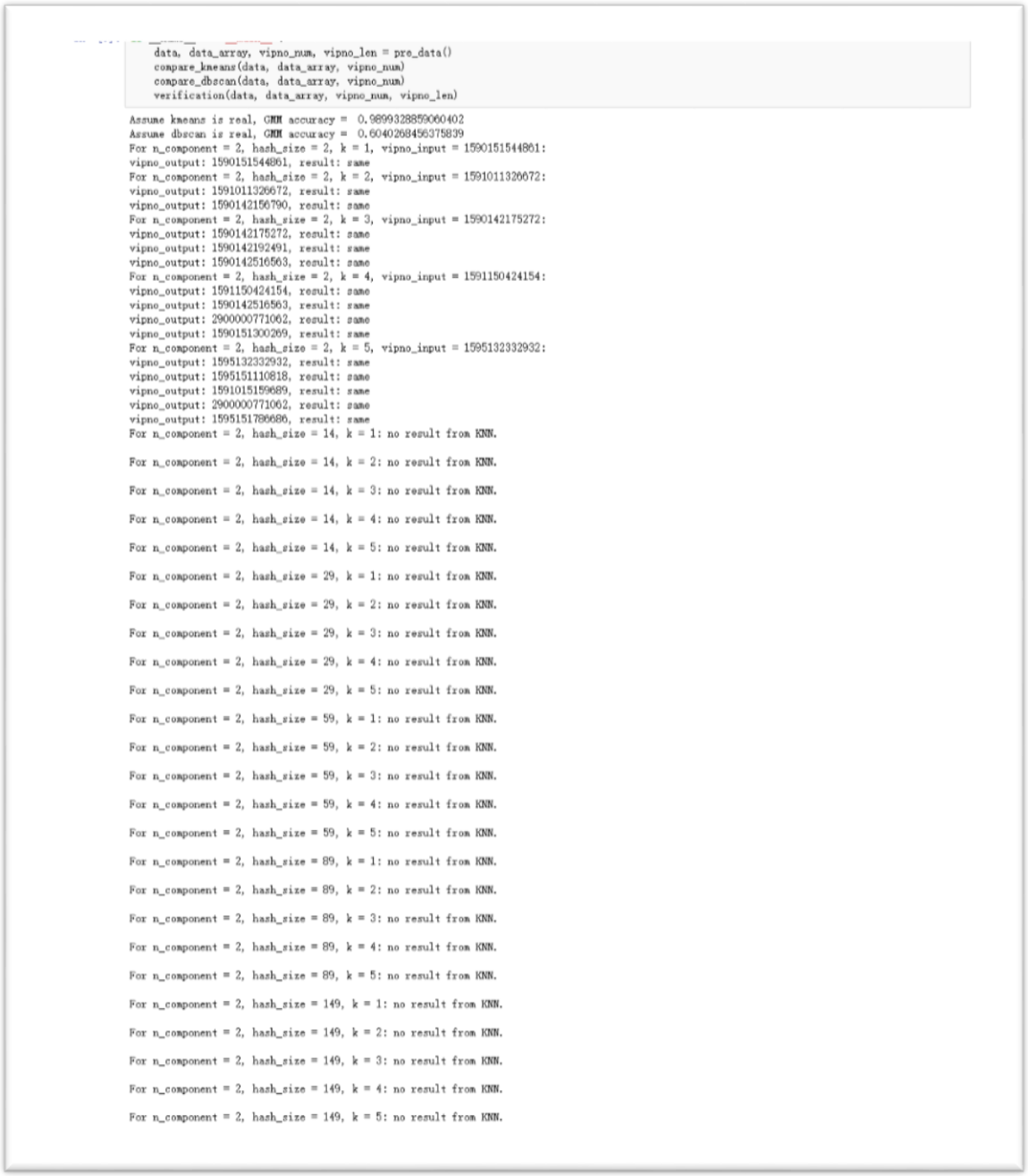
Question 4 报告

一、运行结果

运行结果如下：K-Means 最佳 k 值取 2，DBSCAN 最佳 eps 值取 103



```
data, data_array, vipno_num, vipno_len = pre_data()
compare_kmeans(data, data_array, vipno_num)
compare_dbscan(data, data_array, vipno_num)
verification(data, data_array, vipno_num, vipno_len)
```

```
Assume kmeans is real, GMM accuracy =  0.9899328859060402
Assume dbscan is real, GMM accuracy =  0.6040268456375839
For n_component = 2, hash_size = 2, k = 1, vipno_input = 1590151544861:
vipno_output: 1590151544861, result: same
For n_component = 2, hash_size = 2, k = 2, vipno_input = 1591011326672:
vipno_output: 1591011326672, result: same
vipno_output: 1590142156790, result: same
For n_component = 2, hash_size = 2, k = 3, vipno_input = 1590142175272:
vipno_output: 1590142175272, result: same
vipno_output: 1590142192491, result: same
vipno_output: 1590142516563, result: same
For n_component = 2, hash_size = 2, k = 4, vipno_input = 1591150424154:
vipno_output: 1591150424154, result: same
vipno_output: 1590142516563, result: same
vipno_output: 2900000771062, result: same
vipno_output: 1590151300269, result: same
For n_component = 2, hash_size = 2, k = 5, vipno_input = 1595132332932:
vipno_output: 1595132332932, result: same
vipno_output: 1595151110818, result: same
vipno_output: 1591015159689, result: same
vipno_output: 2900000771062, result: same
vipno_output: 1595151786686, result: same
For n_component = 2, hash_size = 14, k = 1: no result from KNN.

For n_component = 2, hash_size = 14, k = 2: no result from KNN.

For n_component = 2, hash_size = 14, k = 3: no result from KNN.

For n_component = 2, hash_size = 14, k = 4: no result from KNN.

For n_component = 2, hash_size = 14, k = 5: no result from KNN.

For n_component = 2, hash_size = 29, k = 1: no result from KNN.

For n_component = 2, hash_size = 29, k = 2: no result from KNN.

For n_component = 2, hash_size = 29, k = 3: no result from KNN.

For n_component = 2, hash_size = 29, k = 4: no result from KNN.

For n_component = 2, hash_size = 29, k = 5: no result from KNN.

For n_component = 2, hash_size = 59, k = 1: no result from KNN.

For n_component = 2, hash_size = 59, k = 2: no result from KNN.

For n_component = 2, hash_size = 59, k = 3: no result from KNN.

For n_component = 2, hash_size = 59, k = 4: no result from KNN.

For n_component = 2, hash_size = 59, k = 5: no result from KNN.

For n_component = 2, hash_size = 89, k = 1: no result from KNN.

For n_component = 2, hash_size = 89, k = 2: no result from KNN.

For n_component = 2, hash_size = 89, k = 3: no result from KNN.

For n_component = 2, hash_size = 89, k = 4: no result from KNN.

For n_component = 2, hash_size = 89, k = 5: no result from KNN.

For n_component = 2, hash_size = 149, k = 1: no result from KNN.

For n_component = 2, hash_size = 149, k = 2: no result from KNN.

For n_component = 2, hash_size = 149, k = 3: no result from KNN.

For n_component = 2, hash_size = 149, k = 4: no result from KNN.

For n_component = 2, hash_size = 149, k = 5: no result from KNN.
```

二、分析

1. GMM 算法：

   GMM 算法是由多个高斯分布组成，每个高斯分布构成一个 component。对于数据源，可以先假定它们都是由 GMM 生成出来的，那么我们只要根据数据推出 GMM 的概率分布来就可以了，推出的分布中，每个高斯分布对应了一个簇。

2. 对比

a) 假定 K-Means 为真实的

此时得到的 accuracy 约为 0.99，这是因为 GMM 算法和 K-Means 算法很像。他们的共同点在于他们都是通过不断迭代求解，而且迭代的策略都相同，不同之处在于需要计算的参数不同，而且 GMM 迭代一次需要花更多的时间。

b) 假定 DBSCAN 为真实的

更 DBSCAN 相比，得到的 accuracy 只有 0.6，这很有可能是与 DBSCAN 采用的 eps=103,min_sample=2 有关，这样取值是为了保证最后得到两个簇，不过由于前两题的结论。我试着将参数换成 eps=300,min_sample=4:

```
Assume kmeans is real, GMM accuracy =  0.9899328859060402
Assume dbscan is real, GMM accuracy =  0.959731543624161
```

可以看到此时 accuracy 高了许多，而且此时 GMM 的 n_component 值为 1，进一步说明假设是对的。

3. 验证分析

将 knn 的结果带入 GMM 算法中进行验证，验证时 n_component=2，由最上面的图可以看出验证成功。

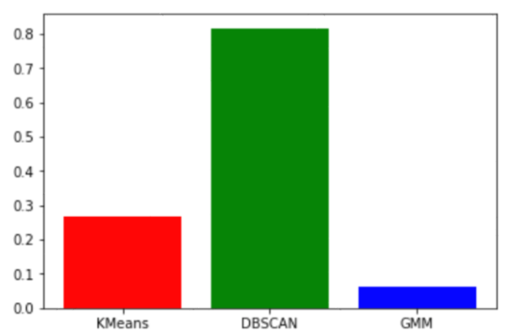
不妨将此时的标签也打印出来

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0]
```

现在基本可以得出结论了，给出的数据中除了少量的噪点之外，其余的数据都是相似的。

三、性能

分别计算出三个算法生成聚类的过程所花的时间



KMeans: 0.26888124739721875 DBSCAN: 0.8163646440417477 GMM: 0.06372097385018738

可以看到，DBSCAN 需要花费的时间最多，而 kmeans 和 GMM 都是迭代算法，正常情况下 GMM 迭代需要更多时间，出现这种情况的原因可能是由于初始数据符合高斯分布，GMM 迭代的次数非常少。