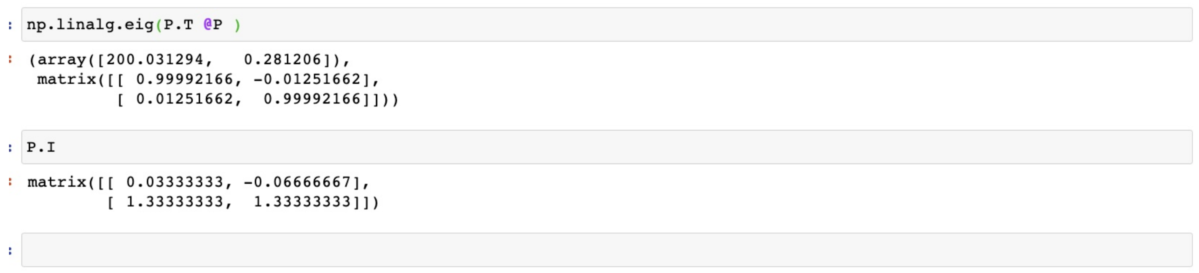
$$\{\Lambda, \Phi\} = \texttt{eig}\left(P^T\right)$$

where $\Lambda = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$ and Φ is a 2×2 matrix with $\phi_i \in \mathbb{R}^2$, a column in Φ. Calculate a new P' such that

$$P' = \Lambda\Phi$$

What is the relationship between P' and the data?
The eigenvectors of **np.linalg.eig(P.T@P)** is very similar to each individual cluster PCA components.

Q2 d)



```
: np.linalg.eig(P.T @P )

: (array([200.031294,   0.281206]),
   matrix([[ 0.99992166, -0.01251662],
           [ 0.01251662,  0.99992166]]))

: P.I

: matrix([[ 0.03333333, -0.06666667],
          [ 1.33333333,  1.33333333]])

:
```

Market Basket Analysis and Algorithms

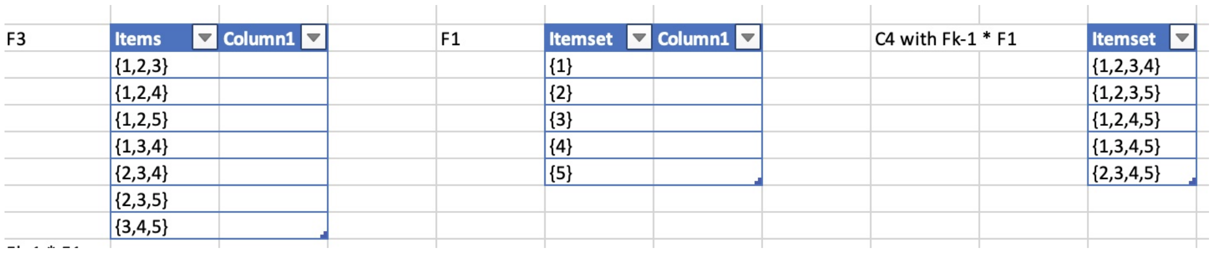
Consider F_3 as the following set of frequent 3-itemsets:

```
{1, 2, 3}, {1, 2, 4}, {1, 2, 5}, {1, 3, 4},
{2, 3, 4}, {2, 3, 5}, {3, 4, 5}.
```

Assume that there are only five items in the data set.

Question 3 [25 pts total]

[10 pts] Question 3a.) List all candidate 4-itemsets obtained by a candidate generation procedure using the $F_{k-1} \times F_1$ merging strategy.



F3

Items	Column1
{1,2,3}	
{1,2,4}	
{1,2,5}	
{1,3,4}	
{2,3,4}	
{2,3,5}	
{3,4,5}	

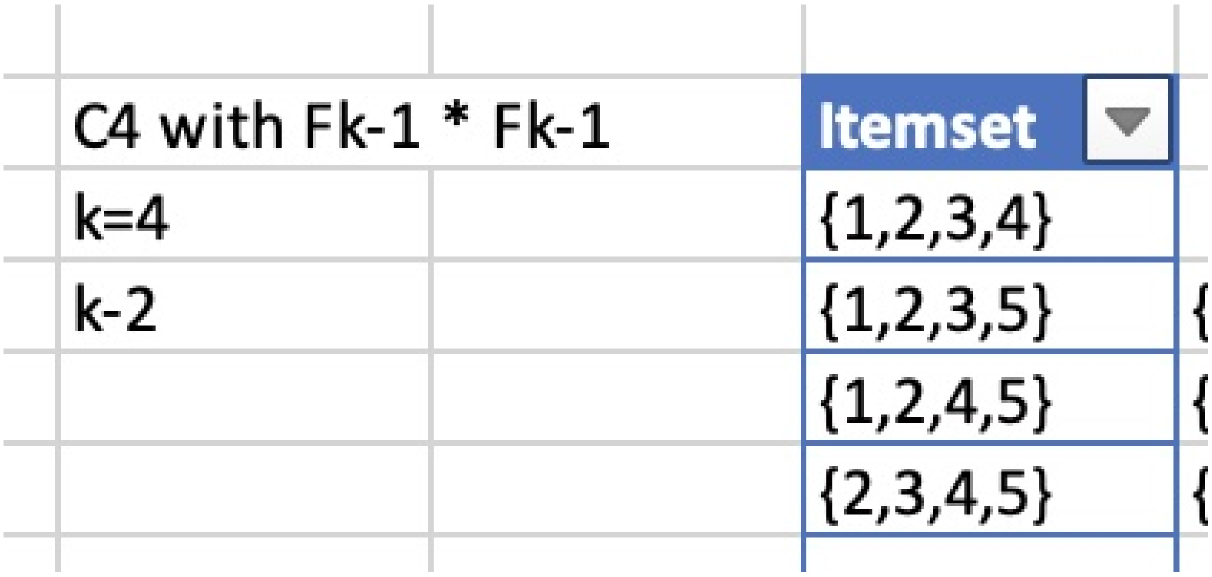
F1

Itemset	Column1
{1}	
{2}	
{3}	
{4}	
{5}	

C4 with Fk-1 * F1

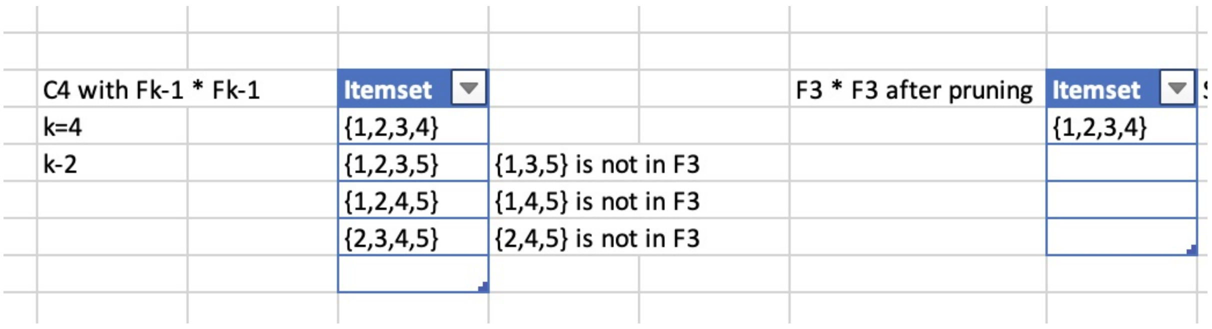
Itemset
{1,2,3,4}
{1,2,3,5}
{1,2,4,5}
{1,3,4,5}
{2,3,4,5}

[10 pts] Question 3b.) List all candidate 4-itemsets obtained by the candidate generation procedure in A Priori, using $F_{k-1} \times F_{k-1}$.



C4 with Fk-1 * Fk-1		Itemset
k=4		{1,2,3,4}
k-2		{1,2,3,5}
		{1,2,4,5}
		{2,3,4,5}

[5 pts] **Question 3c.)** List all candidate 4-itemsets that survive the candidate pruning step of the Apriori algorithm.



C4 with Fk-1 * Fk-1		Itemset			F3 * F3 after pruning	Itemset
k=4		{1,2,3,4}				{1,2,3,4}
k-2		{1,2,3,5}	{1,3,5} is not in F3			
		{1,2,4,5}	{1,4,5} is not in F3			
		{2,3,4,5}	{2,4,5} is not in F3			

Question 4 [25 pts total]

Consider the following table for questions 4a) to 4c):

Transaction ID	Items
1	**{Beer, Diapers}**
2	**{Milk, Diapers, Bread, Butter}**
3	**{Milk, Diapers, Cookies}**
4	**{Bread, Butter, Cookies}**
5	**{Milk, Beer, Diapers, Eggs}**
6	**{Beer, Cookies, Diapers}**
7	**{Milk, Diapers, Bread, Butter}**
8	**{Bread, Butter, Diapers}**
9	**{Bread, Butter, Milk}**
10	**{Beer, Butter, Cookies}**

[3 pts] Question 4a.) What is the maximum number of association rules that can be extracted from this data (including rules that have zero support)?

[3 pts] Question 4b.) What is the confidence of the rule {Milk, Diapers} $\Rightarrow$ {Butter}?

[3 pts] Question 4c.) What is the support for the rule {Milk, Diapers} $\Rightarrow$ {Butter}?

[3 pts] Question 4d.) `True` or `False` with an explanation: Given that {a,b,c,d} is a frequent itemset, {a,b} is always a frequent itemset.

[3 pts] Question 4e.) `True` or `False` with an explanation: Given that {a,b}, {b,c} and {a,c} are frequent itemsets, {a,b,c} is always frequent.

[3 pts] Question 4f.) `True` or `False` with an explanation: Given that the support of {a,b} is 20 and the support of {b,c} is 30, the support of {b} is larger than 20 but smaller than 30.

[3 pts] Question 4g.) `True` or `False` with an explanation: In a dataset that has 5 items, the maximum number of size-2 frequent itemsets that can be extracted (assuming minsup > 0) is 20.

[4 pts] Question 4h.) Draw the itemset lattice for the set of unique items $\mathcal{I} = \{a, b, c\}$.

Q4 a) There are 7 products listed.

$$\# \text{ of rules} = 3^{d} - 2^{d+1} + 1 = 1932$$

4 b) $$c(X \to Y) = \frac{N(Y|X)}{N(X)} = \frac{N(Y,X)}{N(X)} = \frac{4}{6} = \frac{2}{3}$$

$X \downarrow \{M,D\}$ $Y \downarrow \{\text{Butter}\}$

4 c) $$s(X \to Y) = \frac{N(X,Y)}{N} = \frac{4}{10} = \frac{2}{5}$$

$X \downarrow \{M,D\}$

4 d) TRUE. We have a principle: if an itemset is frequent, then its subset is frequent.

4 e) TRUE. $\{a,b,c\}$ we view it as F_3.

We know $F_2\ \{a,b\}$ and $F_2\ \{a,c\}$ are frequent

So according to $\boxed{F_{k-1} \cdot F_{k-1}}$, where $k = 3$, $F_3\ \{a,b,c\}$ is frequent

$\boxed{k-2}$

$\boxed{k} \geq 3$

4 f) FALSE. $s(\{a,b\}) = \dfrac{N(a,b)}{N} = 20$, so # of b ≥ 20

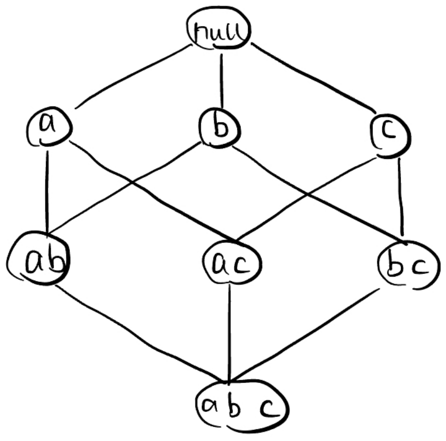
$s(\{b,c\}) = \dfrac{N(b,c)}{N} = 30$, so # of b ≥ 30

$s(\{b\}) = \dfrac{N(b)}{N} \geq 30$

4 g) FALSE. $\binom{5}{2} = \dfrac{5!}{2!\,3!} = 10$

Even if all itemsets are frequent, only 10 of them are 2-item.

4 h)



null
a
b
c
ab
ac
bc
abc