**一、课题综述**

* 1. **课题说明**

任务划分：

-Data Prepare：何诗锟

-Data Preprocess： 彭坤宇

-Model Construct： 刘子牛，彭坤宇，张祖豪

-Train & Test：

-Plot Result：

-Optimize & Review：何诗锟

* 1. **课题目标**

本课题需要我们完成一项聚类实验，并且针对同一个聚类任务，至少给出两种课上提到的聚类方法，然后对以上的聚类方法进行比较分析，这里我们选择了使用K-means聚类，DBSCAN的密度聚类和层次聚类。同时需要选择具有一定新颖性的数据集，任务较为有趣。

* 1. **课题数据集**

本课题使用的数据集为来自UCI的Online Retail，这是一个跨国数据集，包含了在2010年12月1日至2011年9月12日期间发生的所有交易，涉及一家总部位于英国并注册的非实体店在线零售公司。该公司主要销售独特的全场合礼品，许多客户是批发商。数据集各特征值解释如下。

* InvoiceNo（发票号）：每笔交易的唯一标识符，如果以字母 'c' 开头，表示该记录是取消订单。
* StockCode（库存代码）：每个产品的唯一标识符，由5位整数组成。
* Description（描述）：产品名称，描述每个产品的文字信息。
* Quantity（数量）：每种产品在每笔交易中的数量。
* InvoiceDate（发票日期）：记录每笔交易生成的日期和时间。
* UnitPrice（单价）：每个产品的单价，以英镑为货币单位。
* CustomerID（客户ID）：每个客户的唯一标识符，由5位整数组成。
* Country（国家）：每个客户所居住的国家的名称。

我们的目标是根据RFM（最近一次购买时间、购买频率、购买金额）对客户进行分组，以便公司能够更有效地定位目标客户。并且这个任务也在Kaggle竞赛网站上能够找到。（[Online Retail K-Means & Hierarchical Clustering | Kaggle](https://www.kaggle.com/code/hellbuoy/online-retail-k-means-hierarchical-clustering)）。

**二、实验设计**

* 1. **数据准备**

经过筛选，最后选择了OnlineRetail作为聚类的数据集，网购零售业的数据现在已经呗广泛的使用在各个APP的算法中，使用聚类算法能够发现不同用户群体的爱好和特点，从而能够更加精准的推荐相关的内容，该数据集与现实生活息息相关，具有一定新颖性，选择它比较符合我们的需要。

* 1. **数据预处理**

数据集已经经过Kaggle上传人的处理，不存在缺失值，因此可以省略剔除异常值和补充缺失值的过程。

由于原本数据集中给出的数据用来聚类特征不会非常明显，有些数据单个使用起来效果不会很好，所以我们不能直接那原本的数据集中特征值来采用聚类算法，需要通过原数据集的各个特征值进行一下分析和计算得到新的特征，这里我们选择采用**RFM模型**，它是根据用户的价值划分用户群体，从而可以针对不同的用户群体做运营策略以达到挖掘用户价值的模型方法。

RFM模型是通过一个客户的近期购买行为、购买的总体频率以及花了多少钱3个维度来描述该客户价值状况的客户分类模型，这3个维度分别表示为：

* 最近一次消费距离现在的时间 (Recency)：这个值越小越好；
* 某段时间内消费频率次数 (Frequency)：这个值越大越好；
* 某段时间内消费金额 (Monetary)：这个值越大越好；

所以我们的数据预处理阶段,就需要把这些值给从原来的数据集中计算出来。然后我们还要进行异常值的处理，将一些噪声点去除掉，这里我们选择采用**IQR四分位距**方法。它是一种基于数据集中的四分位数计算的统计技术，通过计算数据分布的四分位数范围，识别并移除数据中的异常值。

**步骤如下：**

* 计算购买金额 (df['Amount'])：

将 'Quantity'（购买数量）和 'UnitPrice'（商品单价）相乘，得到每个交易的购买金额，并将其存储在名为 'Amount' 的新列中。

* 计算购买金额总和 (df\_Monetary)：

使用 groupby 按 'CustomerID' 对数据进行分组，然后对每个客户的购买金额 ('Amount') 进行求和。

* 计算购买频率 (df\_Frequency)：

使用 groupby 按 'CustomerID' 对数据进行分组，然后对每个客户的交易次数 ('InvoiceNo') 进行计数。将 'InvoiceDate' 列的数据转换为 pandas 的日期时间格式。

计算每个交易距离数据集中最后一次交易的天数，并将结果存储在 'No\_buy' 列中。

使用 groupby 按 'CustomerID' 对数据进行分组，然后找到每个客户的最小 'No\_buy'（即最后一次购买距今天数）。

* 合并特征 (pd\_MF, pd\_MFR)：

使用 pd.merge 将购买金额总和 (df\_Monetary)、购买频率 (df\_Frequency)、最后一次购买时间 (df\_Recency) 合并为一个 DataFrame。合并过程基于 'CustomerID' 列进行，采用内连接方式（how='inner'）。最后，为合并后的 DataFrame 设置列名为 ['CustomerID', 'Amount', 'Frequency', 'Recency']。

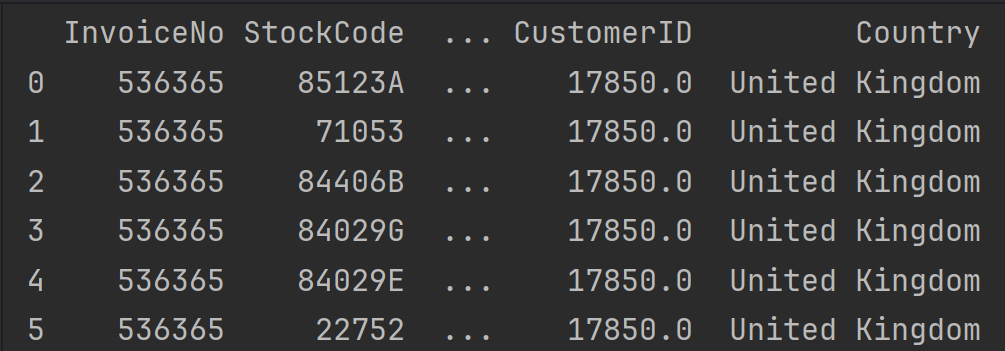
* 移除异常值：

对 'Amount'、'Recency'、'Frequency' 进行异常值处理，使用 IQR 方法。计算每个特征的 Q1（第一四分位数）、Q3（第三四分位数）、IQR（四分位数范围）。移除超出范围 [Q1 - 1.5 \* IQR, Q3 + 1.5 \* IQR] 的数据点。

* 标准化

使用StandardScaler对特征进行标准化处理，将数据转换为均值为0、标准差为1的标准正态分布。

处理结果如下：

 文本

描述已自动生成

后续使用到三个不同的聚类方法时，我们采用的数据预处理时相同的。

* 1. **模型搭建**
* **模型1：DBSCAN模型**

DBSCAN模型采用了sklearn机器学习库来实现。

（1）使用DBSCAN算法对标准化后的数据 df\_scaled 进行聚类。

1. db = DBSCAN(eps=0.3, min\_samples=15).fit(df\_scaled)
2. score = metrics.silhouette\_score(X, df\_scaled["Clus\_Db"])

eps参数表示邻域半径，设置为0.3。

min\_samples参数表示邻域内最小样本数，设置为15。

（2）标记核心样本：

1. core\_samples\_mask = np.zeros\_like(db.labels\_, dtype=bool)
2. core\_samples\_mask[db.core\_sample\_indices\_] = True

创建一个布尔数组 core\_samples\_mask，用于标记核心样本。

根据DBSCAN的结果，将核心样本的索引位置标记为True。

（3）获取聚类标签：

1. labels = db.labels\_
2. df\_scaled["Clus\_Db"] = labels
3. df\_scaled.sort\_values("Clus\_Db")

将DBSCAN的聚类标签存储在 labels 变量中。

将聚类标签列加入原始的标准化后的DataFrame df\_scaled，并将该列命名为 "Clus\_Db"。

（4）统计聚类信息：

1. realClusterNum = len(set(labels)) - (1 if -1 in labels else 0)
2. clusterNum = len(set(labels))

计算实际簇的数量 realClusterNum，排除噪声点（标签为-1）。

计算总簇的数量 clusterNum。

（5）计算轮廓系数

1. score = metrics.silhouette\_score(X, df\_scaled["Clus\_Db"])

* **模型2：层次聚类模型**

层次聚类模型采用了两种实现方式，一种是手写实现，一种是调用sklearn中的层次聚类库来进行实验。

1. 手写实现层次聚类（速度比较慢）

* 首先先初始化每个点为一个聚类
* 然后计算距离矩阵，存储每对数据点之间的欧氏距离

1. dist\_matrix = np.zeros((X.shape[0], X.shape[0]))
2. for i in range(X.shape[0]):
3. for j in range(i+1, X.shape[0]):
4. dist\_matrix[i][j] = dist\_matrix[j][i] = np.linalg.norm(X[i]-X[j])

* 当聚类数量大于指定的 n\_clusters 时，进行循环。在循环中，找到最近的两个聚类 (c1 和 c2)，将它们合并，并更新距离矩阵。

1. while len(clusters) > n\_clusters:
2. *# 找到最近的两个聚类*
3. min\_dist = np.inf
4. c1 = c2 = 0
5. for i in range(len(clusters)):
6. for j in range(i+1, len(clusters)):
7. dist = np.min(dist\_matrix[clusters[i]][:, clusters[j]])
8. if dist < min\_dist:
9. min\_dist = dist
10. c1, c2 = i, j
11. *# 合并最近的两个聚类*
12. clusters[c1].extend(clusters[c2])
13. clusters.pop(c2)
14. *# 更新距离矩阵*
15. for i in range(len(clusters[c1])):
16. for j in range(i+1, len(clusters[c1])):
17. dist\_matrix[clusters[c1][i]][clusters[c1][j]] = dist\_matrix[clusters[c1][j]][clusters[c1][i]] = np.linalg.norm(X[clusters[c1][i]]-X[clusters[c1][j]])

* 最终，返回生成的聚类标签。

1. Sklearn库实现

agg = AgglomerativeClustering(n\_clusters=args.agg).fit(df\_scaled)

这个实现比较简单

但是对比起来，显然是sklearn库做的更好，手写实现层次聚类的时候，计算计算距离矩阵的时间复杂度为 O(N2)，迭代都要找到最近的两个聚类，这需要O(N2)，每次合并后，需要更新距离矩阵，这也需要O(N2)，迭代次数取决于初始时聚类的数量和目标聚类的数量，是O(N)，所以复杂度是O(N3)，运行起来比较慢。

使用HC.py还需这样使用：

Python HC.py –agg <your\_target\_classes\_num>

* **模型3：K-Means聚类模型**

这个模型实现起来简单，所以我们采用了手写实现。

首先，选择要分成的簇的数量K。然后，随机初始化K个质心，将数据点分配到离其最近的质心所属的簇。接着，更新每个簇的质心为该簇中所有数据点的平均值。重复进行分配和更新步骤，直到质心不再发生显著变化或达到预定的迭代次数。最终，每个数据点都被分配到一个簇，形成聚类结果。K-Means通过最小化簇内数据点与其质心的平方距离的总和来实现聚类。

我们自己的手写实现方式中，在fit 方法中使用echoes 控制了最大迭代次数，每次迭代中选择部分数据点的子集作为质心进行更新。最终聚类结果存储在 X （输入的特征）中，每个数据点的第一个元素表示所属簇的标签。

这里给出K-Means的两个核心方法的代码。

1. def computeCenter(self):
2. dis = 0
3. for i in range(len(self.Center)):
4. self.Center[i] = np.mean(np.array([self.X[j][1:] for j in range(len(self.X)) if self.X[j][0] == i]), axis=0)
5. if True in np.isnan(self.Center[i]):
6. self.Center[i] = self.X[random.randint(0, len(self.X)), 1:]
7. continue
8. try:
9. dis += np.sum(np.sqrt(
10. np.sum(np.square([self.Center[i] - self.X[j][1:] for j in range(len(self.X)) if self.X[j][0] == i]),
11. axis=1)))
12. except:
13. print(self.X)
14. print(0, self.Center[i], np.isnan(self.Center[i]))
15. print(1, self.Center)
16. tmp = [self.Center[i] - self.X[j][1:] for j in range(len(self.X)) if self.X[j][0] == i]
17. print(2, tmp)
18. print(3, np.sum(np.square(tmp), axis=1))
19. print()
20. self.check = 0
21. return dis
22. def classify(self):
23. for i in range(len(self.X)):
24. dis = 0x7fffffff
25. for j in range(len(self.Center)):
26. if dis > np.sqrt(np.sum(np.square(self.X[i][1:] - self.Center[j][:]))):
27. dis = np.sqrt(np.sum(np.square(self.X[i][1:] - self.Center[j][:])))
28. self.X[i][0] = j

* computeCenter 方法:

此方法用于计算每个簇的质心，并返回簇内数据点与质心的总距离。

遍历每个簇，对于每个簇，通过筛选出属于该簇的数据点，计算这些数据点的均值，作为新的质心。如果计算得到的质心包含 NaN 值，则重新随机选择一个数据点的特征作为质心。在计算质心时，使用了异常处理机制，以防止在计算距离时出现异常（例如，两个簇内的数据点数量不一致）。最终，返回簇内数据点与质心的总距离 dis。

* classify 方法:

此方法用于将每个数据点分配到最近的簇。

对于每个数据点，计算其与每个质心之间的欧氏距离，将数据点分配给距离最近的簇。

将簇的标签（即簇的索引）存储在数据点的第一个元素 self.X[i][0] 中。

* 1. **模型训练测试&结果**

对于每一次实验，计算了轮廓系数（silhouette score）来评估聚类效果。通过Matplotlib的3D散点图可视化了聚类结果，颜色表示不同的聚类簇，坐标轴表示金额、购买频率和最近一次购买时间。

输出了聚类标签、实际簇数、总簇数、和轮廓系数等信息。

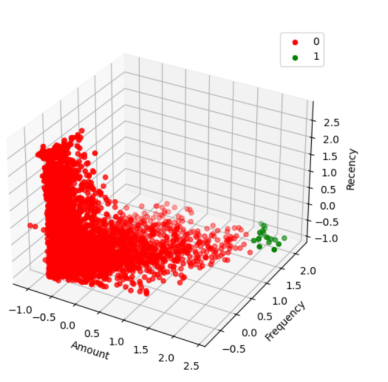
**模型一：DBSCAN模型**

训练结果：

不同eps、min\_samples下的结果。

包含实际cluster的数量，目标cluster数量，和轮廓系数值。

|  |  |  |
| --- | --- | --- |
| eps=0.3 | min\_samples=15 |  |
| eps=0.2 | min\_samples=15 |  |
| Eps=0.1 | Min\_samples=10 |  |

图表, 散点图

描述已自动生成图表

描述已自动生成

分别是3个不同超参数下的执行效果。

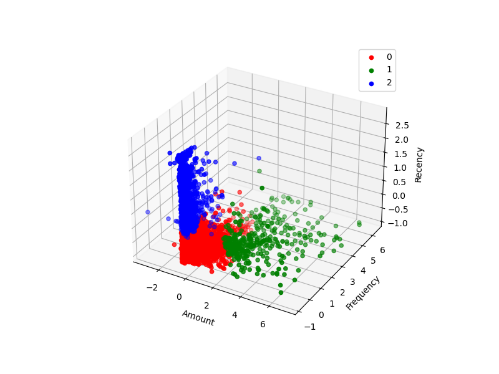
**模型二：层次聚类模型**

与DBSCAN不同，这里唯一的参数就是分类数量K

下面是不同K值情况下的层次聚类分类结果。

|  |  |
| --- | --- |
| K=3 |  |
| K=4 | 手机屏幕的截图  描述已自动生成 |
| K=5 |  |

以下是分类的结果：

 图表, 散点图

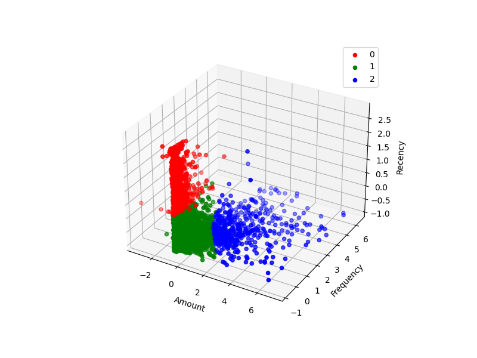
描述已自动生成 图表, 散点图

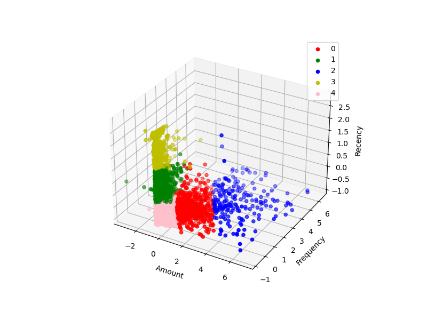
描述已自动生成

**模型三：K-Means聚类**

因为采用了手写的，运行时间也比较长，一次运行需要大约8-10min。

|  |  |
| --- | --- |
| K=3 |  |
| K=4 |  |
| K=5 |  |

 图表, 散点图

描述已自动生成

* 1. **分析和比较**

**分析：**

**评价标准：采用轮廓系数进行评价**

轮廓系数（Silhouette Coefficient）是一种用于衡量数据聚类效果的指标，其值介于-1到1之间。轮廓系数考虑了数据点与其分配的簇的紧密度（内聚度）和与其他簇的分离度。

**取值范围：**

轮廓系数的取值在[-1, 1]之间。一个较高的轮廓系数表示数据点更可能在其分配的簇内部紧密聚集，且远离其他簇。

**计算方式：**

对于每个数据点，计算它与同簇内其他点的平均距离（a）以及与最近的其他簇的所有点的平均距离（b）。​

如果 a 接近0，表示数据点与同簇内其他点的距离很小，说明点在其分配的簇内部紧密聚集。

如果 b 接近0，表示数据点与最近的其他簇的点的距离很小，说明该点在簇之间有很好的分离度。轮廓系数越接近1，表示聚类效果越好。如果轮廓系数为负数，则说明数据点更可能被错误地分配到了不适当的簇中。

**总结：**

在没有真实标签的情况下，轮廓系数是一种评估聚类效果的常用指标。

对于每个簇的平均轮廓系数的均值可以作为整体聚类效果的衡量。

总体而言，轮廓系数提供了一种直观的方式来评估聚类的紧密度和分离度，值越高表示聚类效果越好。

**超参数**：

**DBSCAN聚类方法：**

在DBSCAN（Density-Based Spatial Clustering of Applications with Noise）算法中，eps（邻域半径）和min\_samples（邻域内最小样本数）是两个关键的参数，它们会显著影响聚类的结果。

* eps（邻域半径）：

eps 是一个正数，代表数据点的邻域半径。具体而言，对于一个数据点，DBSCAN将以它为中心，半径为 eps 的邻域内的数据点都视为邻居。增大 eps 将扩大邻域，更多的点将被认为是邻居，导致更大的簇。缩小 eps 将减小邻域，使得更少的点被认为是邻居，导致更多的小簇或噪声点。

* min\_samples（邻域内最小样本数）：

min\_samples 是一个正整数，表示一个簇中至少应包含的数据点的数量。增大 min\_samples 将导致算法更加严格，只有邻域内数据点数量达到或超过 min\_samples 才能形成簇，这可能会减少噪声点的影响。减小 min\_samples 将导致算法更容易形成簇，即使簇的密度较低，这可能会导致更大的簇和更多的噪声点。

* 影响：

eps 和 min\_samples 交互影响： 选择合适的 eps 和 min\_samples 需要在寻找适当簇的平衡中权衡。较大的 eps 和较小的 min\_samples 可能导致更大的簇，而较小的 eps 和较大的 min\_samples 可能导致更小的簇。

噪声敏感性： DBSCAN对这两个参数的选择非常敏感，特别是对于噪声点的处理。选择合适的参数可以有效地将噪声点标记为孤立点。

密度不均匀数据： 在处理密度不均匀的数据时，适当选择 eps 和 min\_samples 是关键的，以确保算法能够正确地捕捉簇的结构。

在使用DBSCAN时，通常需要通过尝试不同的参数值来进行调优，可以借助验证指标（如轮廓系数）来评估不同参数组合下的聚类效果。

**对于层次聚类和K-Means：**

我们实现的层次聚类和K-Means聚类有关的超参数就是需要聚类的数量。

从经过处理后的散点图来看，绝大部分点都聚集在左下角部分，实则我们很难去确定究竟该分为几类比较好。而且运行时间比较慢，不好做K和轮廓系数的图像，所以我们这里给出了几种比较正常的K=3\4\5的分类。可以看到聚类的效果都还是比较正常的。

那我们也可以通过以下几种方法来确定我们的K值

* 肘部法则（Elbow Method）： 通过绘制不同 K 值下的聚类误差（样本到簇中心的平方距离）曲线，找到“肘部”，即误差开始迅速减小的点，这个点对应的 K 值通常是一个合适的聚类数量。
* 轮廓系数（Silhouette Score）： 轮廓系数考虑了簇内的紧密度和簇间的分离度，值越大表示聚类效果越好。
* Gap 统计量： 比较聚类结果和随机数据集的差异，找到最合适的 K 值。

**比较：**

我们在实验中运用到了三种聚类方法。从上面的实验结果可以看出，由于数据集的聚集程度的原因，使用不需要指定K的方法时（DBSCAN，虽然层次聚类也不需要指定K，但我们实现的和sklearn的是支持指定K的），很明显的看出聚类的效果稍微差了些，轮廓系数的分数也没有后面两个好，而且类与类之间的分界线很模糊，有些超参数下更是直接只有一类了。

对于K-Means和层次聚类来说，我们分类的效果就比较好，虽然直观上我们也很难看出数据集上的明显的分界。但这就是聚类方法的特性。有些人们肉眼难以看出的分界它也能看出。这两种方法的轮廓系数要比DBSCAN好一些。

最后，总结一下三种方法的优劣和适用场景。

**DBSCAN：**能够发现任意形状的聚类，并且对密度变化敏感。能够自动识别和排除噪声点，不受异常值的干扰。但受制于超参数eps和min\_samples，同时不适用于高维的数据。

**层次聚类：**层次聚类能够生成层次结构，对于理解数据的层次性质很有帮助。但计算复杂度较高，不适用于大规模数据集。也不适用于非凸形状的簇。

**K-Means：**计算效率高，适用于大规模数据集。简单易于理解，聚类结果也相对容易解释。但K-Means对初始点选择敏感，不同初始点导致不同的聚类结果。

**适用场景选择：**

* 如果数据集具有明显的密度变化和噪声，可以考虑使用 DBSCAN。
* 如果数据集具有层次结构，可以考虑使用层次聚类。
* 如果数据集具有明显的凸形状簇，且对计算效率要求较高，可以考虑使用 K-Means。

1. **总结**

本次实验为课程的第三次实验，对机器学习的分类任务有了一定的体会。小组合作方面更加的成熟。分工更加明确，每个人都曾做过不同工作，所以大家都对整体的实验流程都能够掌握，这样认识更加的全面。

在我们的聚类实验中，一共采用了三种不同的聚类算法：DBSCAN、层次聚类和K-Means，并结合了RFM模型作为数据预处理的方法。这次多模型的选择和综合运用使得实验更加全面和多角度。在这个过程中，我们对不同的聚类方法有了更加深入的理解：

首先，DBSCAN算法通过密度的概念，能够有效地发现数据中的任意形状的簇，而不需要提前指定簇的数量。这使得它在处理不规则形状的簇或噪声较多的数据集时表现出色。

其次，层次聚类通过自底向上或自顶向下的层次划分，形成层次结构的聚类。这种方法在数据集存在明显层次结构或嵌套关系时表现良好。它提供了一种直观的聚类结果可视化方式，并且可以通过截断树状图来灵活控制聚类的数量。

K-Means是一种迭代的、基于距离的聚类方法，通过最小化簇内数据点与质心的距离来进行聚类。它适用于各种形状的簇，并且在高维数据集中通常具有较好的性能。

为了达到更好的训练效果，我们在数据预处理阶段，采用了RFM模型，将顾客的购买行为转化为三个维度：最近一次购买时间（Recency）、购买频率（Frequency）和购买金额（Monetary）。这样的预处理能够更好地反映顾客的行为特征，为聚类算法提供更有价值的输入。

总的来说，通过结合不同的聚类算法和数据预处理手段，我们能够更全面地理解数据的分布情况，发现不同模型的优劣势，并根据实际情况选择最合适的聚类结果。这种综合运用的方法能够更好地适应不同类型和特征的数据集，提高聚类结果的鲁棒性和可解释性。

**这次实验是我们本学期的最后一次实验。**

非常感谢每一位小组成员在整个学期中在机器学习实验中的不懈努力和合作。回顾这段时间，我深刻体会到了团队的力量和协同合作的重要性。在这个机器学习实验的过程中，我们面临了各种挑战和问题，但每一位小组成员都展现了出色的团队协作精神。大家不仅仅是完成自己分工的任务，更是在整个团队中共同努力，相互支持。我们通过积极的讨论，分享各自的见解和经验，使得整个实验过程更加充实和有趣。每个人都在自己的领域内做出了出色的贡献，无论是在数据预处理、模型选择还是实验结果的分析上。通过团队合作，我们不仅仅学到了更多关于机器学习的知识，还培养了解决问题的能力和团队协作的技巧。大家在合作中展现的才华和毅力让我深感荣幸能够成为这个团队的一部分。我想特别感谢每一位小组成员为实验所做出的努力和奉献。大家的付出为我们的小组带来了巨大的成功，也让这个学期的机器学习实验成为了一段难忘的经历。希望我们在未来的学习和工作中，能够保持这种团队协作的精神，共同迎接更多的挑战。

最后，感谢每一位在这次机器学习实验中付出心血的小组成员，祝愿大家在未来的学习和事业中都能够取得更大的成就。我们一路走来，收获满满，愿未来的道路上我们仍能携手前行，共同迎接更多的挑战。再次感谢大家的努力和合作！

参考文献：

[1]数据分析实战系列（一）RFM模型：<https://zhuanlan.zhihu.com/p/98488644>

[2] 数据分析IQR（Interquartile Range）四分位距的理解与应用及Python实现：https://blog.csdn.net/redrose2100/article/details/130211842