# Weather Learn 项目报告

1. 引言

该项目是对mongodb中weatherWarning数据库的分析，总共包括连接数据库、数据获取和保存、简易的可视化、运用kmeans算法进行数据分析四个部分。该报告解释了大部分代码的含义，并且展示了分析后部分结果。

1. 连接数据库

数据库的连接被封装为类mongoToPandas，保存在mongoToPandas.py中。除了初始化函数外，类mongoToPandas还有两个方法：read\_mongo和count，用于之后的数据获取。

初始化函数：由用户输入所需mongdb数据库的服务器和路径，调用pymongo中的MongoClient进行数据库连接。同时，初始化类方法read\_mongo和count的返回变量：self.df（数据类型为dataframe）和self.cursor（数据类型为list）。

类方法read\_mongo(query, no\_id): 根据用户输入的query对数据库进行查询操作，结果存放在类属性self.df中。由于数据库中的数据id这一属性有重复，故设置参数no\_id。当参数no\_id为True时，类属性self.df将会删除重复的数据id属性。

类方法count(para): 根据用户输入的属性名para，返回数据库中该属性名中所有不相同的值。

之后的数据获取多次仅仅用到数据库的查询和去重操作，因此将数据库的连接封装为类，只提供对应操作的类方法，是合理的。

1. 数据获取和保存

数据的获取和保存被封装在类weatherLearn中，分别对应load\_data和write\_sheet两个类方法。笔者默认将所有天气数据按照城市分类，存放在weatherLearn.xlsx的不同工作薄中。

初始化函数：用类mongoToPandas连接数据库，获取数据库中所有城市并对其进行排序，保存为类属性self.places。初始化用于保存数据的类属性self.data（数据类型为list，数据索引与类属性self.places一致）；创建表格weatherLearn.xlsx。

load\_data(): 遍历self.places中的所有城市，通过类mongoToPandas的read\_mongo方法筛选出该城市所有的天气数据，按照时间排序，根据该城市在self.places中的索引保存在self.data中。

write\_sheet(): 遍历self.places中的所有城市，将self.data中对应的数据写入weatherLearn.xlsx对应的工作薄中。

1. 数据可视化

获取和保存数据后，我们对数据进行简单的可视化。该部分调用了交互式可视化库bokeh，保存在analysisWeather.py中。由于这部分代码只涉及页面布局，数据仅作为可视化的来源，故在此不再分析代码原理，仅展示可视化结果。

在终端输入指令bokeh serve –show analysisWeather.py后，弹出交互式网页，可选择Warn Color和Warn Type两个标签页作为查询类型。

以Warn Type为例，统计上海市嘉定区各种灾害类型的天数。

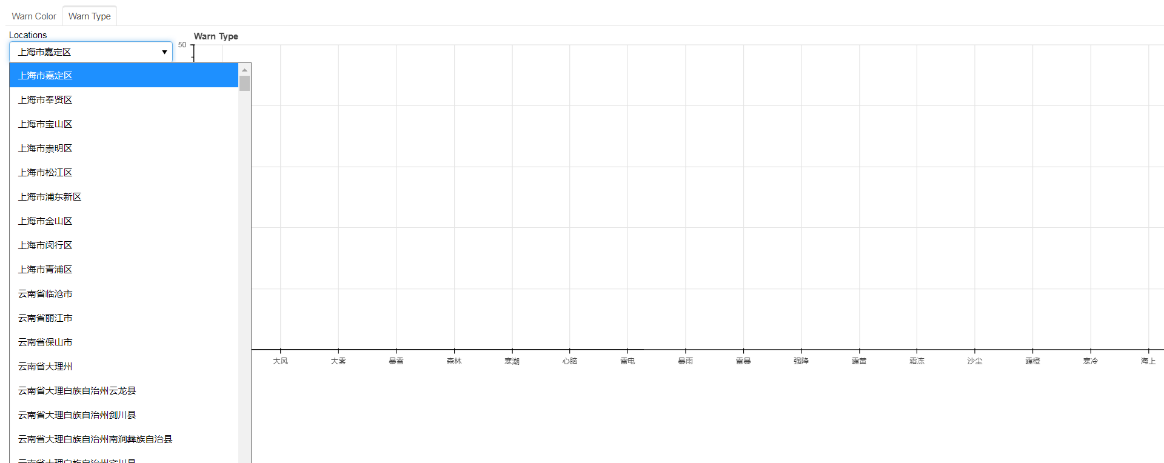


图1：交互按钮选择查询地点

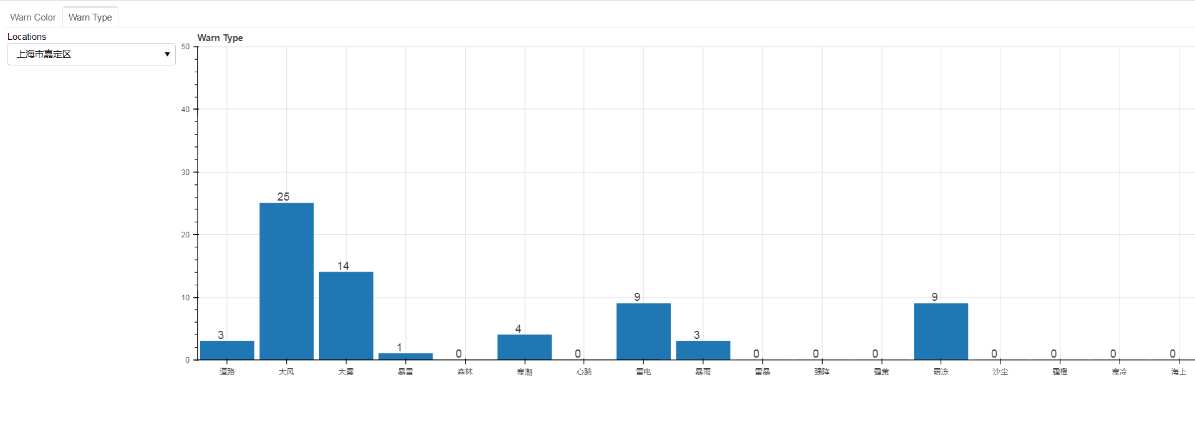


图2：查询结果

1. 数据分析（Kmeans模型+可视化）
   1. Kmeans模型实现+可视化

根据生活经验，城市可依据气象预警类型划分从两个维度划分：南方和北方，内陆和沿海。根据以上思想，我们选择Kmeans算法对城市进行分类。Kmeans算法的实现保存在kmeans.py中。接下来介绍kmeans.py中函数的用途。

distance(vec1, vec2): 返回两个向量的欧氏距离

loadDataSet(): 初始化应用kmeans算法的数据集，数据集的类型为字典，数据集的键为各个城市，值为对应气象预警类型出现的次数构成的list。

initCentroids(dataset, k): 初始随机选取k个数据作为质心

minDistance(dataset, centroidList): 对每个数据集中的数据，计算数据和现有的每个质心的距离，将数据放入对应的簇中

getCentroids(clusterDict): 计算每簇的均值，得到对应的质心

getVar(clusterDict, centroidList):计算簇集合间的均方误差

kmeans流程如下:

表1：kmeans流程

初始化质心，更新数据集的距离

计算簇集合间的均方误差

重新计算质心，更新距离，多次迭代

前后两次均方误差相等，迭代结束

函数write\_dict将以类别作为键的字典clusterDict转换为以地名作为键的字典locDict，以便数据的输出和保存。函数write\_excel将字典locDict数据写入对应表格中。

kmeans可视化部分调用了pyecharts库，源码保存在kmeans\_dv.py中。

* 1. K=4分类结果

选取聚类中心数量k=4，结果保存在kmean.xlsx中，可视化运行结果（kmeans.html）得到

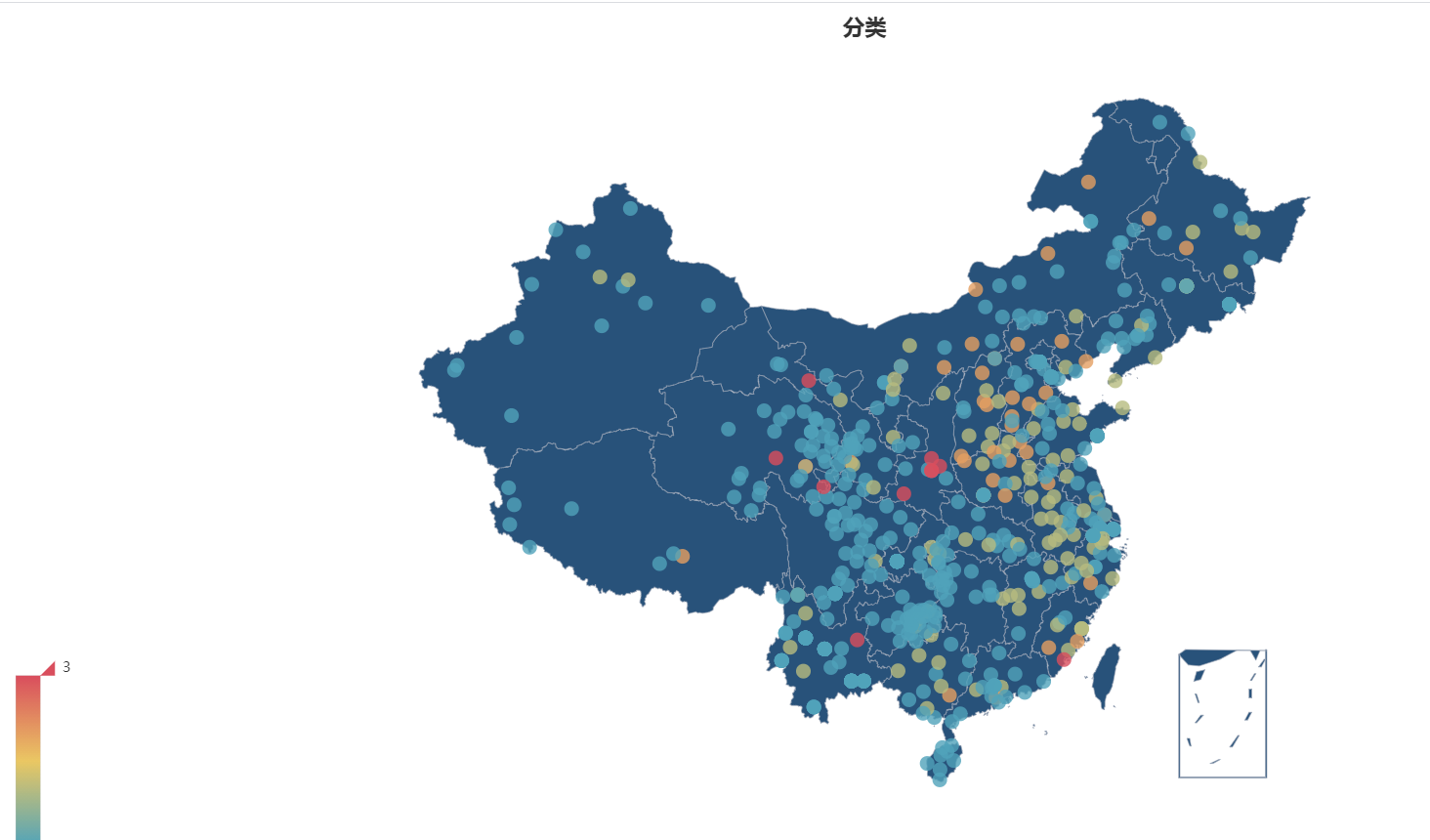
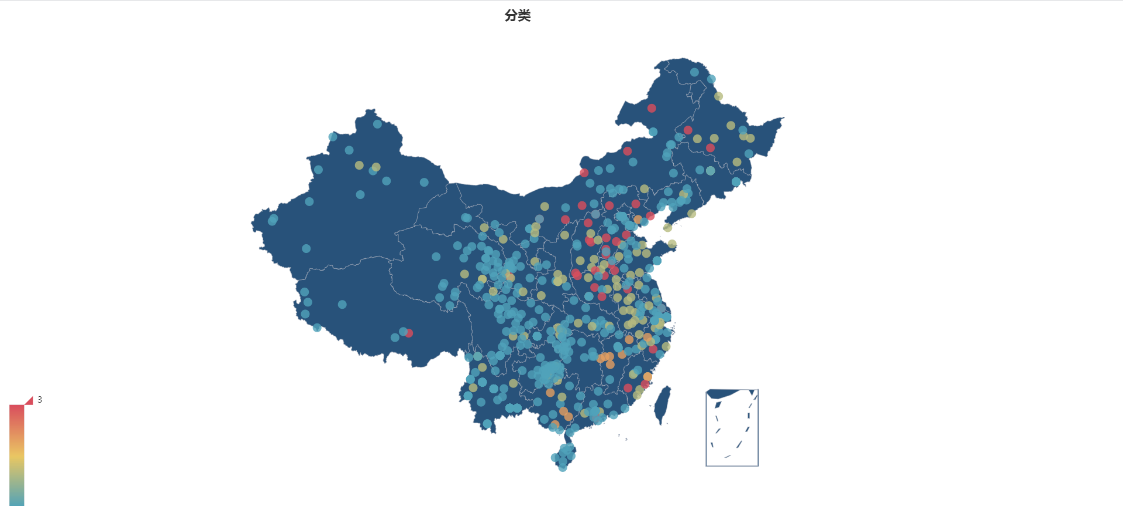
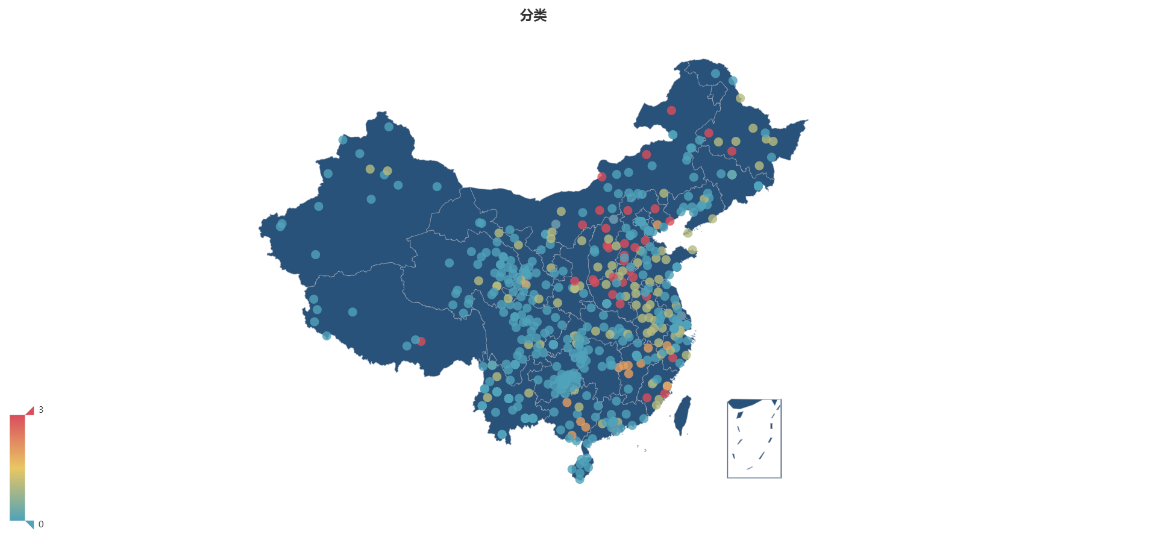


图3：k=4城市分类可视化(1)

从图3可以看出，大多数城市被划分为第一类，这与我们的预期不符。我们考虑由于kmeans是随机选取质心点，运行结果可能偶然性较大，又运行了两次，结果如图4（kmeans分类\_4\_iter1\_time1.html）和图5（kmeans分类\_4\_iter1\_time2.html）。

 图4：k=4城市分类可视化(2) 图5：k=4城市分类可视化(3)

,图4和图5展现了几乎一样的分类效果，并且图4和图5的分类效果也与图3一致，这似乎排除了由于kmeans的随机选取质心导致的偶然性使得分类效果不理想这一可能性。为了彻底排除该猜想，我们运行10次kmeans算法（结果保存在kmeans\_4.xlsx中），计算出每个城市在10次kmeans中最多次数被划分的类，作为该城市的类，可视化结果如图6（kmeans分类\_4.html）。

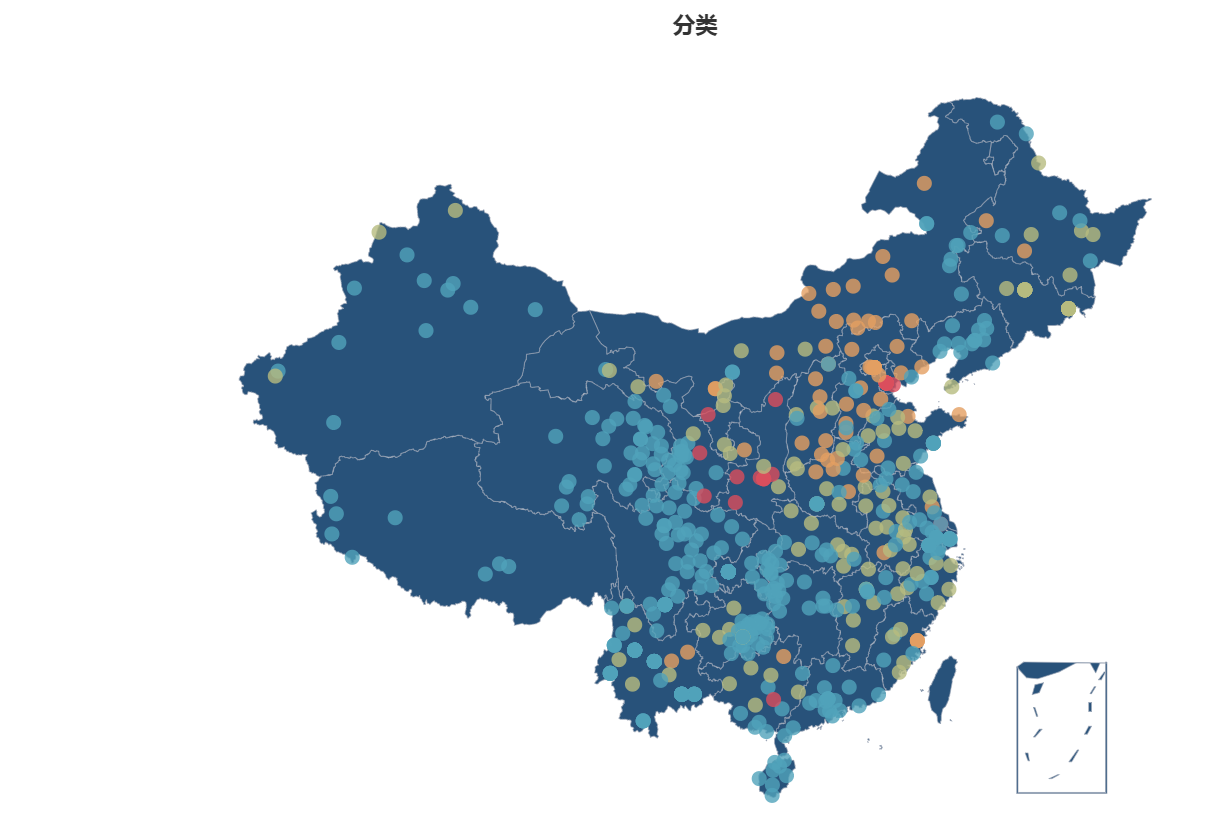
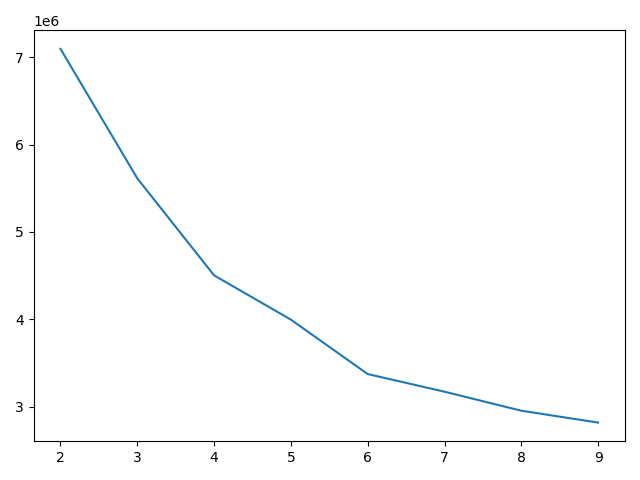
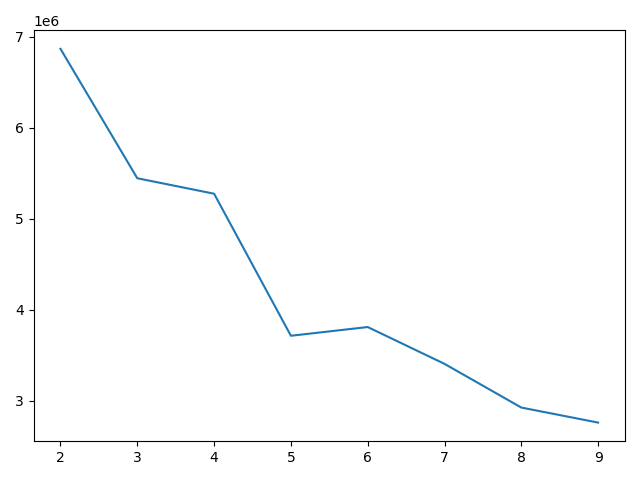
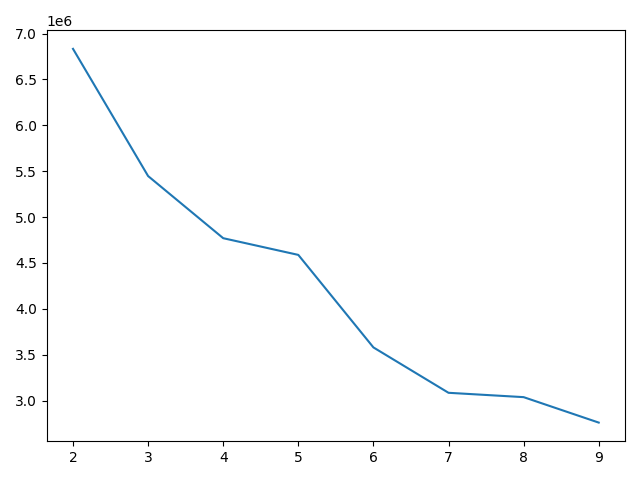


图6：k=4城市分类可视化（运行10次）

图6的分类与图3，图4，图5高度一致，因此，我们的猜想被彻底排除，kmeans的随机选取质心导致的偶然性并不是分类效果不理想的原因。

* 1. 改进

仔细分析各城市的气象灾害数据之后，我们认为导致分类效果不理想的原因可能是选取的聚类中心数量k不恰当。基于此猜想，我们画出不同聚类中心数量k对应的簇内误差平方和 (SSE)，寻找出现肘型图线的时间。为减小误差，我们运行了三次在不同聚类中心数量k下的簇内误差平方和，结果如图7所示。



(1) 第一次运行 (2) 第二次运行 (3) 第三次运行

图7：不同聚类中心数量k下的簇内误差平方和 (SSE)

根据图7显示，大概在k=5, 6, 7左右出现肘型图线。因此，我们分别可视化k=5, 6, 7的运行结果（分别保存在kmeans\_5.xlsx, kmeans\_6.xlsx, kmeans\_7.xlsx中）。

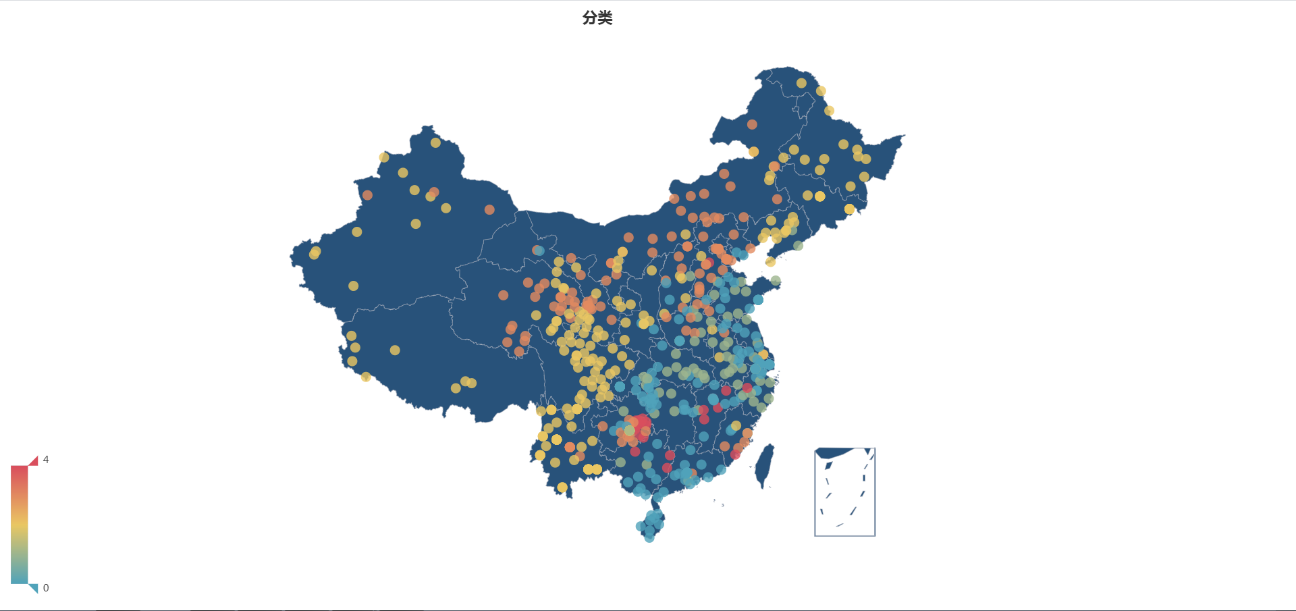


图8：k=5分类可视化结果

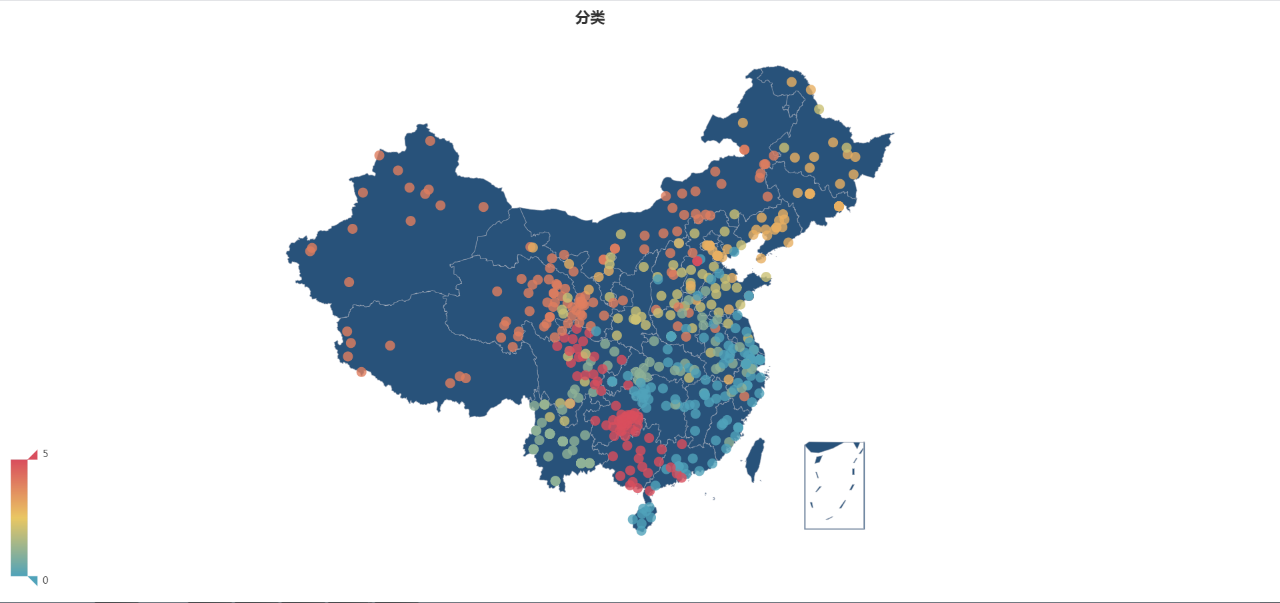


图9：k=6分类可视化结果

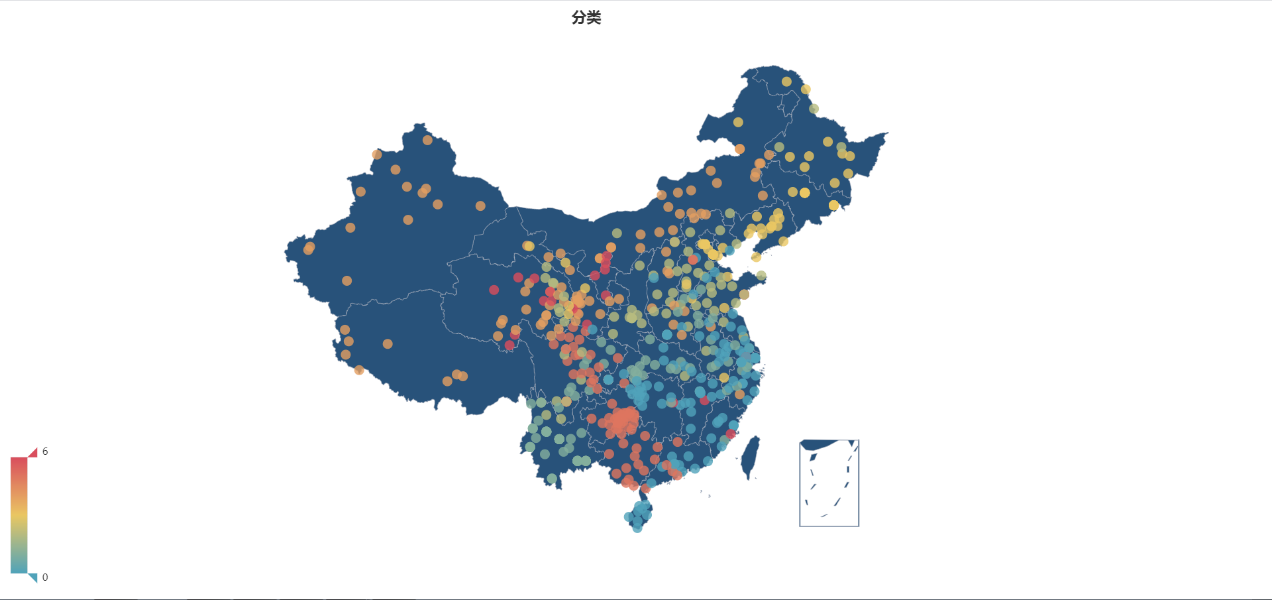


图10：k=7分类可视化结果

当k=5时，分类较为清晰，但第一类城市和第二类城市区别相比于k=6不明显。并且，由图8可以看出，k=5时很多北方城市被划分为了第一类城市（指图例中数值为0的标签）。另外，k=6相对于k=5时新增的分类集中在广西、贵阳一带，这具有其合理性。因此k=5的分类效果不如k=6的分类效果。当k=7时，分类效果和k=6时分类效果相差无几，新增的分类所含城市数量较少，分布在青海一带，其它地区偶有分布，因此，该新增的分类代表的气象特征不显著，新增该分类意义不大。综上所述，选取聚类中心数量k=6是该数据下应用kmeans算法的最优解。

1. 结论

该报告基于mongodb数据库中的半年内多地气象灾害类型数据，统计半年内各地区各灾害类型的天数，运用kmeans模型对城市进行分类。在实现kmeans算法中，我们发现将城市分成4类效果不佳。之后我们通过计算簇内误差平方和(SSE)推断kmeans分类数量k的最优解应该是5,6 或者 7。通过对比三种分类数量可视化图像，k=6时分类效果最好。