# 基于 Topsis、LSTM、STL、Dijkstra 算法、GBDT 的快递需求分析

王铎磊, 袁文璐, 赵　钊

## 目录

# 1 问题分析

## 1.1 问题 1

1. 先对数据进行数据数学化，具体数学化方式见符号说明与模型建议求解
2. 然后统计每个城市的收获量、发货量、快递数量变化率以及相关性方面选出 8 组数据作为对一座城市进行评估的参考指标
3. 将数据进行正向化、标准化，并采用熵权法评估出不同数据的权重
4. 使用 TOPSIS 分析法对每一个城市进行评分，然后进行排序得到重要程度

## 1.2 问题 2

根据下述的模型假设，将问题转化为单变量的预测问题，并采用时间序列分析的预测方法，预测所有可通物流的城市之间 2019 年 4 月 18 和 19 日的物流量。

## 1.3 问题 3

见下文模型建立与求解

## 1.4 问题 4

见下文模型建立与求解

## 1.5 问题 5

对于问题五，根据题目背景，按照季度因素给出计算固定需求和非固定需求的办法并进行实际应用，这里首先进行了数据的预处理，删去了为 0 的数据并对季度进行了标注，之后，对于每个季度每条运输线路进行了分类，汇总，计算，其均值，标准差等表征值，在不断计算调整的情况下，最终将（均值-标准差）/2 作为了固定需求量的表征值。基于 GBDT 模型，采用固定需求量作为自变量，季节作为分类应变量进行了训练，最终得出训练集精度为百分之九十以上，说明固定需求常数选取准确合理。采用当日实际运货量减固定需求常数作为非固定需求量，进行两类分类汇总统计，对于题目要求的非固定需求均值及标准差及其总和进行了计算。

# 2 模型假设

1. 针对问题 1，假设只有选出的 8 指标对城市重要性的影响显著，剩余的指标或与选出的 8 个指标关联性很强因此不必重复利用，或影响很小。因此假设城市重要性只取决于选出的 8 个指标。

2. 问题 2 中，假设两座城市之间的运输量随时间变化的趋势，受到这两座城市和其他城市之间的关联的影响非常小，因此只针对这两座城市间的货物运输量随时间变化做分析和预测，将问题简化为单变量预测。

# 3 符号说明

| 序号 | 符号 | 符号说明 |
|---|---|---|
| 1 | $City$ | 城市的指标集，即 $City = \{A, B, ..., Y\}$ |
| 2 | $x_{ij-t}$ | 第 $t$ 天从城市 $i$ 到城市 $j$ 的运输量，其中 $i, j \in City$ |
| 3 | $Rmax_j$ | 城市 $j$ 的最大收货量 |
| 4 | $Rmin_i$ | 城市 $i$ 的最小收货量 |
| 5 | $Smax_i$ | 城市 $i$ 的最大发货量 |
| 6 | $Smin_j$ | 城市 $j$ 的最小发货量 |
| 7 | $DRmax_j$ | 城市 $j$ 的最大收货变化量 |
| 8 | $DSmax_i$ | 城市 $i$ 的最大发货变化量 |
| 9 | $NRmax_j$ | 某一天给城市 $j$ 的发货的城市数量的最大值 |
| 10 | $NSmax_i$ | 某一天城市 $i$ 对外发货的城市数量的最大值 |

表 1: 问题 1 符号说明

# 4 模型的建立与求解

## 4.1 问题 1

### 4.1.1 考量指标的选择与数学化

参考表格问题 1 符号说明中的序号 3 到 10，我们选择这 8 个指标对一个城市的重要性进行评估。首先根据这 8 个指标的意义，可以写出对应的计算公式，从而对数据进

行处理和计算：

$$Rmax_j = \max_i \sum_i x_{ij-t}$$

$$Rmin_j = \min_i \sum_i x_{ij-t}$$

$$Smax_i = \max_i \sum_j x_{ij-t}$$

$$Smin_i = \min_i \sum_j x_{ij-t}$$

$$DRmax_j = \max_i \left| \sum_i x_{ij-t+1} - \sum_i x_{ij-t} \right|$$

$$DSmax_i = \max_i \left| \sum_j x_{ij-t+1} - \sum_j x_{ij-t} \right|$$

定义

$$sgn(x_{ij-t}) = \begin{cases} 0 & x_{ij-t} = 0 \\ 1 & x_{ij-t} > 0 \end{cases}$$

则

$$NRmax_j = \max_i \sum_i sgn(x_{ij-t})$$

$$NSmax_i = \max_i \sum_j sgn(x_{ij-t})$$

统计整理后的数据见附录.

### 4.1.2 数据的正向化、标准化

先对数据采取正向化。

某一城市的最大收货量、最大发货量、最小收获量、最小发货量，这 4 个指标均是越大说明这个城市流通的物流越多，这个城市的重要性就越高。因此上述 4 个指标均为正向指标。

同样的，某一城市一天的发货城市数量和收获城市数量，也越多说明此城市物流越多，因此是正向指标。

而最大的城市收货量和发货量变化量越大，说明这座城市收发货的极差大，也就是该城市可能不稳定，容易有物流掉线的时刻。因此我们评估的时候，认为最大收货变化量与最大发货变化量为负向指标。

对参数进行正向化之后，将同一组数据进行伸缩使得 $||\cdot||_2$ 为 1，该过程为对数据进行标准化。正向化标准化后的数据见附录.

#### 4.1.3 运用 Topsis 分析法进行评估

使用熵权法计算 8 个指标的权重占比，计算结果如下表

| 参数 | $Rmax_j$ | $Rmin_i$ | $Smax_i$ | $Smin_j$ | $DRmax_j$ | $DSmax_i$ | $NRmax_j$ | $NRmax_i$ |
|---|---|---|---|---|---|---|---|---|
| 权重/% | 2.602 | 51.957 | 2.661 | 34.065 | 4.654 | 3.998 | 0.036 | 0.027 |

表 2: 熵权法计算权重比

根据上述计算得出的权重，进行优劣解距离分析，得到每个城市的综合评分（得分与全排名见附录），前 5 名为

| 排序 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 城市名称 | B | T | P | G | L |

表 3: 问题 1 结果

### 4.2 问题 2

#### 4.2.1 数据处理

将每天每两个城市之间的货物流通量进行整理，以便后续预测使用。

处理后的数据见附件 Porblem 2.xlsx

#### 4.2.2 时间序列分析

使用时间序列分析法（python 代码见附录）预测出 2019 年 4 月 18 和 19 日的物流量各个城市的物流量，全部预测数据见附录。其中问题询问的预测数据如下表:

| 日期 | “发货-收货”城市之间的快递运输数量 | | 总快递运输数量 |
|---|---|---|---|
| 2019 年 4 月 18 日 | M-U | 81 | 12117 |
| | Q-V | 53 | |
| | K-L | 100 | |
| | G-V | 573 | |
| 2019 年 4 月 19 日 | V-G | 463 | 12131 |
| | A-Q | 94 | |
| | D-A | 44 | |
| | L-K | 72 | |

表 4: 问题 2 结果

## 4.3 问题 3

### 4.3.1 数据预处理

考虑到不同地区之间的运输情况随着时间的变化是一个典型的时间序列，我们使用 (From, To) 的有序对作为 Key，可以对数据进行一个重新整理。对整理后的某个 Pair，我们可以绘制出它的运输情况随时间的变化图，以 N 城运输到 V 城的为例：

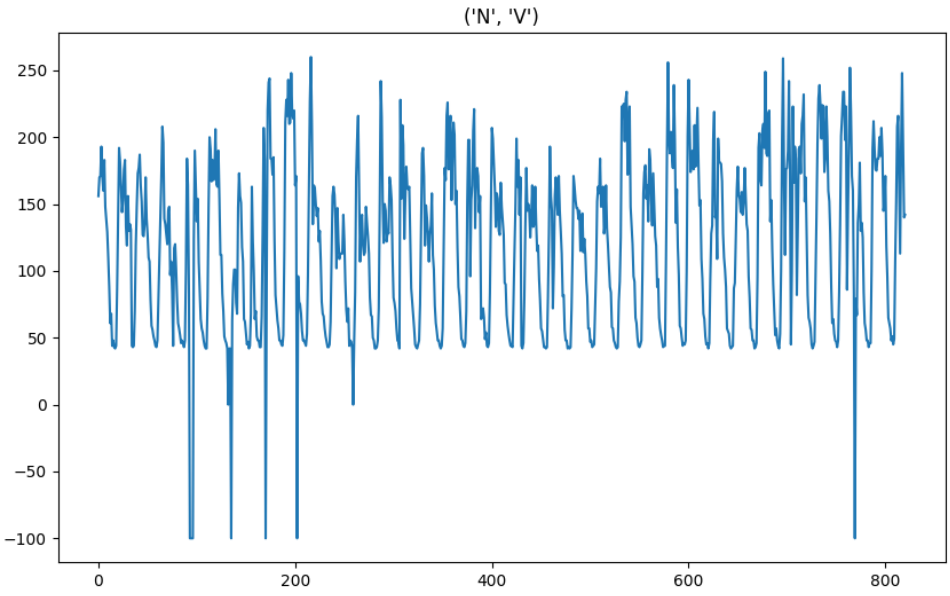


图 1: The tend from N to V

为了预测的方便，我们采取一个离群的负数作为无需求的异常点。预测时，如果出现负数的情况，可以认为是异常值。

#### 4.3.2 思路 1. STL 分解

考虑到两地之间的运输需求随着时间变化的趋势是典型的时间序列的数据。并且因为人类活动的周期性，物流需求受到了季节影响。因此，做出一个合理的推测：数据 = 季节性规律 + 趋势 + 噪音。

考虑到物流与购物消费有较大关系，我们认为它以月为基本单位进行相关的变化。使用 statsmodel 库中提供的 STL 方法，我们可以得到 N to V 的分解效果图：

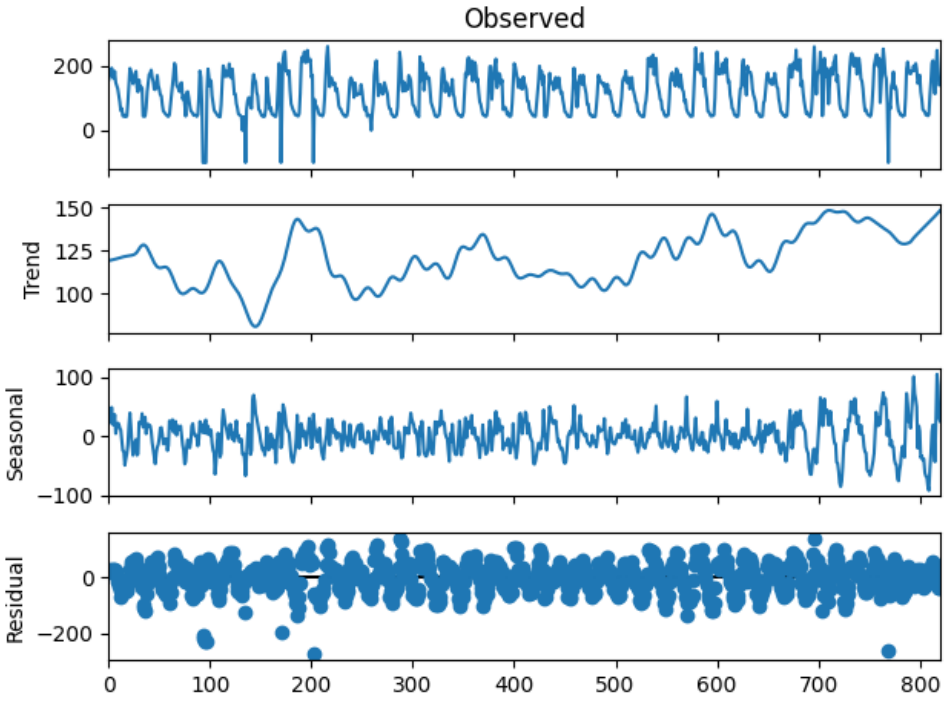


图 2: aa

可以注意到，其中季节性的规律在最近的一年来峰值差距很大，而处于 2020 年到 2022 年之间的波动相对较小，我们观察了其他数据，季节性的部分也具有类似特征。这里猜想可能与疫情原因造成的消费水平降低，以及疫情后的报复性消费有关。

由于问题中的时间序列具有特殊的周期性，我们考虑使用过往周期中相似时间段附近的数值进行相关预测，并用置信度作为其权重。考虑到距离当前时间比较久远的数据的时效性不强，故对其系数进行额外的折扣。

通过测试发现，该方法对于异常值的检测比较不敏感，如果尝试使用置信区间进行折扣，容易造成整体的预测偏小。于是，本结果对于预测是否能正常发货的价值比预测实际快递数量要更精准。后续改进中考虑针对噪声进行额外的学习，可以尝试使用 KL 散度等度量对其分布的突变点进行预测。

| 日期 | 城市对 | 是否能正常发货 | 快递运输数量 |
|---|---|---|---|
| 2023 年 4 月 28 日 | I-S | Y | 21.76 |
| | M-G | Y | 36.21 |
| | S-Q | N | -9.14 |
| | V-A | Y | 29.58 |
| | Y-L | N | 6.83 |
| 2023 年 4 月 29 日 | I-S | Y | 21.73 |
| | M-G | Y | 36.26 |
| | S-Q | N | -9.58 |
| | V-A | Y | 29.43 |
| | Y-L | N | 7.15 |

表 5: 问题 3 结果

#### 4.3.3 思路 2. LSTM

考虑循环神经网络 (RNN) 在处理时间序列数据的时候具有良好效果，RNN 在训练的过程中，会用上次训练得到的结果作为数据进行进行新一轮的学习。长短期记忆 (英语：Long Short-Term Memory，LSTM) 是一种时间循环神经网络 (RNN)。由于独特的设计结构，LSTM 适合于处理和预测时间序列中间隔和延迟非常长的重要事件。

LSTM 的表现通常比时间循环神经网络及隐马尔科夫模型（HMM）更好。LSTM 采用了特殊的门结构，对输入输出和遗忘功能进行了优秀的处理。但是由于训练成本和时间的缘故，这里没有再尝试使用 LSTM 进行具体的实验。

### 4.4 问题 4

#### 4.4.1 预处理

根据题目给出的图示，可以建立一个网络的基本框架。接下来，对于给定的一天，需要计算两点之间的实际运输量，而且每次两点之间的最短路径寻找都需要重新更新这个网络图。

#### 4.4.2 思路

首先，根据题目的信息构建一个城市的网络图。那么对于给定一个起点、终点，其单次运输货物的总量是常数，因此，针对这一次运输只需要寻求一个最优解即可。注意到，该次运输的道路权重会根据起点终点的不同而改变，因此要注意确定起点终点之后要更新网络的权重。接下来就是一个最短路径问题，可以使用 Dijkstra 等算法寻找一个

5 步以内的最优解即可。这里直接计算结果，如果超过 5 步，我们舍弃这个结果，认为是不可到达。

| 日期 | 最低成本 |
|---|---|
| 23 | 161.92 |
| 24 | 147.96 |
| 25 | 160.48 |
| 26 | 167.82 |
| 27 | 156.15 |

表 6: 问题 4 结果

## 4.5 问题 5

### 4.5.1 数据预处理

先删去无发货需求数据，增加一个季度因素。对数据做如下处理：新增季节列，对每个数据进行标注。接下来进行实际计算。

### 4.5.2 固定需求常数的计算

采用均值减去标准差作为固定需求常数。但此时计算的固定需求常数并不是大部分小于最小值的，因此，将该固定需求常数除以 2 作为最终的固定需求常数。

### 4.5.3 基于 GBDT 分类预测的精度计算

计算精度：用需求阐述反过来预测其属于哪个季度，如果训练时预测精度良好，那么说明常数选择较为准确。具体来说，GBDT 算法的步骤如下：

1. 初始化模型。首先，通过训练数据集生成第一棵决策树，得到初始的分类预测结果。

2. 针对预测结果的误差，构建下一棵决策树。GBDT 根据上一次预测结果和真实结果的差距（即残差），训练下一棵决策树，通过迭代不断提升模型的准确性。

3. 将每棵决策树的预测结果加权求和。对于每个样本处理，GBDT 将每棵决策树的预测结果加权求和，最终得到该样本的最终类别预测结果。

4. 重复步骤 2 和 3，建立更多的决策树。迭代训练直到达到预设迭代次数或满足收敛条件为止。

最终得出，训练集精度为百分之 99.7。

### 4.5.4 指定城市固定需求常数及其季度总和

按如上方法计算，筛选出指定城市。

### 4.5.5 非固定需求计算

如前文进行数据预处理。然后计算非固定需求数，进行分类汇总统计。接着计算均值和标准差总和即可。求解完毕后汇入表格。

可视化后发现，非固定需求分布基本均匀。因此，标准差可以作为非固定需求的表征。

### 4.5.6 最终结果

| 季度 | 2022 年第三季度 (7—9 月) | | 2023 年第一季度 (1—3 月) | |
|---|---|---|---|---|
| “发货-收货”站点城市对 | V-N | V-Q | J-I | O-G |
| 固定需求常数 | 17.49 | 22.68 | 62.09 | 33.36 |
| 非固定需求均值 | 90.66 | 80.78 | 154.57 | 158.91 |
| 非固定需求标准差 | 67.77 | 65.37 | 98.13 | 97.64 |
| 固定需求常数总和 | 2507.15 | | 2817.29 | |
| 非固定需求均值总和 | 7079.75 | | 7150.78 | |
| 非固定需求标准差总和 | 5099.38 | | 4869.62 | |

图 3: 问题 5 结果

# 5 模型检验与评价

## 5.1 问题 1 模型评价

本问题是一个典型的评价类问题，所采用的 Topsis 分析法的优缺点总结如下：

### 5.1.1 优点

1. 避免了数据的主观性，不需要目标函数，不用通过检验，而且能够很好的刻画多个影响指标的综合影响力度。

2. 对于数据分布及样本量、指标多少无严格限制，既适于小样本资料，也适于多评价单元、多指标的大系统, 较为灵活、方便。

### 5.1.2 缺点

1. 需要的每个指标的数据，对应的量化指标选取会有一定难度。

2. 不确定指标的选取个数为多少适宜，才能够去很好刻画指标的影响力度。

3. 必须有两个以上的研究对象才可以进行使用。

## 5.2 问题 2 优缺点评价

本问题采用了时间序列分析 LSTM 模型进行预测，具有趋势性、周期性、随机性、综合性等特点，是对 RNN 处理不了长期依赖问题所做出的改进模型，对于之前的历史数据只提取有用的，而过滤掉了噪声，这样从某种程度上就减少了模型的计算量。但是通过假设城市间的两两物流是独立的本身可能对数据的准确性造成一些影响。

## 5.3 问题 3 优缺点评价

本题主要是用 STL 分解的方法，认为数据具有一定的周期性。STL 分解本身可以较好的描述数据的趋势，但是不具有很强的预测功能。因此在预测的时候，为什么采用历史数据进行一定的加权计算值得商榷。此外，LSTM 训练的时候，数据集如果拆分为具体城市而言，数据量较少，可能还要进行额外的数据处理或者扩充工作，这可能会影响准确度。

另外，上述考虑仅仅考虑了时间因素，并没有把空间因素计算在内，如果考虑空间因素，可以引入马尔可夫链模型，尝试对邻接矩阵进行一定的处理。但是这个思路需要额外的数据处理工作，把单日的数据变化变换成矩阵形式的数据变化。

## 5.4 问题 4 优缺点评价

本问题在计算的时候，因为代码设计的缘故，出现了若干大大小小的 bug。本题的思路很简单，但贪心的策略比较有趣。因为题目提供的数据恰好是当日从某地发往某地的全部货运量，这为分别考虑两地之间的货运提供了方便。另外，题目简化了路线之间的复杂规划问题，因此使得问题可以直接变成求解一个加权图的最短路径。考虑题目给出的运输成本公式比较有趣，实际上是一个关于 fix 和 PCS 和实际货运量的公式。

$$cost = fix * (1 + (\frac{real}{PCS})^3)$$

在进行计算的时候，有考虑去手动推导货运量的临界值，以使得当货运量位于某个区间内的时候，最短路径恒定。但是精力有限，粗糙结果还请见谅。

## 5.5 问题 5 优缺点评价

基于 GBDT 分类预测的精度计算具有以下优缺点：

### 5.5.1 优点

1. 准确性高：GBDT 是一种被广泛认为精度高的分类器，因为它结合多个弱分类器，能够通过迭代提升模型的精度。
2. 自动特征选择：GBDT 在训练过程中，自动考虑不同属性对分类器的贡献，因此无需手动选择特征。
3. 适用于高维数据：GBDT 适合处理高维数据，可以捕捉多个不同特征之间的复杂关系。
4. 可处理混合特征：GBDT 可以同时处理离散特征和连续特征，这使得它适用于混合特征的数据。

### 5.5.2 缺点

1. 容易过拟合：当数据量较小或训练树的深度较大时，GBDT 容易出现过拟合的问题。
2. 不适用于需要实时预测的场景：GBDT 是一种有监督的学习算法，需要先进行模型训练，然后才能进行预测，因此不适用于需要实时预测的场景。
3. 对噪声敏感：GBDT 在训练过程中，可能会受到数据噪声的干扰，导致模型效果下降。

# 参考文献

无

# 6 附录

| $City$ | $Rmax_j$ | $Rmin_i$ | $Smax_i$ | $Smin_j$ | $DRmax_j$ | $DSmax_i$ | $NRmax_j$ | $NRmax_i$ |
|---|---|---|---|---|---|---|---|---|
| A | 191 | 0 | 268 | 0 | 233 | 157 | 4 | 2 |
| B | 455 | 42 | 340 | 45 | 169 | 339 | 1 | 1 |
| C | 59 | 0 | 50 | 0 | 130 | 86 | 2 | 4 |
| D | 164 | 0 | 92 | 0 | 196 | 123 | 4 | 4 |
| E | 148 | 0 | 267 | 0 | 128 | 170 | 2 | 1 |
| G | 842 | 0 | 860 | 0 | 1059 | 1336 | 9 | 7 |
| H | 362 | 0 | 447 | 0 | 501 | 422 | 3 | 3 |
| I | 375 | 0 | 367 | 0 | 346 | 407 | 3 | 3 |
| J | 447 | 0 | 375 | 0 | 611 | 668 | 4 | 4 |
| K | 345 | 0 | 362 | 0 | 413 | 472 | 3 | 3 |
| L | 822 | 0 | 842 | 0 | 1164 | 1090 | 10 | 10 |
| M | 293 | 0 | 221 | 0 | 382 | 362 | 4 | 5 |
| N | 243 | 0 | 293 | 0 | 309 | 272 | 5 | 3 |
| O | 340 | 0 | 386 | 0 | 493 | 736 | 6 | 3 |
| P | 136 | 0 | 164 | 42 | 112 | 92 | 1 | 1 |
| Q | 386 | 0 | 235 | 0 | 308 | 512 | 5 | 5 |
| R | 356 | 0 | 343 | 0 | 587 | 525 | 5 | 5 |
| S | 343 | 0 | 319 | 0 | 277 | 212 | 2 | 5 |
| T | 0 | 0 | 455 | 42 | 511 | 0 | 0 | 2 |
| U | 221 | 0 | 150 | 0 | 336 | 200 | 2 | 4 |
| V | 860 | 0 | 692 | 0 | 932 | 1052 | 6 | 6 |
| W | 550 | 0 | 558 | 0 | 576 | 643 | 3 | 3 |
| X | 455 | 0 | 576 | 0 | 963 | 1176 | 5 | 4 |
| Y | 576 | 0 | 442 | 0 | 364 | 521 | 2 | 3 |

表 7: 问题 1 原始数据整理

| $City$ | $Rmax_j$ | $Rmin_i$ | $Smax_i$ | $Smin_j$ | $DRmax_j$ | $DSmax_i$ | $NRmax_j$ | $NRmax_i$ |
|---|---|---|---|---|---|---|---|---|
| A | 6.108 | 0 | 6.971 | 0 | 3.869 | 5.021 | 12.79 | 3.198 |
| B | 14.55 | 1 | 9.274 | 0.6039 | 1.823 | 10.84 | 3.198 | 0 |
| C | 1.887 | 0 | 0 | 0 | 0.5756 | 2.750 | 6.396 | 9.593 |
| D | 5.244 | 0 | 1.343 | 0 | 2.686 | 3.933 | 12.79 | 9.593 |
| E | 4.733 | 0 | 6.939 | 0 | 0.5117 | 5.436 | 6.396 | 0 |
| G | 26.93 | 0 | 25.90 | 0 | 30.28 | 4.272 | 28.78 | 19.19 |
| H | 11.58 | 0 | 12.70 | 0 | 12.44 | 13.49 | 9.593 | 6.396 |
| I | 11.99 | 0 | 10.14 | 0 | 7.483 | 13.02 | 9.593 | 6.396 |
| J | 14.29 | 0 | 10.39 | 0 | 15.96 | 21.36 | 12.79 | 9.593 |
| K | 11.03 | 0 | 9.977 | 0 | 9.625 | 15.09 | 9.593 | 6.396 |
| L | 26.27 | 0 | 25.33 | 0 | 33.64 | 34.86 | 31.98 | 28.78 |
| M | 9.370 | 0 | 5.462 | 0 | 8.634 | 11.58 | 12.79 | 12.79 |
| N | 7.771 | 0 | 7.771 | 0 | 6.300 | 8.698 | 15.99 | 6.396 |
| O | 10.87 | 0 | 10.74 | 0 | 12.18 | 23.54 | 19.19 | 6.396 |
| P | 4.349 | 0 | 3.646 | 0.5636 | 0 | 2.942 | 3.198 | 0 |
| Q | 12.34 | 0 | 5.916 | 0 | 6.268 | 16.37 | 15.99 | 12.79 |
| R | 11.38 | 0 | 9.370 | 0 | 15.19 | 16.79 | 15.99 | 12.79 |
| S | 10.97 | 0 | 8.602 | 0 | 5.276 | 6.779 | 6.396 | 12.79 |
| T | 0 | 0 | 12.95 | 0.5636 | 12.76 | 0 | 0 | 3.197 |
| U | 7.067 | 0 | 3.198 | 0 | 7.163 | 6.396 | 6.396 | 9.593 |
| V | 27.50 | 0 | 20.53 | 0 | 26.22 | 33.64 | 19.19 | 15.99 |
| W | 17.59 | 0 | 16.24 | 0 | 14.84 | 20.56 | 9.593 | 6.396 |
| X | 14.55 | 0 | 16.82 | 0 | 27.21 | 37.61 | 15.99 | 9.593 |
| Y | 18.42 | 0 | 12.54 | 0 | 8.059 | 16.66 | 6.396 | 6.396 |

表 8: 问题 1 正向化标准化后数据

注: $Rmax_j$、$Smax_i$、$DRmax_j$、$DSmax_i$ 单位为 $10^{-4}$, $NRmax_j$、$NRmax_i$ 单位为 $10^{-6}$

| 索引 | 综合得分指数 | 排序 |
|---|---|---|
| A | 0.059803616 | 21 |
| B | 0.776879389 | 1 |
| C | 0.016388104 | 24 |
| D | 0.037513949 | 23 |
| E | 0.052350443 | 22 |
| G | 0.267481879 | 4 |
| H | 0.12593418 | 13 |
| I | 0.107242696 | 15 |
| J | 0.15095047 | 9 |
| K | 0.113583731 | 14 |
| L | 0.261836435 | 5 |
| M | 0.089994201 | 17 |
| N | 0.078847629 | 19 |
| O | 0.140351278 | 10 |
| P | 0.407963547 | 3 |
| Q | 0.1045013 | 16 |
| R | 0.132235778 | 12 |
| S | 0.085047073 | 18 |
| T | 0.417147627 | 2 |
| U | 0.06492305 | 20 |
| V | 0.238540899 | 6 |
| W | 0.165168187 | 8 |
| X | 0.222722121 | 7 |
| Y | 0.138568602 | 11 |

表 9: 问题 1 结果

| From | To | 2019.4.18 | 2019.4.19 | From | To | 2019.4.18 | 2019.4.19 |
|---|---|---|---|---|---|---|---|
| A | O | 114 | 95 | K | J | 46 | 50 |
| A | Q | 70 | 94 | K | L | 100 | 111 |
| B | G | 279 | 251 | L | D | 45 | 46 |
| C | M | 39 | 39 | L | G | 490 | 466 |
| C | N | 46 | 5 | L | H | 96 | 91 |
| C | U | 40 | 39 | L | J | 78 | 78 |
| C | V | 35 | 34 | L | K | 59 | 72 |
| D | A | 44 | 44 | L | O | 49 | 68 |
| D | E | 83 | 80 | L | P | 93 | 94 |
| D | L | 33 | 37 | L | R | 73 | 95 |
| D | R | 26 | 32 | L | W | 324 | 248 |
| E | I | 170 | 137 | L | X | 48 | 45 |
| G | L | 450 | 428 | M | C | 47 | 46 |
| G | N | 42 | 37 | M | G | 65 | 64 |
| G | O | 244 | 249 | M | N | 45 | 47 |
| G | Q | 57 | 58 | M | U | 81 | 138 |
| G | R | 59 | 61 | M | V | 109 | 96 |
| G | V | 573 | 639 | N | G | 43 | 44 |
| G | X | 35 | 33 | N | M | 173 | 171 |
| H | J | 92 | 169 | N | V | 132 | 161 |
| H | K | 54 | 50 | O | G | 209 | 190 |
| H | L | 49 | 48 | O | Q | 187 | 169 |
| I | E | 118 | 115 | O | R | 236 | 194 |
| I | J | 287 | 224 | P | D | 131 | 130 |
| I | S | 60 | 62 | Q | A | 61 | 71 |
| J | H | 43 | 43 | Q | M | 102 | 91 |
| J | I | 337 | 316 | Q | N | 21 | 31 |
| J | K | 73 | 217 | Q | O | 31 | 36 |
| J | L | 94 | 113 | Q | V | 53 | 64 |
| K | H | 91 | 86 | R | D | 40 | 39 |

表 10: 问题 2 结果

| From | To | 2019.4.18 | 2019.4.19 | From | To | 2019.4.18 | 2019.4.19 |
|---|---|---|---|---|---|---|---|
| R | G | 59 | 69 | V | C | 27 | 39 |
| R | L | 61 | 84 | V | G | 393 | 463 |
| R | O | 222 | 217 | V | M | 102 | 95 |
| R | S | 343 | 306 | V | N | 56 | 66 |
| S | D | 109 | 103 | V | Q | 41 | 67 |
| S | I | 48 | 44 | W | L | 392 | 377 |
| S | L | 84 | 84 | W | X | 405 | 412 |
| S | Q | 43 | 46 | W | Y | 302 | 230 |
| S | R | 206 | 184 | X | G | 43 | 42 |
| T | B | 348 | 284 | X | L | 107 | 101 |
| T | X | 398 | 389 | X | W | 442 | 402 |
| U | A | 93 | 105 | X | Y | 124 | 136 |
| U | G | 36 | 38 | Y | L | 45 | 60 |
| U | O | 55 | 49 | Y | W | 292 | 299 |
| U | V | 128 | 105 | Y | X | 72 | 87 |
| V | A | 37 | 67 | | | | |

表 11: 问题 2 结果-续表

Problem_2.py

```python
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM
import datetime

data = pd.read_excel(r"C:\Users\charl\Desktop\数学建模\Project\Project 4\附件1(Attachment 1)2023-51MCM-Problem B.xlsx").values

lines = np.unique(np.array([i[0]+i[1]for i in data[:,[1,2]]]))
lines = np.array([[i[0],i[1]] for i in lines])

all_pre = []

for i in range(lines.shape[0]):
    y = data[np.logical_and(data[:,1]==lines[i,0],data[:,2]==lines[i,1])][:,-1].astype(np.float64)
    # 将数据划分为训练集和测试集
    train_size = int(len(y) * 0.8)
    train, test = y[:train_size], y[train_size:]

    # 数据归一化
    scaler = MinMaxScaler()
    train = scaler.fit_transform(train.reshape(-1, 1))
    test = scaler.transform(test.reshape(-1, 1))

    # 创建数据生成器
    def create_dataset(data, look_back=1):
        X, Y = [], []
        for i in range(len(data) - look_back):
            X.append(data[i:(i + look_back), 0])
            Y.append(data[i + look_back, 0])
        return np.array(X), np.array(Y)

    look_back = 7
    X_train, y_train = create_dataset(train, look_back)
    X_test, y_test = create_dataset(test, look_back)

    # 重塑数据以适应 LSTM 模型的输入格式
    X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
    X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

    # 创建 LSTM 模型
    model = Sequential()
    model.add(LSTM(50, input_shape=(look_back, 1)))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')

    # 训练模型
    model.fit(X_train, y_train, epochs=100, batch_size=1, verbose=1)

    # 预测
    train_predict = model.predict(X_train)
    test_predict = model.predict(X_test)

    # 将预测结果转换回原始尺度
```

```
    train_predict = scaler.inverse_transform(train_predict)
    y_train = scaler.inverse_transform(y_train.reshape(-1, 1))
    test_predict = scaler.inverse_transform(test_predict)
    y_test = scaler.inverse_transform(y_test.reshape(-1, 1))

    # 计算预测准确性，例如使用均方误差（MSE）作为评估指标
    train_mse = np.mean((train_predict - y_train) ** 2)
    test_mse = np.mean((test_predict - y_test) ** 2)

    # 计算预测日期与最后一个训练日期之间的天数
    last_train_date = datetime.date(2019, 4, 17)
    start_pred_date = datetime.date(2019, 4, 18)
    end_pred_date = datetime.date(2019, 4, 20)
    days_to_predict = (end_pred_date-start_pred_date).days

    # 使用训练数据的最后一部分来开始预测
    input_data = train[-look_back:]

    predictions = []

    # 预测每一天的货量
    for i in range(days_to_predict):
        input_data_reshaped = input_data.reshape(1, look_back, 1)
        pred = model.predict(input_data_reshaped)
        predictions.append(pred[0, 0])

        # 更新输入数据，用预测值替换最早的值
        input_data = np.roll(input_data, -1)
        input_data[-1] = pred

    # 将预测值转换回原始尺度
    predictions = scaler.inverse_transform(np.array(predictions).reshape
    (-1, 1))

    tdays = []
    # 打印预测结果
    for i, pred in enumerate(predictions, start=1):
        pred_date = start_pred_date + datetime.timedelta(days=i - 1)
        tdays += [pred[0]]
    all_pre+=[tdays]

all_pre = np.array(all_pre)
all_pre
```

Problem_4.ipynb

```
from typing import Dict
import pandas as pd
from pandas import DataFrame
df_2 = pd.read_excel('Appendix_2.xlsx')
df_3 = pd.read_excel('Appendix_3.xlsx')
df_3.head()

def getEmptyDictPair(data):
    dic_pair = {}
    for index, row in data.iterrows():
        start = row[0]
        end = row[1]
        fix = row[2]
        pcs = row[3]
```

```
        dic_pair[(start, end)] = (fix, pcs)
    return dic_pair

day_23 = df_2[df_2.iloc[:, 0] == '2023-04-23']
day_23.head()

dic_nx = {}

lst_pair = [('A', 'C'), ('A', 'T'), ('C', 'L'), ('L', 'T'), ('T', 'E'),
    ('L', 'N'),
 ('N', 'E'), ('N', 'Q'), ('Q', 'P'), ('E', 'P'), ('E', 'W'), ('W', 'U'),
 ('P', 'U'), ('W', 'G'), ('G', 'U'), ('P', 'V'), ('V', 'F'), ('U', 'F'),
 ('G', 'O'), ('O', 'F'), ('F', 'R'), ('R', 'K'), ('V', 'K'), ('R', 'X'),
     ('O', 'X'), ('O', 'M'), ('K', 'J'), ('J', 'X'), ('X', 'I'), ('M', '
    I'), ('M', 'D'), ('D', 'S'), ('D', 'Y'), ('S', 'Y'), ('S', 'I'), ('I
    ', 'B'), ('B', 'J'), ('B', 'H'), ('H', 'J')]

for pair in lst_pair:
    dic_nx[pair] = 0

def calWeight(fix, pcs, real):
    return fix * (1 + (1.0*real/pcs)**3)

# 把某天的实际货运数据转化成日期的 dict

def getDictDay(data_day: DataFrame, pairs) -> Dict:
    dic_real = {}

    for pair in pairs:
        dic_real[pair] = 0

    for index, row in data_day.iterrows():
        start = row[1]
        end = row[2]
        real = row[3]
        dic_real[(start, end)] = real


    return dic_real

def getNx(day, start, end):
    dic_nx = {}

    pairs = [
        ('A', 'C'), ('A', 'T'), ('C', 'L'), ('L', 'T'), ('T', 'E'), ('L'
    , 'N'),
        ('N', 'E'), ('N', 'Q'), ('Q', 'P'), ('E', 'P'), ('E', 'W'), ('W'
    , 'U'),
        ('P', 'U'), ('W', 'G'), ('G', 'U'), ('P', 'V'), ('V', 'F'), ('U'
    , 'F'),
        ('G', 'O'), ('O', 'F'), ('F', 'R'), ('R', 'K'), ('V', 'K'), ('R'
    , 'X'),
        ('O', 'X'), ('O', 'M'), ('K', 'J'), ('J', 'X'), ('X', 'I'), ('M'
    , 'I'),
        ('M', 'D'), ('D', 'S'), ('D', 'Y'), ('S', 'Y'), ('S', 'I'), ('I'
    , 'B'),
        ('B', 'J'), ('B', 'H'), ('H', 'J')]

    for pair in pairs:
        dic_nx[pair] = 0

```

```
    row = df_3[df_3['起点 (Start)'] == start][df_3['终点 (End)'] == end]
    if(row.empty):
        return dic_nx
    fix = row.iloc[0, 2]
    pcs = row.iloc[0, 3]

    dic_day = getDictDay(day, pairs)

    for key, value in dic_nx.items():
        real = dic_day[key]
        dic_nx[key] = fix * (1 + (1.0*real/pcs)**3)

    return dic_nx

from networkx.classes.function import path_weight
def calculateTheDay(day, pairs):
    res = 0
    dic_real = getDictDay(day, pairs)

    for key, value in dic_real.items():
        dic_nx = getNx(day, key[0], key[1])
        network = nx.Graph()
        for pair, weight in dic_nx.items():
            network.add_edge(pair[0], pair[1], weight=weight)

        if(nx.shortest_path_length(network, key[0], key[1]) > 5):
            continue
        else:
            res += path_weight(network, nx.shortest_path(network, key
    [0], key[1], weight='weight'), weight='weight')

    return res


result = calculateTheDay(df_2[df_2.iloc[:, 0] == '2023-04-27'], pairs)
result
```