探索性数据分析

0. Github
1. 对数据集的基本分析
    1.1. 数据集的背景
    1.2. 数据集的结构
2. 数据清洗
    2.1. 时间序列连续性处理
    2.2. 异常值处理
3. 数据分析
    3.1. 时间序列相关性分析
        自相关性分析
        滞后值(lags)
        季节性与去季节化
        ACF
        PACF
        通过季节性分解、ACF、PACF观察流量时间序列的周期性规律
        自相关性显著性检验
    3.2. 空间聚类分析
        K-means算法执行流程
        评价一个K-means聚类的指标：肘部法则与肘部图
        K-means聚类结果与热力图对比
    3.3. 在空间聚类分析中加入时间维度的特征
        特征抽样
        特征压缩
    3.4. 通过特征压缩的方式加入业务种类维度

0. Github

branch:lab1

```
https://github.com/Liyanhao1209/BDL.git
```

1. 对数据集的基本分析

老师在群中提供的数据集：`all_data_ct_10M.h5`

1.1. 数据集的背景

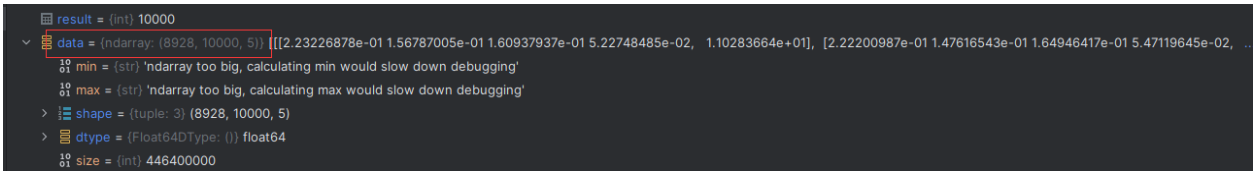
来自米兰城区的流量监控历史数据。米兰根据城市内流量基站的位置与覆盖范围，将整个城区划分为了100*100=10000个小的区域。每10分钟检测一次区域内产生的流量，其中流量分为五种不同的业务（如电话、短信、上网和其他），如此累计了共8928条数据。

1.2. 数据集的结构

通过python代码，我读入了这个数据集。首先我先看了一下这个hdf5文件里包括了哪些dataSets，最终发现有2个，分别为：

1. data
2. idx

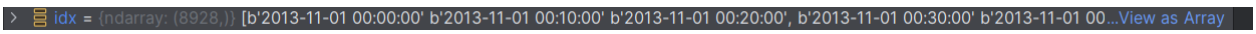
先来看data，通过调试信息，发现data是一个8928*10000*5的ndarray。



结合这个数据集的背景，我们不难发现：

data[i][j][k]的含义是：第j个城区的第k个业务在第i个10分钟内的流量数据。其中0≤i≤8927，0≤j≤9999，0≤k≤4。

而idx的形式更加简单，就是一个8928*1的ndarray，查看其中数据，发现是每个时间段的id。



2. 数据清洗

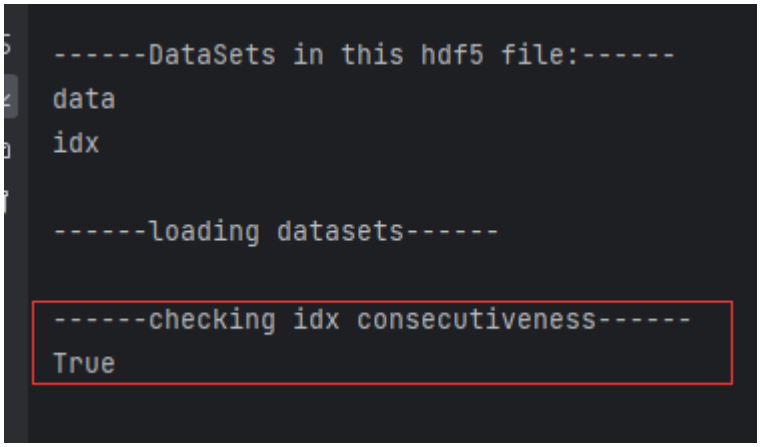
2.1. 时间序列连续性处理

对于时间序列来说，最重要的就是连续性。也即相邻两个数据点之间的时间间隔是一样的。如果发现有缺失的时间点，我们需要进行插值补全；如果有多余的，我们应该取均值替代重复的点。

因此这里我写了个脚本用来判断idx是否合法。

```
def are_time_intervals_equal(idx):
    # 将字节串数组转换为datetime对象数组
    timestamps = [dt.strptime(i.decode('utf-8'), '%Y-%m-%d %H:%M

    # 检查数组长度是否大于1，因为至少需要两个时间戳才能比较间隔
    if len(timestamps) < 2:
        return False

        # 计算第一个时间间隔
    first_interval = (timestamps[1] - timestamps[0]).total_secon

    # 检查剩余时间间隔是否都等于第一个时间间隔
    for i in range(1, len(timestamps) - 1):
        current_interval = (timestamps[i + 1] - timestamps[i]).t
        if current_interval != first_interval:
            logging.info("索引"+str(i)+"处时间间隔不同")
            return False

            # 所有时间间隔都相等
    return True
```

最终控制台输出是合法的。这样我们就不用对原始数据集进行插值或去重的处理了。

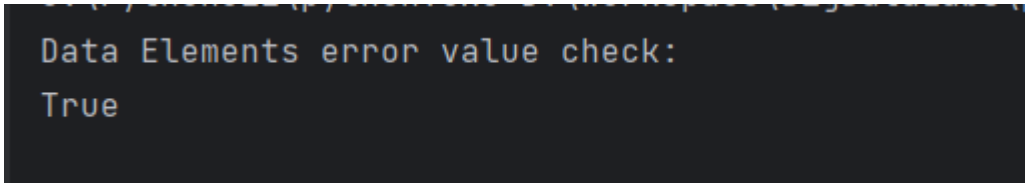


2.2. 异常值处理

除了时间序列需要连续，每个数据点的取值也需要符合实际意义。比如流量应该是一个非负的值。因此我们需要对所有数据单元进行检查，看看数据点是否有<0的异常值。

```
import sys

import numpy as np

from h5Reader.readHdf5 import readH5

if __name__ == '__main__':
    args = sys.argv
    h5_path = args[1]
    data, idx = readH5(args[1], ['data', 'idx'])

    print("Data Elements error value check:")
    print(np.all(data >= 0))
```

输出如下：



实际上异常值不仅仅包含负值，也包括了那些与历史或未来数据趋势大不相符的数据，比如给你一个流量的时间序列：

```
{ 1 1 1 1 9223372036854775808 1 1 1}
```

假设时间间隔为1s，这有可能是0x8000000000000000被当成unsinged long解释了，原本0的值被当成了2的63次方。

但问题是我们的流量数据是10分钟一次的，所以很难给这种突发的异常值定一个benchmark，进行处理。所以这部分我先跳过了。

3. 数据分析

3.1. 时间序列相关性分析

自相关性分析

自相关性函数（ACF，Autocorrelation Function）是时间序列分析中的一个重要概念，用于衡量一个时间序列在不同时间点的值与其自身在其他时间点的值之间的相关性。

滞后值(lags)

在时间序列分析中，滞后通常用来描述**一个变量与其自身在过去某个时间点的值之间的相关性**。例如，如果我们有一个每天记录的股票价格的时间序列，那么今天的价格与一周前的价格之间的相关性就可以通过计算一阶滞后的相关系数来分析。这里的“一阶滞后”就是指时间序列中当前观测值与其前一个时间点（在这个例子中是一周前）的观测值之间的时间差。

这个概念在做时间序列预测中很有用，因为了解数据的滞后效应可以帮助我们更准确地**捕捉到数据的动态变化**，提高预测的准确性。

季节性与去季节化

时间序列一般可能会有周期性、季节性的变化。而8928个10分钟相当于62天。对于一个如此长的时间序列，我们首先要对其进行周期性（季节性）的数据分析与处理。随后才能利用自相关分析，探究**除了周期性之外的其他规律**。

pandas提供了一套完整的API，用来处理时间序列的季节性。首先我们要为其指定一个开始的时间，以及时间跨度，还有就是数据点的采样频率。随后生成采样时间点，对应于

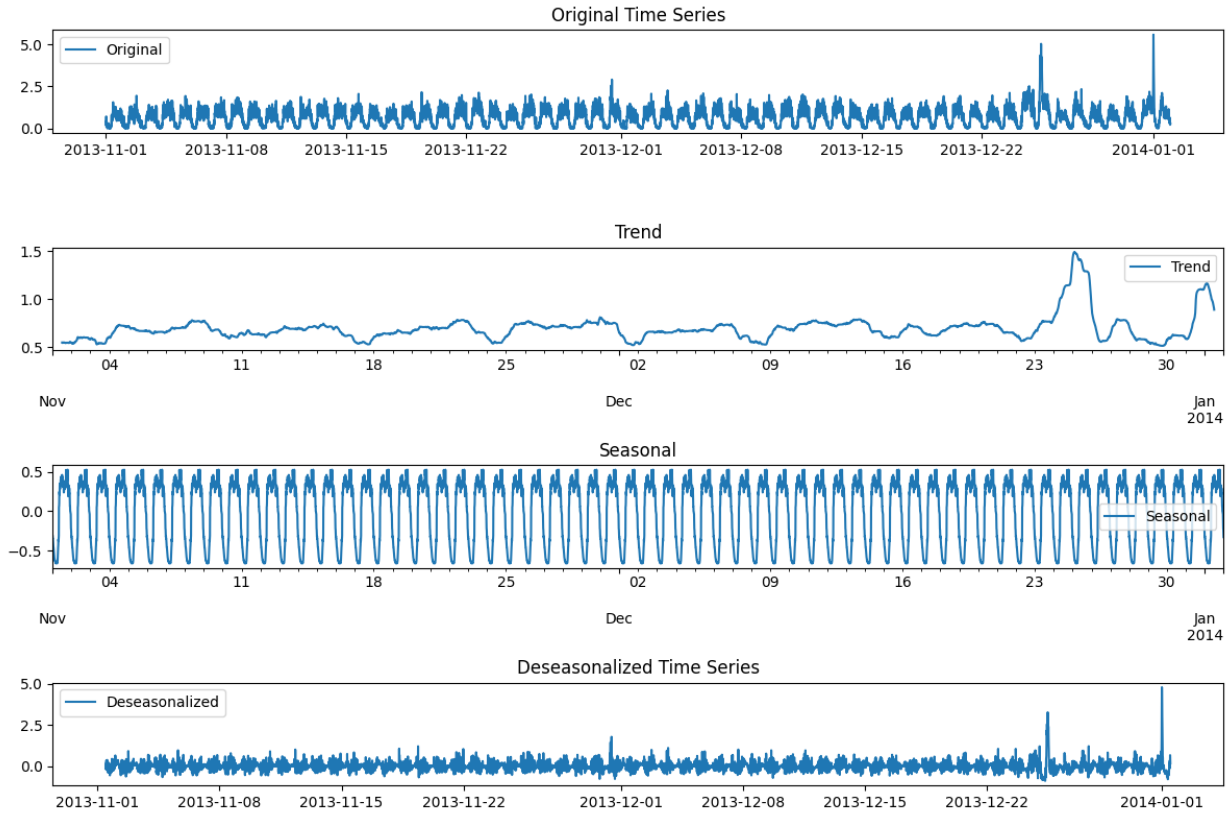
采样数据，进行季节性分解。

```
def remove_seasonality_and_trend(ts, start_date, freq='10min'):
    # 根据起始时间和频率生成日期范围
    dates = pd.date_range(start=start_date, periods=len(ts), fre

    # 将一维数组和日期转换为pandas Series
    ts_series = pd.Series(ts, index=dates)

    # 季节性分解
    result = seasonal_decompose(ts_series, model='additive', per
```

时间序列分解共计能得到三个结果：

1. Trend：趋势。趋势组件表示时间序列的长期趋势或发展方向。它是时间序列在较长时间尺度上的变化趋势，反映了数据的上升、下降或稳定状态。
2. Seasonal：季节性。季节性组件代表了时间序列在固定时间间隔内的规律性波动。
3. Residual：残差。残差组件，也称为不规则成分或噪声，是时间序列中无法通过趋势和季节性组件解释的部分。

举个例子，对于0号地区的第0号业务（短信接入），季节性分解如下图：



最上面那个是原始的时间序列，第二个是趋势，第三个是季节性，第四个是残差。因为残差部分是我们要探究的部分（除了周期性、季节性的部分，还有什么其他规律），所以我取名为Deseasonalized Time Series（去季节化时间序列）。

我们可以从图中看出一些规律：

1. 对于**长期趋势**：这个趋势总体来说是比较平稳的，而到了2013年12月底，突然暴增了一段，而2014年年初也有大幅增长。这是可以解释的，年关将近，做业务的要冲业绩；人们与亲人的联系也更加频繁（即便这是国外）。短信接发的频率上升也是可以理解的。
2. 对于**季节性趋势**：这个季节趋势也是十分稳定，我们可以看到04到11中共有7个数据点，这个意思其实是本月的4号到11号的意思。那么4号到5号之间出现了一个峰值，随后又开始降低，可以理解为：白天人们要工作学习，对于流量的使用需求大；而晚上要睡觉了，所以使用的流量少。
3. 对于**残差**：这个基本没有规律（或者说肉眼看不出规律）。留待自相关性分析。

ACF

自相关函数衡量的是时间序列中一个时间点的值与其之前或之后某个时间点的值之间的相关性。具体来说：

1. 定义：对于时间序列Xt，自相关函数ρ(k)在滞后k时的值定义为在滞后0和滞后k时的协方差与时间序列方差的比值。举个例子，我给个长度为10的等差数列，首项为1，公差为1。然后算一下ACF函数的值，k=3。也就是4-10这个序列和1到7这个序列的相关性（后者滞后了3个单位）：
   a. 计算公式：ACF(3) = **Σ[(x_t - μ)(x_(t-3) - μ)] / Σ[(x_t - μ)^2]**
   b. 首先计算均值：μ = (1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10) / 10 = 5.5
   c. 随后计算分子：Σ(i=4:10) (i-5.5)(i-3-5.5) = 12,25
   d. 然后计算分母: ∑(i=4:10) (i-5.5)^2=43.75
   e. 相除得到：0.28

   也就是ACF(3) = 0.28（这个值只会出现在[-1,1]之间）
2. 应用：ACF 通常用于识别时间序列的周期性和季节性模式。例如，如果时间序列具有明显的季节性，那么在季节性周期的滞后处，ACF 会显示出较高的峰值。

PACF

偏自相关函数衡量的是在控制了时间序列中其他时间点的值之后，一个时间点的值与其之前某个时间点的值之间的相关性。

与ACF的区别是，ACF(k)绝对值比较大，意味着时间序列收到过去k个时间单位变化的影响；而PACF(k)绝对值比较大，意味着时间序列的自回归部分可能在k个单位后仍有较大影响。

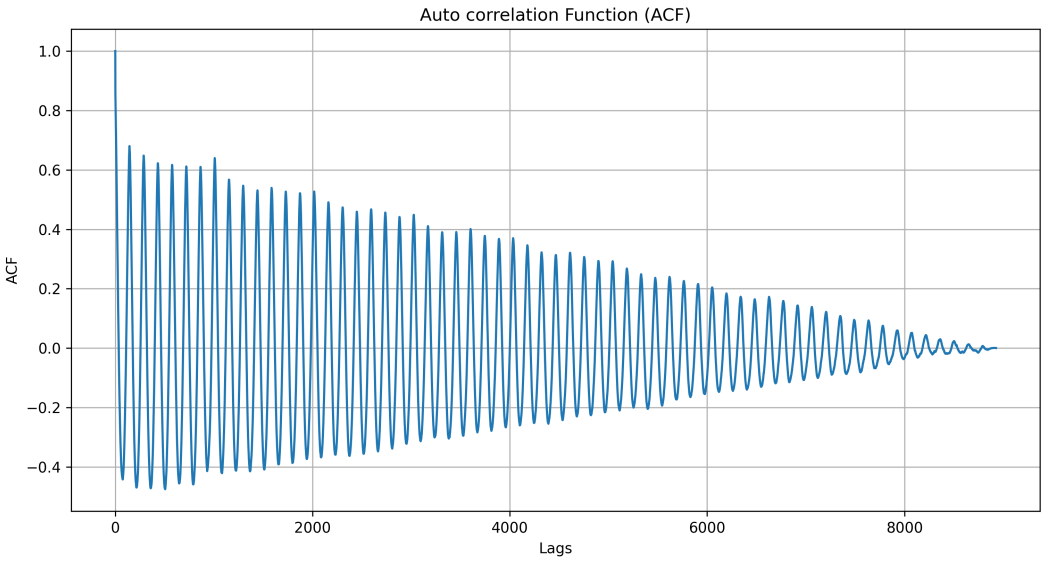
通过季节性分解、ACF、PACF观察流量时间序列的周期性规律

我打算通过两种模式的对比来查看这个周期性规律。

1. 不经过季节性分解的ACF与PACF函数
2. 经过季节性分解的ACF与PACF函数

对于**第一部分**来说，ACF最高阶数是8927阶，PACF最高阶数是8阶（后者涉及的计算更多，阶数大了我电脑跑不动）

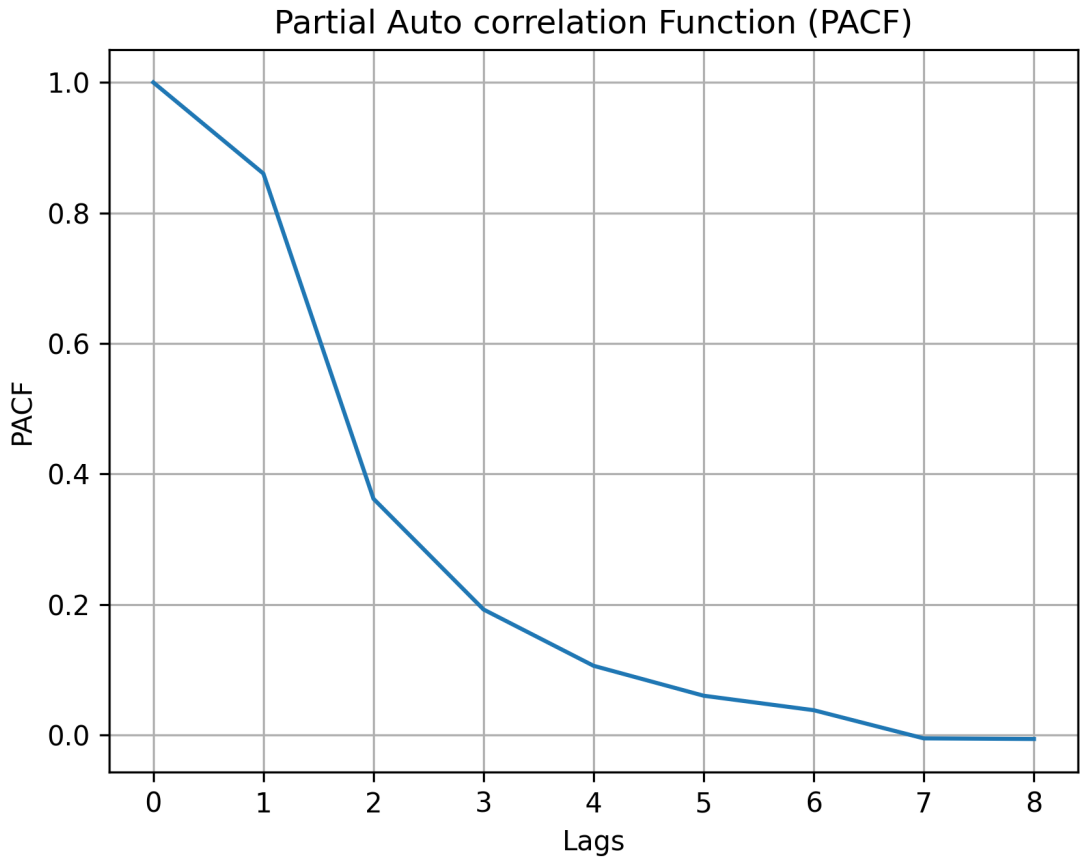
ACF图像：



可以看到的是ACF图像成震荡趋势，也就是说比方说ACF(10) = -ACF(20) = 0.6。那么意思就是前10个数据点与当前数据的相关性是正相关的，但前20个与当前数据的相关性是负相关的。而这样的震荡反复出现，就代表了原始数据是有周期性的。

我举个例子，可能是因为前10个10分钟，有7个和现在一样，都是白天，有3个是夜晚，那么对流量的需求比较大，所以是正相关的。但是前20个，可能就是13个夜晚，7个白天，而现在是白天，但前20个总体上来说，是夜晚，所以是负相关的。

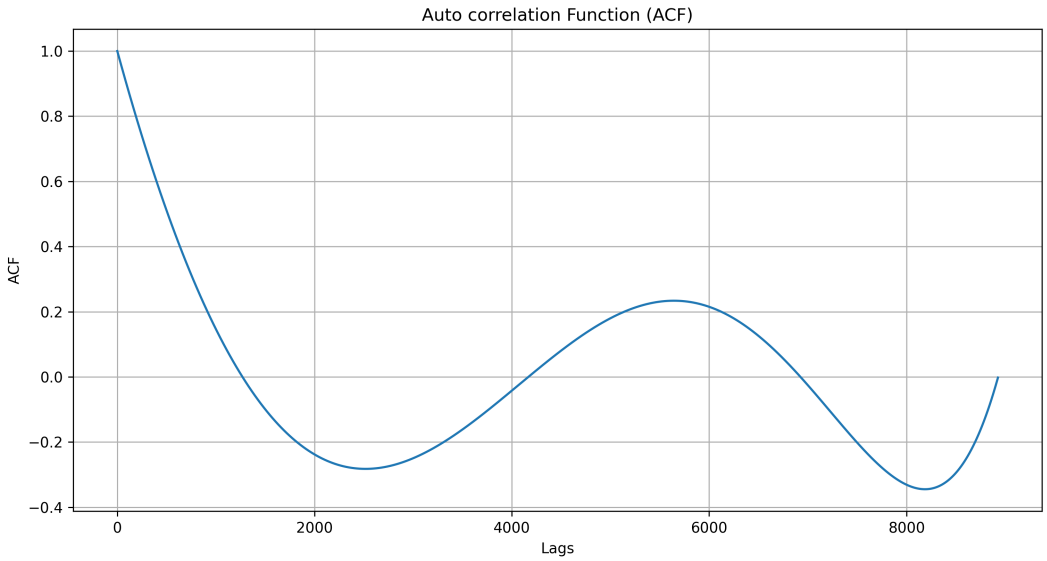
PACF图像：



而PACF图像随着阶数的增大，一路走低。从直观上解释，是因为ACF探求的是前k个与当前数据点的相关性。而PACF探求的是前面第k个和当前数据点的关系。因此前面第8个数据点和当前数据点不相关，是可以理解的。

对于第二部分：

ACF图像：

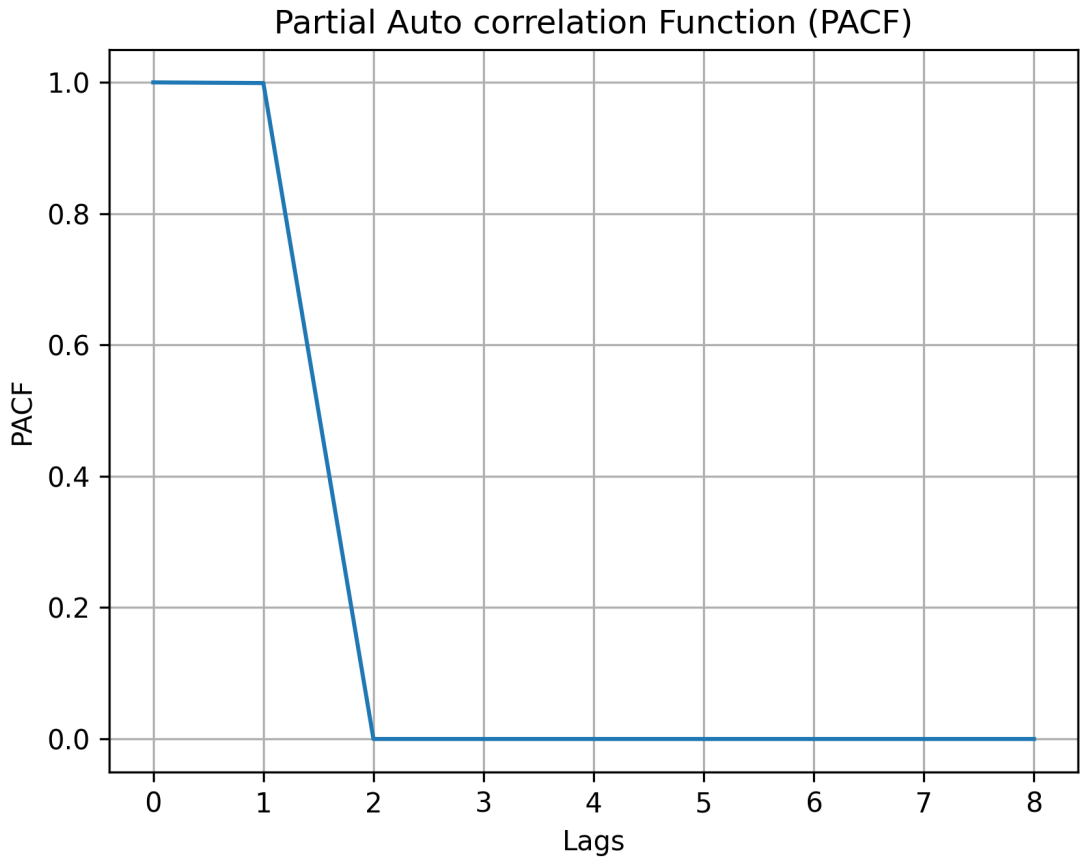


当我们去掉了季节性规律，可以看到这个时间序列中的数据呈现的规律是随机波动，出现三个峰值与两个谷值。

1. 当滞后值≤1500，基本上数据是呈现正相关的，不过这个正相关程度是随着滞后值的增大持续走低的。也就是说前k个点控制第k+1点的能力越来越低，我们可以认为这个时候时间序列除了周期性的因素，没有什么别的规律了。
2. 当滞后值为1500到2200左右。时间序列是呈负相关的。并且绝对值持续增大。也就是说前k个点控制第k+1个点的能力越来越强。说明1500-2200之间的节点k的取值，很大程度上受到了前0-k-1个节点的取值的影响。且这个影响是负相关的，也就是说前面的k个节点总体呈现的趋势和第k+1个节点的取值是反着来的。

总之，就是如果自相关系数向着0靠近，那么就代表这段时间的数据和之前的数据关系不大；不然代表相关性逐渐提升。

PACF图像：



不过PACF图像和去掉季节性因素之前的图像还是很相似的。

自相关性显著性检验

一般Ljung-Box Q可以做自相关性检验。我这里也是用这个统计指标了。

```
def ljung_box_q_test(time_series, lags=None):
    # 执行Ljung-Box Q检验
    q_stat, p_values = acorr_ljungbox(time_series, lags=lags)

    return q_stat, p_values
```

不过要注意的是，源码注释中说了：

```
def acorr_ljungbox(x, lags=None, boxpierce=False, model_df=0, period=None,
                   return_df=True, auto_lag=False):
    """
    Ljung-Box test of autocorrelation in residuals.
```

这个指标专门给残差用的，所以我们要先做季节性分解，才能看自相关显著性。

但是对于这块，我最终分析出来的数据很差。显示每一阶的自相关性都非常高，p值全部都是0.0。所以这一块我很疑惑，感觉可能自己做错了。但不知道怎么改正。

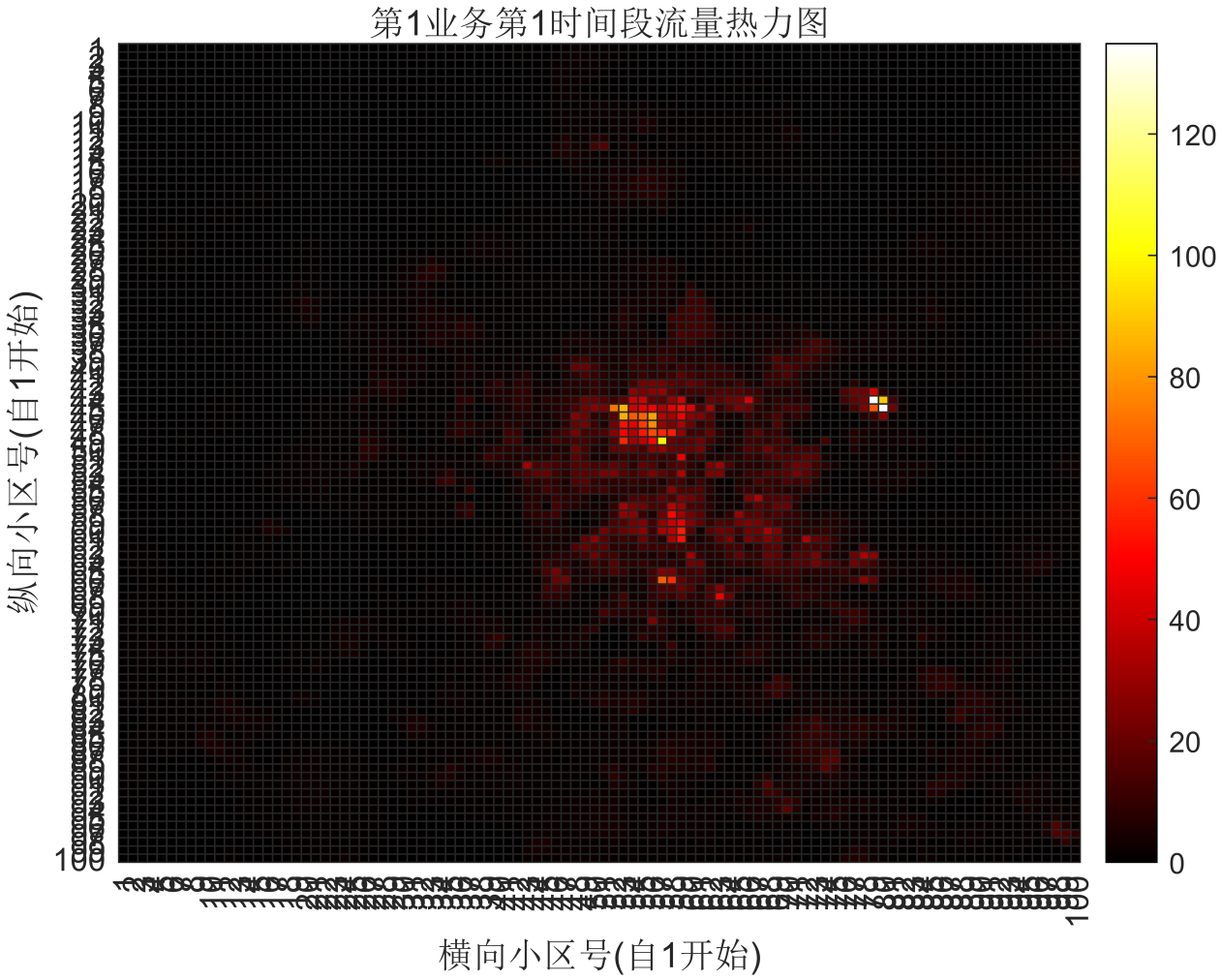
3.2. 空间聚类分析

首先我先尝试着画了一点城市的热点图，比如我选择了第一种业务和第一个十分钟，画出了城市的热点图：

```
function [] = region(business,time_id,sx,ex,sy,ey,data,target_d
    arr = data(business,1:10000,time_id);
    city = zeros(100,100);

    for i=1:100
        for j=1:100
            city(i,j) = arr( (i-1)*100+j);
        end
    end

    m = ex-sx+1;
    n = ey-sy+1;
    mat = zeros(m,n);

    for i=1:m
        for j=1:n
            mat(i,j) = city(i+sx-1,j+sy-1);
        end
    end
```

```
end
```

这里要注意的是，data的第二维度是个一维数组，所以如果想画二维热力图，要先转为二维数组。不过考虑到用户可能自己想要指定一个区域，所以我用sx,ex,sy,ey分别标注行列的开始结束索引。这样就可以自定义地挖去100*100的网格图中的任意区块了。



可以看到，城市的中心集中了大部分的流量。而城市的下方有着少量流量，上方可以说是几乎没有流量。

那么我就想根据流量大小，为城市区域划分流量等级。这就要涉及到聚类算法。这里我选择的是最基础的K-means聚类算法：

K-means算法执行流程

1. 指定需要划分的簇的个数K值（类的个数）；
2. 随机地选择K个数据对象作为初始的聚类中心（不一定要是我们的样本点）；
3. 计算其余的各个数据对象到这K个初始聚类中心的距离，把数据对象划归到距离它最近的那个中心所处在的簇类中;
4. 调整新类并且重新计算出新类的中心;
5. 循环步骤三和四，看中心是否收敛（不变），如果收敛或达到迭代次数则停止循环;

评价一个K-means聚类的指标：肘部法则与肘部图

肘部法则计算了一个指标：**WCSS（各个类的畸变程度之和）**

这个指标的定义是：每个聚簇中心于其内部成员位置距离的平方和。

假设一共将n个样本划分到K个类中（$K \leq n-1$,即至少有一类中有两个元素）

用C_k表示第k个类($k=1,2,\cdots,K$)，且该类重心的位置记为u_k

那么第k个类的畸变程度为：$\sum_{i \in C_k} |x_i - u_k|^2$

（这里的绝对值符号的意义表示的是距离，可视为一种广义的记号）

定义所有类的总畸变程度：$J = \sum_{k=1}^{K} \sum_{i \in C_k} |x_i - u_k|^2$

在部分多元统计教材中，J又被称为聚合系数。

那么WCSS是多少合适呢？

越小越好？如果越小越好，那么就多少个点就分成多少个簇就行了，相当于根本没有聚类。

越大越好？如果越大越好，那么就一个聚类，这样肯定最大，所以也相当于没有聚类。

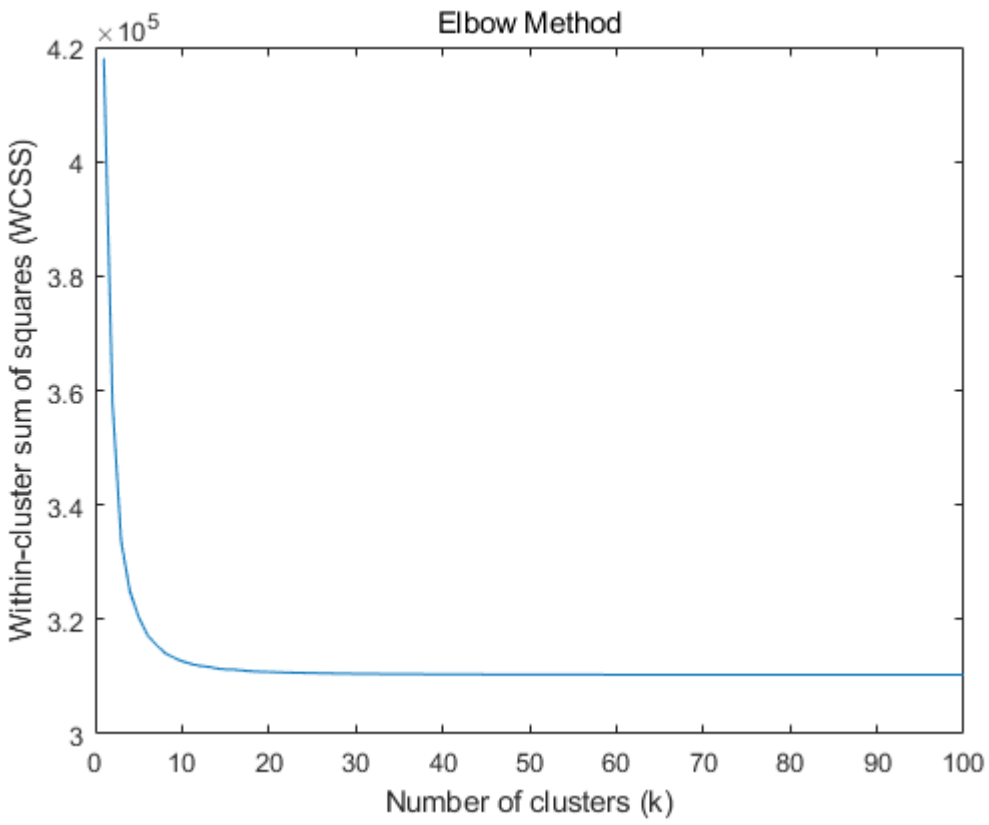
一般来说，当我们画出WCSS与聚类个数的关系后（成为**肘部图**），可以观察图像，在拐点处，一般就是比较合适的聚簇个数。

比如我对第一个业务的第一个时间点进行城市聚类分析，我可以先画出肘部图：

```
function [] = WCSS(region,k_max)
    % 将二维数组转换为一维数组
```

```
    region_1d = region(:);

    % 数据标准化
    region_std = (region_1d - mean(region_1d)) / std(region_1d);

    % 初始化 wcss 向量
    wcss = zeros(1, k_max);

    % 使用肘部法则确定聚类数目
    for k = 1:k_max
        [cluster_idx_1d, cluster_center] = kmeans(region_std, k,

        % 计算 WCSS
        for j = 1:k
            cluster_points = region_1d(cluster_idx_1d == j);
            cluster_center_j = cluster_center(j, :);
            within_cluster_sum = sum((cluster_points - cluster_c
            wcss(k) = wcss(k) + sum(within_cluster_sum);
        end
    end

    % 绘制肘部曲线图
    plot(1:k_max, wcss);
    xlabel('Number of clusters (k)');
    ylabel('Within-cluster sum of squares (WCSS)');
    title('Elbow Method');
    hold on;
end
```

这里就先设置一个最大的聚簇上限，一般根据经验来。这里我跑了几次，最终设置成了100。随后遍历每个k值，计算WCSS，然后画出肘部图。



可以看到，到了10-20中间，曲线的趋势彻底平缓，因此我最终选择了20作为聚类个数。

K-means聚类结果与热力图对比

随后我们就可以跑K-means聚类算法了。官网对K-meansAPI的介绍如下：

`idx = kmeans( X , k )` 执行 k 均值聚类，以将 n×p 数据矩阵 `X` 的观测值划分为 `k` 个聚类，并返回包含每个观测值的簇索引的 n×1 向量 (`idx`)。`X` 的行对应于点，列对应于变量。

默认情况下，`kmeans` 使用平方欧几里德距离度量，并用 k-means++ 算法进行簇中心初始化。

也就是说我们要传进去一个一维数组，因为我们的观测值只有1种，就是流量大小。所以最后我们给kmeans的X矩阵，就是10000*1的数组。

```
function [cluster_idx, cluster_center] = K_Means(region,k)
    % 将二维数组转换为一维数组
    region_1d = region(:);
```

```
    % 数据标准化
    region_std = (region_1d - mean(region_1d)) / std(region_1d);

    [cluster_idx_1d, cluster_center] = kmeans(region_std, k, 'Re

    % 将一维的聚类索引映射回原始的二维数组形式
    cluster_idx = reshape(cluster_idx_1d, size(region));
end
```

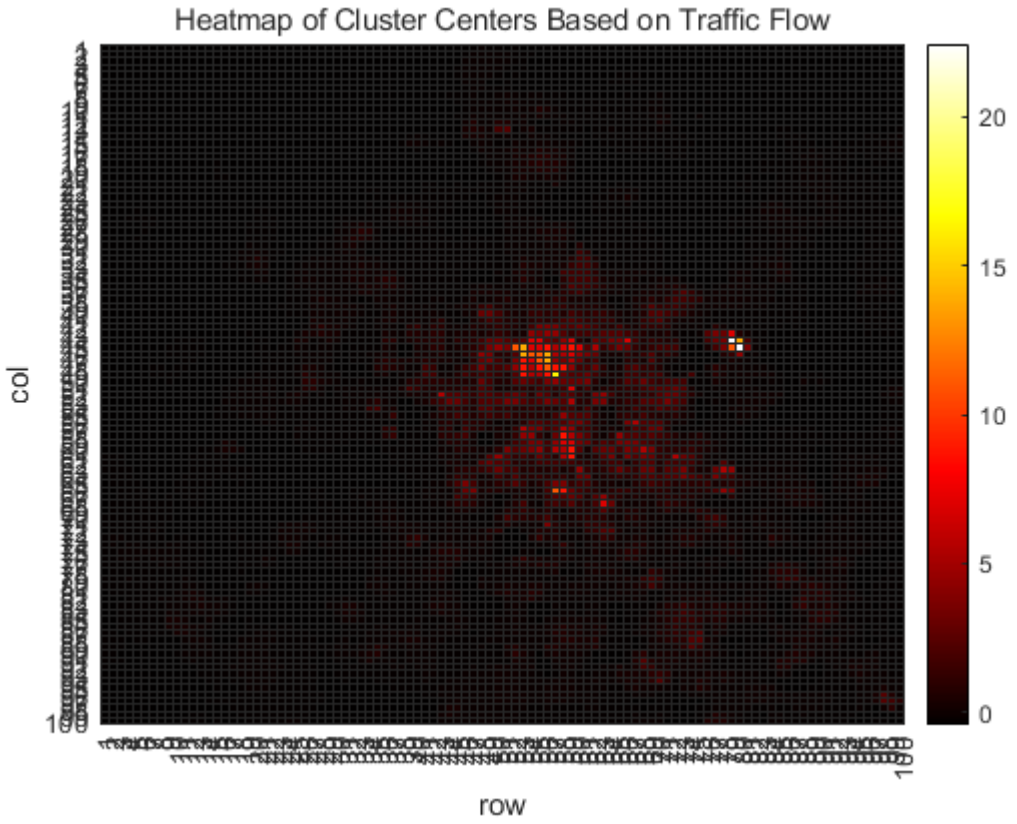
而K-means聚类的返回值包括两个，一个是对原数组的聚类标注，也即idx，另一个是聚类中心的特征，也即center。对于我们的这个问题，center代表了某个区域的中心的流量大小，比如有多个网格的标注都是1，代表1号聚类，那么center(1)就是这些网格中位于聚类重心上的网格的流量大小。

因此我们可以使用这两个返回值进行聚类热力图的绘制。首先我们把具有同样标注的网格的值全部改成这个标注对应的中心的流量的值，随后再进行聚类绘制：

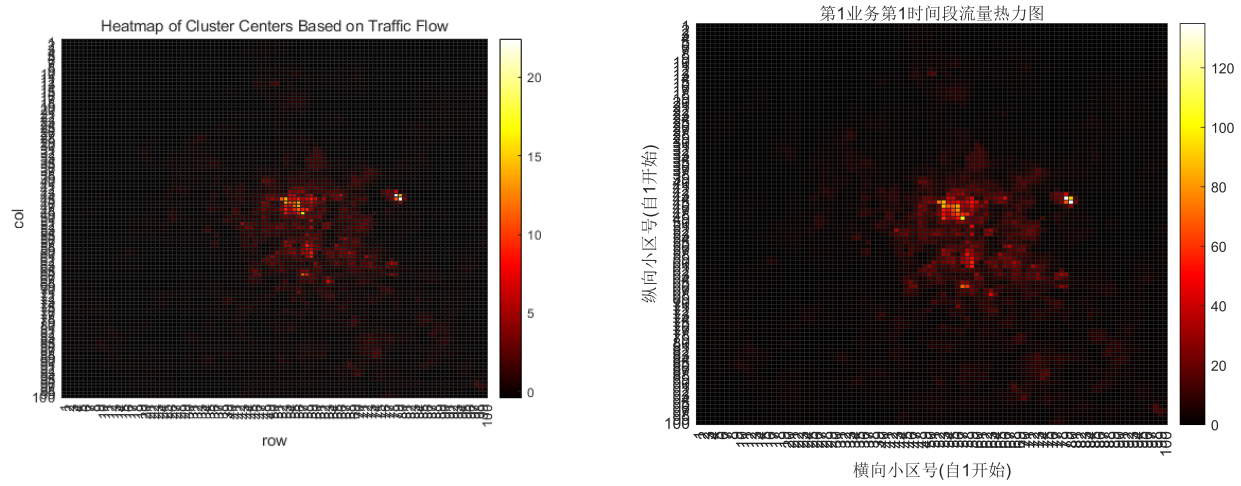
```
function [] = HeatMapForCluster(cluster_center,cluster_idx)

% 将 cluster_idx 转换为与聚类中心对应的流量大小
cluster_size = length(cluster_center);
cluster_value = zeros(size(cluster_idx));
[m,n] = size(cluster_idx);
for i = 1:m
    for j = 1:n
        cluster_value(i,j) = cluster_center(cluster_idx(i,j));
    end
end

% 使用 heatmap 函数绘制热图
heatmap(cluster_value);

colormap('hot')
colorbar;

grid on
```

```
% 设置标题和轴标签
title('Heatmap of Cluster Centers Based on Traffic Flow');
xlabel('row');
ylabel('col');
end
```

绘制结果如下：



我们可以对比一下城区热力图与聚类热力图：



几乎长得**完全一样**，这说明我们聚类的个数20选对了，划分出来跟原图都快一样了，聚类效果好。

3.3. 在空间聚类分析中加入时间维度的特征

这个数据集有三个维度，我们一开始对时间维度进行了自相关分析；而后对空间聚类进行了分析，但如何对时间与空间进行一个综合的分析？换句话说，我们怎么对历史上的所有小区进行聚类分析，而不是对某个时间点上的小区做聚类分析？

我们知道K-means聚类是支持多个特征值的，因此我们可以构建一个10000*p的矩阵传给K-means聚类。其中p是特征的数量，比方说最简单粗暴的一种，就是将某个地区的所有流量作为特征向量，也即8928*5=44640个特征。这样我们给K-means的数组就是10000*44640的size。

不过这样传进去的数组太大了，K-means要跑太久了。因此我们就需要对特征进行抽样和压缩。

特征抽样

既然8928*5太大了，我们就可以指定某项业务，随后选取该业务的8928项历史流量数据作为特征向量。不过这样撇开了不同种类业务的因素。这个后面再进行修补。

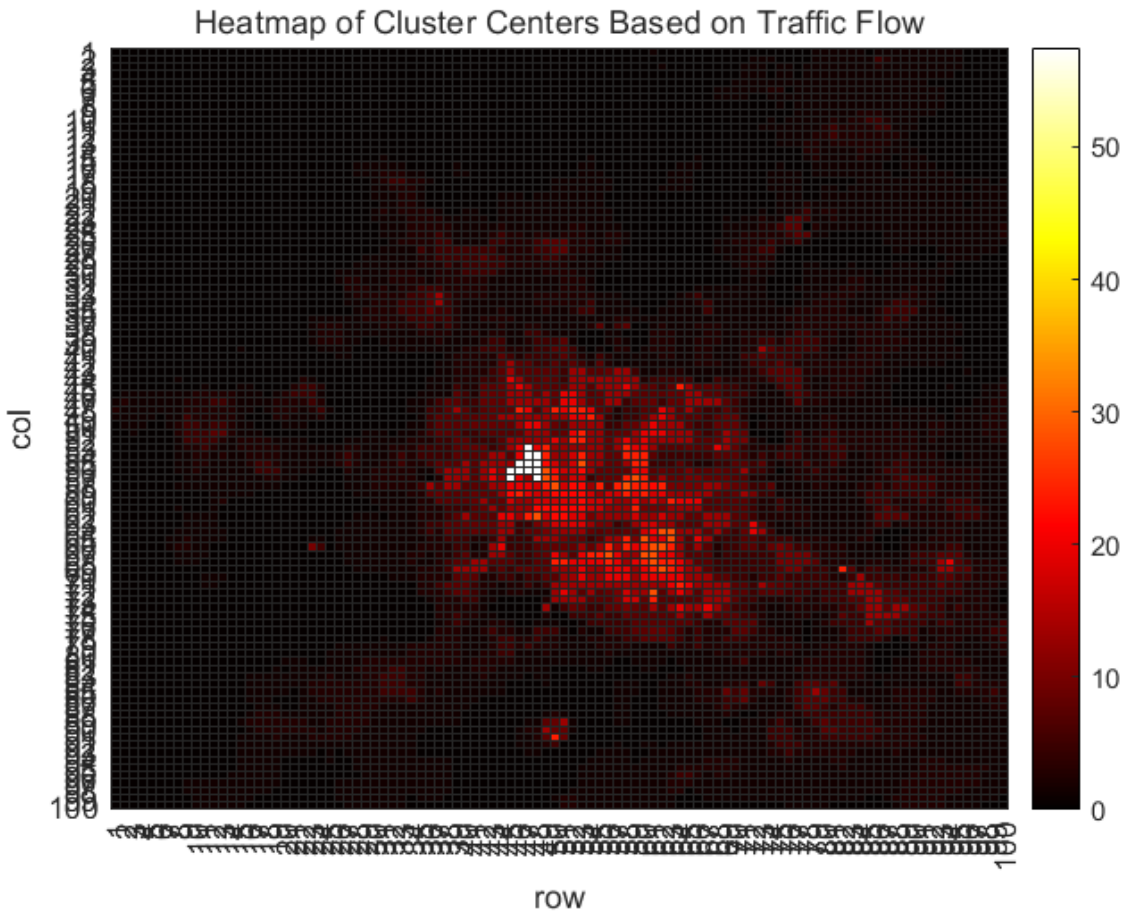
特征压缩

即便我们的特征向量是8928长度的，这样返回的聚类中心也要达到k*8928的大小，其中k是聚类个数。无法绘制热图。

因此我们要把多维特征向量压缩成一维的。这里我选择了平均值的压缩方式：

```
function compressed_features = Feature_Compression(cluster_cente
    % cluster_centers: 聚类中心矩阵，大小为 [num_clusters, num_feat
    % compressed_features: 压缩后的特征向量，大小为 [num_clusters, 1

    % 计算每个聚类中心向量的均值
    num_clusters = size(cluster_centers, 1); % 获取聚类的数量
    compressed_features = mean(cluster_centers); % 计算所有特征的均

    % 因为我们想要一个列向量作为结果，所以需要转置
    compressed_features = compressed_features'; % 转置行向量得到列

    % 确保输出的向量有正确的行数（尽管这里的转置已经保证了）
    compressed_features = compressed_features(1:num_clusters); %
end
```

随后即可绘制热图：



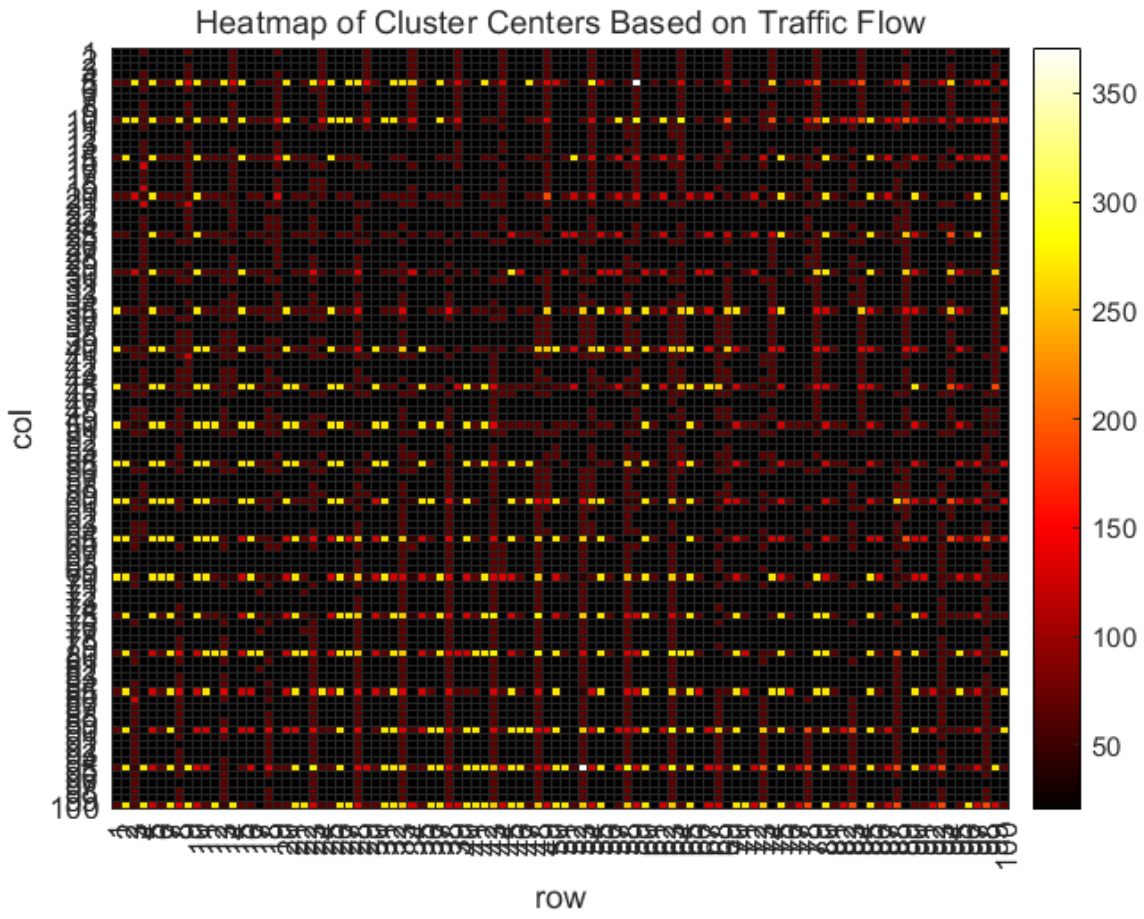
3.4. 通过特征压缩的方式加入业务种类维度

由于每两个时间点的间隔是10分钟，因此需要把data压缩为时间点为1天的三维数组。换句话说，一天有144个10分钟，因此最终压缩得到的data应该是(p*q*(r/144))大小的。压缩方式采用平均值的方式，也就是每144个数据点求平均。对于这个问题，最终的形式应该是5*10000*62的（共62天）

```
function [data_compressed] = time_seq_compression(data)
    % 假设 data 是一个 p x q x r 的三维数组
    % p 是业务数量，q 是地区数量，r 是时间点的数量（每个时间点代表10分钟）

    % 获取数组的尺寸
    [p, q, r] = size(data);

    % 计算每天的时间点数量（144个10分钟）
```

```
    time_per_day = 144;

    % 检查r是否是time_per_day的整数倍
    if mod(r, time_per_day) ~= 0
        error('时间点的数量不是每天时间点数量（144）的整数倍。');
    end

    % 初始化一个用于存储压缩后数据的三维数组
    data_compressed = zeros(p, q, r/time_per_day);

    % 遍历每一天的数据，计算平均值
    for day = 1:time_per_day:r
        % 获取当前天的时间点索引范围
        idx = (day-1)+1:(day+time_per_day-1);
        % 提取当前天的数据（对于所有业务和地区）
        data_day = squeeze(data(:,:,idx)); % 去除可能的单一维度
        % 计算当前天的平均值（沿着时间维度）
        data_day_mean = mean(data_day, 3); % 3 是时间维度的索引
        % 将计算得到的平均值存储到压缩后的数组中
        data_compressed(:,:,ceil(day/time_per_day)) = data_day_m
    end
end
```

随后我们再对这个压缩后的三维数组进行多观测值的K-means聚类，绘制热图：



不过这样做跑出来的结果不太好，聚类都太分散了。感觉这样的尝试是有问题的。