2022 年首届“钉钉杯”大学生
大数据挑战赛论文

题　　目　基于 Logistic 回归模型的航班数据分析与预测

摘　要

近年来，随着航班延误事件的频繁发生，既影响了人们的出行效率，也给航空公司造成了声誉上和经济上的损失，给很多乘客和航空公司带来了许多负面影响。本文基于题目附件中所给出的航班起降数据，采用 Python 编程语言，运用 Dijkstra 算法，同时还建立 Logistic 回归分析模型，分析航班是否延误以及延误的各类原因，提前对航班的延误情况进行预测，采取合理的措施，构建航班转机功能，实现时间最短方案，从而使乘客的出行进行合理规划。这样有助于提高空中资源的优化管理，提高生产生活效率，可以为乘客提供更优质的服务，对缓解航班延误产生的负面影响有着重要意义。

针对问题一，本文采用 Python 编程语言，在给定航班出发机场、目的机场、出行时间等数据信息的前提下，对原始数据进行统计以及数据融合、数据清洗、数据转换，然后进行数据分析，采用 Dijkstra 算法，将算法中原本的距离替换为数据中的航班飞行时长，从而构建航班转机功能，确定时间最短的航班转机方案。最终根据题目要求，查询到 2003 年 7 月 4 日出发，2003 年 7 月 5 日到达，从 CVG 机场到 ANC 机场的最短时间转机方案为 CVG-ORD-ANC。

针对问题二，本文先明确航班延误的含义，分两种情况进行研究。对附件给定的数据进行统计以及数据预处理，展开数据分析，确定对航班延误预测影响的特征，从而进行数据建模。建立起分别以影响航班是否延误以及延误原因的指标作为自变量，以是否延误为因变量的 Logistic 回归分析模型。然后对 Logistic 回归分析模型进行训练，再将测试集特征集带入训练完成的预测模型中，得到预测结果。训练过的模型可以预测出近 100%的准确率，表现出了较高的预测精度，可以为航班延误的预测问题提供较为准确的参考。

综上所述，本文依据各题所给条件全面分析了相关因素对航班延误的影响，并对乘客的出行进行了合理规划。经过分析验证，本文的模型具有合理性和一定的现实意义。

关键词：　航班延误；Dijkstra 算法；Logistic 回归分析模型；数据分析；预测

目录

一、问题重述 1
1.1 问题的背景和意义 1
1.2 问题的重述 1
1.2.1 问题一 1
1.2.2 问题二 2
二、问题分析 2
2.1 问题一的分析 2
2.2 问题二的分析 2
三、模型假设 3
四、问题一的求解 3
4.1 数据预处理 4
4.1.1 数据介绍 4
4.1.2 数据处理 5
4.2 基于 Dijkstra 算法的数据分析 6
4.2.1 Dijkstra 算法介绍 6
4.2.2 Dijkstra 算法描述 6
4.2.3 数据分析 10
4.3 结果分析 14
五、问题二的求解 16
5.1 航班延误的相关定义 16
5.2 问题二（1）的数据预处理 17
5.2.1 数据介绍 17
5.2.2 特征值和目标值的选取 17
5.2.3 数据处理 17
5.3 问题二（1）的数据分析 19
5.4 问题二（1）数据建模 22
5.4.1 训练集航班是否延误对比 22
5.4.2 Logistic 回归分析模型介绍 22
5.4.3 基于 Logistic 回归分析模型的预测 22
5.5 问题二（2）数据预处理 25

5.5.1 数据介绍 25
5.5.2 特征值和目标值的选取 25
5.5.3 数据处理 25
5.6 问题二（2）数据分析 25
5.7 问题二（2）数据建模 25
5.7.1 训练集航班延误原因时长对比 25
5.7.2 基于 Logistic 回归分析模型的预测 26
六、总结 27
参考文献 28
附录 28
附录I 使用软件 28
附录II 第一题完整代码 28
附录III 第二题完整代码 32

一、 问题重述

1.1 问题的背景和意义

随着科技的发展，以及航空业的不断发展，航空旅客运输量也不断增加，乘坐飞机出行为人们的生活带来了极大的便利，航空交通管理影响着人们的工作和生活效率。在大数据和人工智能时代的今天，各种各样的信息科学和工程技术广泛应用于航空领域，为人们的生产生活提供更高的便利性，因此提高空中资源的优化配置，一直都是计算机科学与技术、信息科学与工程、数学等领域的一门热点研究方向。由于航线网络越来越密集，航线结构越来越复杂，但是受到空中资源、机场设施以及航空公司的限制，再加上天气、出发地或目的地机场管理、航空公司管理、旅客个人缘由等原因，航空运输的需求和容量之间的矛盾也变得日益突出，航班延误甚至取消航班的情况愈来愈多，使得航班延误成为衡量航空运输系统效率的重要指标。

航空延误的频繁发生，不仅会影响机场以及管制部门的正常运行，额外增加航空公司的运营成本，造成公共运输服务资源的浪费，加重管制的工作负荷，增加航空安全风险。还会影响乘客的出行体验，造成时间上的浪费，降低乘客对于承运人的信任度和服务满意度。在发生大面积延误时，会打乱乘客的实际规划，大量滞留在机场的乘客很可能会引发混乱与纠纷[1]，甚至与工作人员发生冲突，严重的情况还会波及整个机场群的正常运行，给航空公司带来负面影响，严重影响社会经济生活和交通运输保障体系。

当前，航空延误成为了空中资源优化配置的一项经典课题。分析与预测航空延误有助于提高资源的优化管理，使相关负责人员及时对航班排班进行有序的调度与合理的资源分配，同时给航空公司、机场、空管单位以提前预警有助于其提前做好航班延误处置计划，减少经济损失，提高生产生活效率，更可以给以乘客提示以便其做好出行计划，从而提供更优质的服务。在一定程度上尽可能降低延误带来的影响。

1.2 问题的重述

基于以上背景，根据附件给出的 2001-2005 年各机场航班起降数据、所有机场与航空公司信息以及天气信息，我们将进行数据分析，并且建立合适的数学模型进行实验，来解决以下问题：

1.2.1 问题一

航班转机功能实现：当两个城市之间没有直飞航班或者在购买机票附件时间没有直飞航班的时候，乘客通常需要购票 APP 实现转机功能。

（一）以附件中 2001-2003 年航班数据作为依据，实现在 2001-2003 年从 A 地到 B 地的多种转机方案中找到在包含航班延误的情况下时间最短的方案的航班转机功能。

（二）用上述转机功能，查询 2003 年 7 月 4 日出发 7 月 5 日到达，从 CVG 机场到

ANC 机场最短时间方案。

1.2.2 问题二

进行迈阿密（MIA）起飞航班的延误分析：

先给出我们对问题（一）和（二）航班延误的分析，再建立模型做实验得到航班是否延误的准确率，以实验结果检验我们最初的分析。

（一）以附件中的 2001-2003 年的航班数据作为训练集，以附件 2004-2005 年的航班数据作为测试集，以从迈阿密（MIA）到洛杉矶（LAX）和从迈阿密（MIA）到纽约（JFK）这两组航班数据作为研究对象，先以文字形式叙述预测航班是否延误的依据，再建立模型预测从迈阿密（MIA）起飞航班（从 MIA 到 LAX 和从 MIA 到 JFK）是否延误，以预测准确率和实验结果检验我们的分析。

提示：可以在训练、验证和预测中使用机场所在地天气情况等信息，详见附件数据属性说明表。

（二）以附件中的 2001-2003 年的航班数据作为训练集，以附件 2004-2005 年的航班数据作为测试集，以从迈阿密（MIA）到洛杉矶（LAX）和从迈阿密（MIA）到纽约（JFK）这两组航班数据作为研究对象，先以文字形式分析航班延误的各种原因，再建立模型预测从迈阿密（MIA）起飞航班（从 MIA 到 LAX 和从 MIA 到 JFK）延误的原因，以预测延误原因的准确率和实验结果检验我们的分析。

二、 问题分析

2.1 问题一的分析

题目要求我们以附件中 2001-2003 年航班数据作为依据，实现在 2001-2003 年的航班转机功能。该题是一个规划问题，我们需要从中找到最短时间方案，由于航班数量太多，可以进行数据预处理、数据分析通过最短路径算法来达到目的。首先我们在附件中提取出需要的原始数据，对原始数据进行数据融合、数据清洗、数据转换，然后对处理完的数据进行分析，通过最短路径算法 Dijkstra 算法，将原算法中的距离替换为时间，即可实现转机功能。最后在实现功能后，按照题目要求查询 CVG 到 ANC 机场的最短时间转机方案。具体流程图如下：

2.2 问题二的分析

题目要求我们以附件中的 2001-2003 年的航班数据作为训练集，以附件 2004-2005 年的航班数据作为测试集，以从 MIA 到 LAX 和从 MIA 到 JFK 这两组航班数据作为研究对象，来对航班延误以及延误原因进行预测。在解决问题之前，我们首先需要明确航班延误的定义，从而筛选出我们需要的航班数据。得到数据以后，先对数据进行预处理和分析，以文字形式叙述航班是否延误以及延误的原因，从而选取需要特征值，在进行

数据分析以后，我们可以得到训练集和测试集的目标集和特征集。接下来，我们就可以建立 Logistic 回归分析模型，用训练集对模型进行训练，然后对测试集进行预测，将预测结果与目标集对比进行分析，通过预测准确率等各种指标和实验结果检验我们的分析。具体流程图如下：

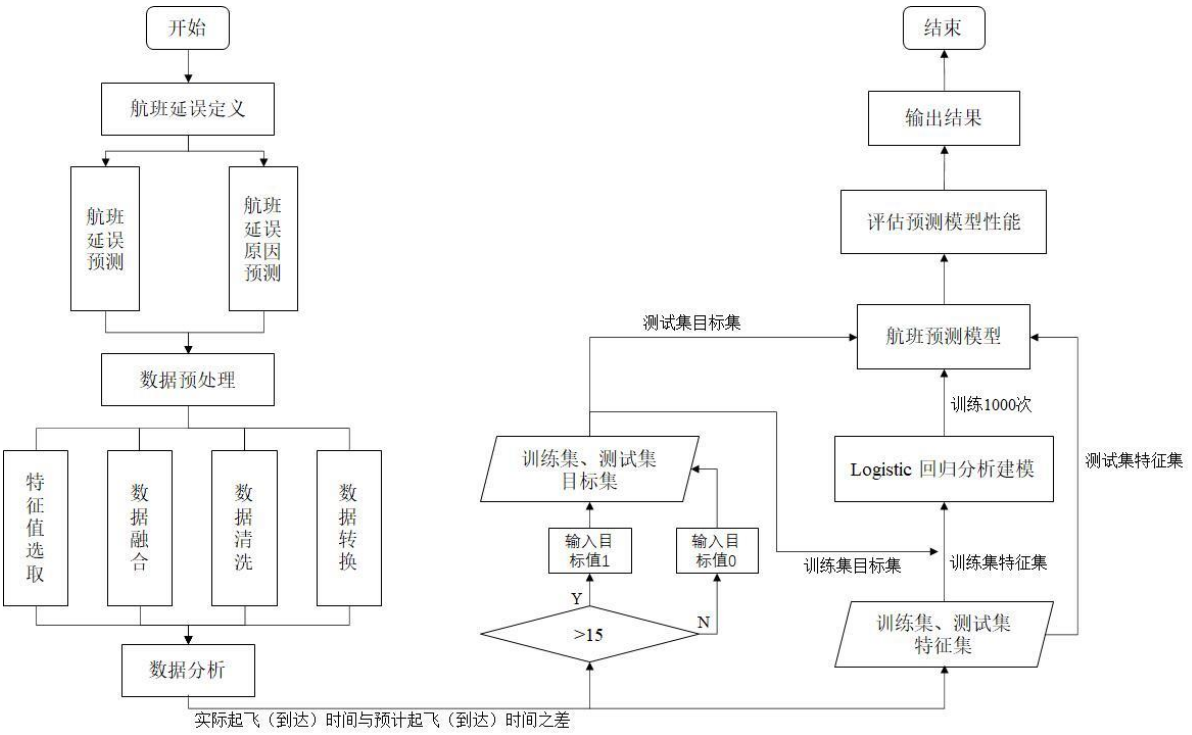


图 1 问题二思维导图

三、 模型假设

（1）假设附件所提供的数据及使用的数据都是真实准确的，数据量化合理有效。

（2）假设所分析年份内未发生重大事件和变故等特殊情况影响航空公司相关政策与管理。

（3）假设所分析年份内未发生较大的人口迁入和迁出等情况致使航班非正常增加。

（4）假设所分析年份内未发生重大事故和灾难等情况致使航班非正常取消。

（5）假设在第一问研究航班转机方案时不考虑在转机机场等待的时间。

四、 问题一的求解

当今社会，人们的出行越来越方便，当我们要从一个城市前往另一个城市时，如果两个城市之间没有直飞航班或者在购买时没有直飞航班的话，我们往往需要换乘中转。通过对问题一进行分析，我们将根据附件给出的 2001-2003 年航班数据，来实现在 2001-2003 年从 A 地到 B 地的多种转机方案中找到在包含航班延误的情况下时间最短方案的航班转机功能。具体流程如下图 2 所示。

而经过粗略统计，2001-2003 年航班数据累计共有 1700 多万条，每条数据包括 29 个属性，由于数据庞大且并不是每一个属性都是我们需要的，所以我们首先对数据进行预处理。

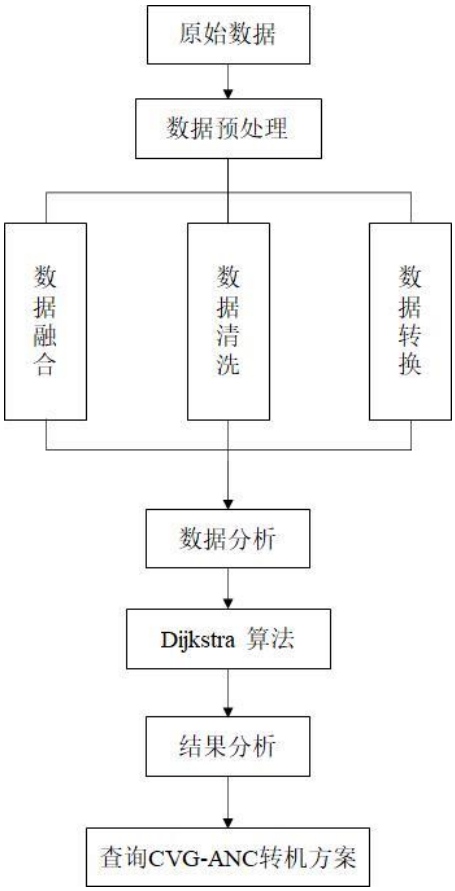


图 2 问题一具体流程图

4.1 数据预处理

4.1.1 数据介绍

本文附件提供了一个数据集，数据集共有 4 种不同的表格和两个文件，分别为 2001-2005 年各机场的航班起降数据、包含所有机场信息的 airports 表、包含航空公司信息的 carriers 表、包含相应机场天气信息的 rawweatherdata 表以及附件数据属性说明、数据集说明两个文件。

本题使用的数据主要是由附件提供的 2001-2003 年各机场的航班起降数据，原始数据共有 17727679 条，其中每条数据由 Year（对应年份）、Month（对应月份）、DayOfMonth（航班在一个月中的哪一天起飞）、DayOfWeek（航班在一星期中的哪一天起飞）、DepTime（实际起飞时间）、ArrTime（实际到达时间）、ActualElapsedTime（实际到达时间与实际起飞时间之差）等等共 29 个属性组成。另外，airports 表和附件数据属性说明文件也给我们提供了很大的帮助。

4.1.2 数据处理

拿到数据以后，我们首先对数据进行整理，使我们充分了解数据。在整理的过程中，我们发现：根据题目要求以及附件属性说明文件，我们需要重点考虑的数据属性有年份表中的飞机起飞时滑行时间、飞机降落时滑行时间、空中飞行时间、实际到达时间与实际起飞时间之差、实际起飞时间、出发机场、到达机场以及航班是否被取消这 8 个属性和 airports 表中的国际机场缩写、机场名称、机场所在城市以及机场所在的州这 4 个属性，另外我们还需得到起飞日期这一属性，而其他的属性则对问题没有影响。

因此，我们将利用 python 代码实现数据融合、数据清洗和数据转换。

1.数据融合是将数据综合与加工的过程，是为了完成既定的决策和评价任务，对在不同数据源上的数据信息按照一定的准则加以综合分析[2]。本文是利用年份表中的对应年份、对应月份、航班在一个月中的哪一天起飞这三个属性融合转化为新的一列“xxxx-x-x”格式的起飞日期。

2.数据清洗是指利用一定的方法对原始数据存在的“脏数据”进行清洗，以便增强数据的质量。本文主要是将对应年份、对应月份、航班在一个月中的哪一天起飞以及上文未列出的问题不需要的数据删除掉。

3.数据转换是将数据由某种形式转换成另外一种形式，数据转换的方式有语言转换、语义转换、数据标准化等。本文是将问题需要的数据英文列名替换为中文列名，方便阅读理解，增加数据可读性。

详细代码如下：

```
# 函数名：data_process
# 参数说明：
#  flight_data：需要清洗的数据
# 作用：数据清理和数据转换，删除不需要的数据，将英文列名替换为中文列名，方便阅读理解
def data_process(flight_data)：
  flight_data['Day'] = flight_data['DayofMonth']
  flight_data['起飞日期'] = pd.to_datetime(flight_data[['Year','Month','Day']])
  # 删除年月日和不需要的数据数据，节约内存
  clean_data = flight_data.drop(columns=['Year', 'Month', 'DayofMonth', 'LateAircraftDelay',
                                         'SecurityDelay', 'NASDelay', 'WeatherDelay', 'CarrierDelay',
                                         'Distance', 'CRSElapsedTime', 'DayOfWeek', 'FlightNum'])
  # 修改列名
  clean_data.rename(columns={'Cancelled': '航班是否取消', 'TaxiIn': '飞机起飞时滑行时间
                             ', 'TaxiOut': '飞机降落时滑行时间', 'Origin': '出发机场', 'Dest': '到达
```

```
机场', 'ActualElapsedTime': '实际到达时间与实际起飞时间之差',
'DepTime': '实际起飞时间', 'airport': '机场名称', 'UniqueCarrier': 'iata
代码', 'AirTime': '空中飞行时间'}, inplace = True)
    return clean_data
```

4.2 基于 Dijkstra 算法的数据分析

经过对原始数据的预处理，我们对于数据信息的掌握更加清楚。接下来，我们将对数据进行分析，根据题目得知要实现从 A 地到 B 地时间最短方案的航班转机功能，经过我们研究讨论发现，Dijkstra 算法是图论中经典的求最短路径的算法，如果把其中的距离替换为时间，即可实现两地之间最短时间的求解。因此，我们最终决定使用图论算法中的 Dijkstra 算法来完成这一航班转机功能。

4.2.1 Dijkstra 算法介绍

图论（GraphTheory）是数学的一个分支。它以图为研究对象。图论中的图是由若干给定的点及连接两点的线所构成的图形，这种图形通常用来描述某些事物之间的某种特定关系，用点代表事物，用连接两点的线表示相应两个事物间具有这种关系。图提供了一种处理关系和交互等抽象概念的更好的方法，它还提供了直观的视觉方式来思考这些概念。图论的应用十分广泛，它经常用于解决航空系统、交通流、社交网络、电商推荐系统等问题。而 Dijkstra 算法是图论中经典的求最短路径的算法，可以用来实现本题所要求的功能。

迪杰斯特拉算法(Dijkstra)是由荷兰计算机科学家狄克斯特拉于 1959 年提出的，因此又叫狄克斯特拉算法。是从一个顶点到其余各顶点的最短路径算法，解决的是有权图中最短路径问题。迪杰斯特拉算法主要特点是从起始点开始，采用贪心算法的策略，每次遍历到始点距离最近且未访问过的顶点的邻接节点，直到扩展到终点为止[3]。

4.2.2 Dijkstra 算法描述

我们通过一个简单的问题来描述 Dijkstra 算法的原理。如下：

问题描述：在无向图 G=(V,E)中，假设每条边 E[i]的长度为 d[i]，找到由顶点 v 到其余各点的最短路径。（单源最短路径）

算法思想：设 G=(V,E)是一个带权有向图，把图中顶点集合 V 分成两组，第一组为已求出最短路径的顶点集合（用 S 表示，初始时 S 中只有一个源点，以后每求得一条最短路径,就将加入到集合 S 中，直到全部顶点都加入到 S 中，算法就结束了），第二组为其余未确定最短路径的顶点集合（用 U 表示），按最短路径长度的递增次序依次把第二组的顶点加入 S 中。在加入的过程中，总保持从源点 v 到 S 中各顶点的最短路径长度不大于从源点 v 到 U 中任何顶点的最短路径长度。此外，每个顶点对应一个距离，S 中的顶点的距离就是从 v 到此顶点的最短路径长度，U 中的顶点的距离，是从 v 到此顶点

只包括 S 中的顶点为中间顶点的当前最短路径长度[4]。

①初始时，S 只包含起点 v；U 包含除 v 外的其他顶点，且 U 中顶点的距离为“起点 v 到该顶点的距离”（例如，U 中顶点 a 的距离为(v,a)的长度，然后 a 和 v 不相邻，则 a 的距离为∞）。

②从 U 中选出“距离最短的顶点 k”，并将顶点 k 加入到 S 中；同时，从 U 中移除顶点 k。

③更新 U 中各个顶点到起点 v 的距离。之所以更新 U 中顶点的距离，是由于上一步中确定了 k 是求出最短路径的顶点，从而可以利用 k 来更新其它顶点的距离；例如，(v,a)的距离可能大于(v,k)+(k,a)的距离。

③重复步骤②和③，直到遍历完所有顶点。

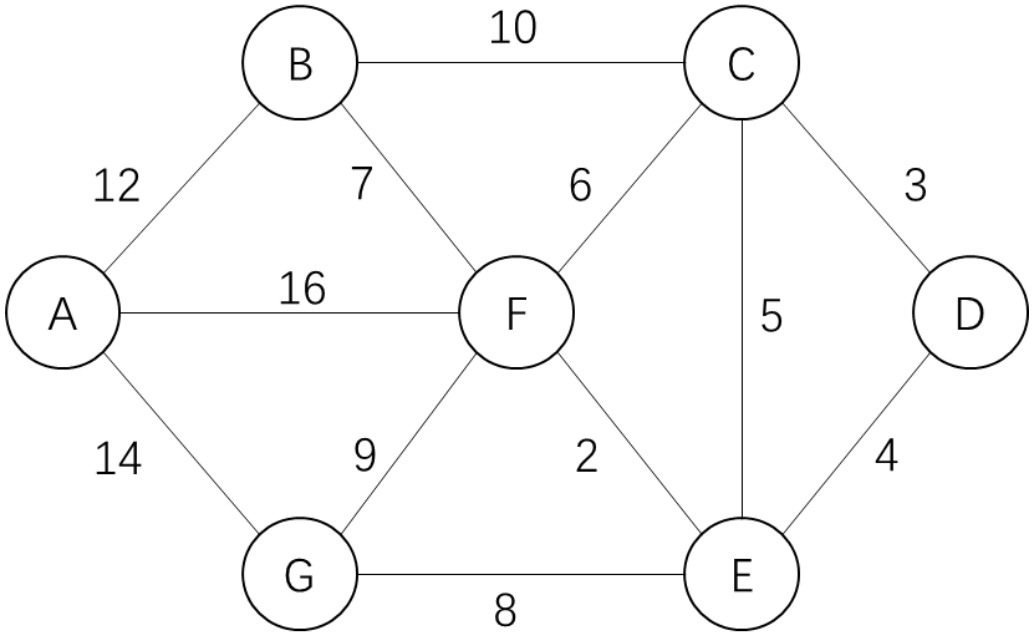
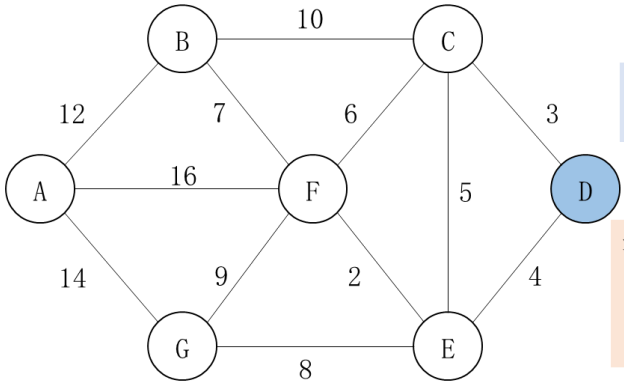


图 3 问题描述所需无向图

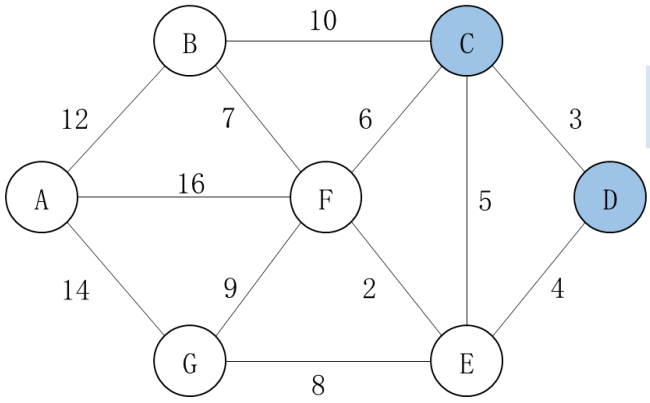
以图 1 为例，来描述 Dijkstra 进行算法演示，我们以第四个顶点 D 为起始点。



S={D(0)}
U={A(∞),B(∞),C(3),E(4),F(∞),G(∞)}

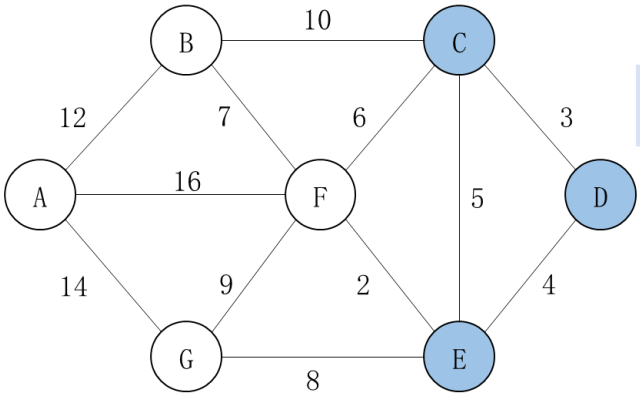
注：
（1）S是已计算出最短路径的顶点集合
（2）U是未计算出最短路径的顶点集合
（3）C(3)表示顶点C到起点D的最短距离是3

（a） 第一步：选取顶点 D



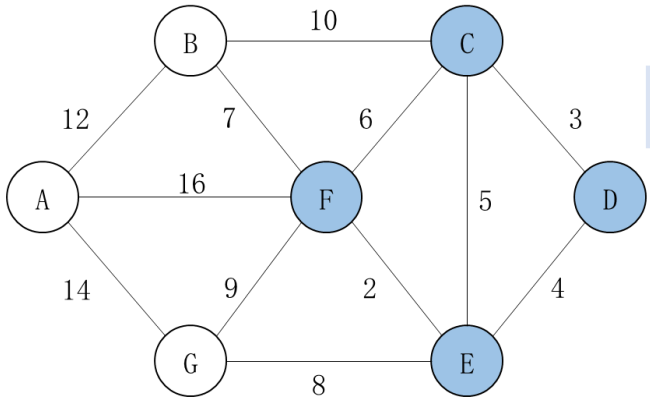
S={D(0), C(3)}
U={A(∞), B(13), E(4), F(9), G(∞)}

（b）第二步：选取顶点 C



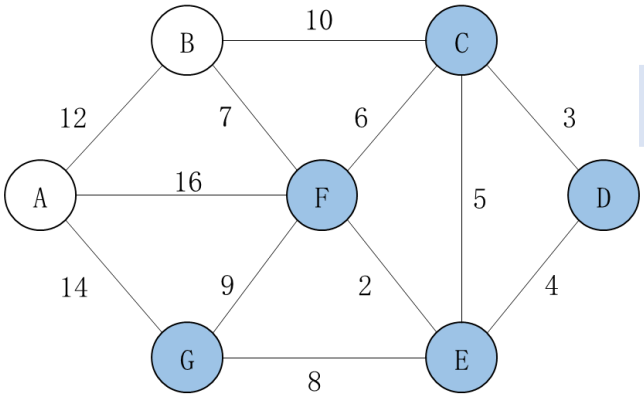
S={D(0), C(3), E(4)}
U={A(∞), B(13), F(6), G(12)}

（c） 第三步：选取顶点 E



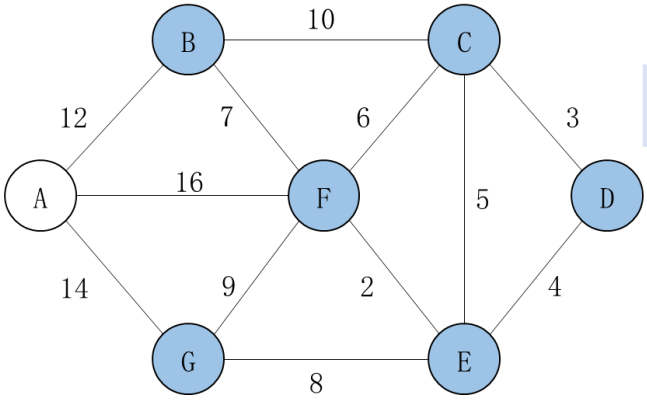
S={D(0), C(3), E(4), F(6)}
U={A(22), B(13), G(12)}

（d） 第四步：选取顶点 F



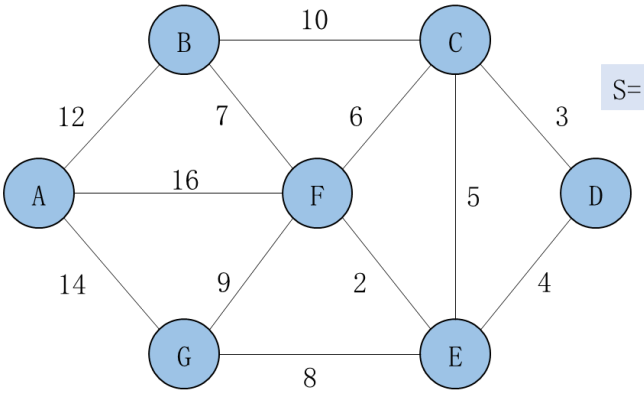
S={D(0)，C(3)，E(4)，F(6)，G(12)}
U={A(22)，B(13)}

（e） 第五步：选取顶点 G



S={D(0),C(3),E(4),F(6),G(12),B(13)}
U={A(22)}

（f） 第六步：选取顶点 B



S={D(0),C(3),E(4),F(6),G(12),B(13),A(22)}

（g） 第七步：选取顶点 A

图 4 Dijkstra 算法问题描述过程演示图

初始状态：S 是已计算出最短路径的顶点集合，U 是未计算除最短路径的顶点的集合！

第 1 步：将顶点 D 加入到 S 中。

此时，S={D(0)}, U={A(∞),B(∞),C(3),E(4),F(∞),G(∞)}。

第 2 步：将顶点 C 加入到 S 中。

上一步操作之后，U 中顶点 C 到起点 D 的距离最短；因此，将 C 加入到 S 中，同时更新 U 中顶点的距离。以顶点 F 为例，之前 F 到 D 的距离为∞；但是将 C 加入到 S 之后，F 到 D 的距离为 9=(F,C)+(C,D)。

此时，S={D(0),C(3)}, U={A(∞),B(13),E(4),F(9),G(∞)}。

第 3 步：将顶点 E 加入到 S 中。

上一步操作之后，U 中顶点 E 到起点 D 的距离最短；因此，将 E 加入到 S 中，同时更新 U 中顶点的距离。还是以顶点 F 为例，之前 F 到 D 的距离为 9；但是将 E 加入到 S 之后，F 到 D 的距离为 6=(F,E)+(E,D)。

此时，S={D(0),C(3),E(4)}, U={A(∞),B(13),F(6),G(12)}。

第 4 步：将顶点 F 加入到 S 中。

此时，S={D(0),C(3),E(4),F(6)}, U={A(22),B(13),G(12)}。

第 5 步：将顶点 G 加入到 S 中。

此时，S={D(0),C(3),E(4),F(6),G(12)}, U={A(22),B(13)}。

第 6 步：将顶点 B 加入到 S 中。

此时，S={D(0),C(3),E(4),F(6),G(12),B(13)}, U={A(22)}。

第 7 步：将顶点 A 加入到 S 中。

此时，S={D(0),C(3),E(4),F(6),G(12),B(13),A(22)}。

此时，起点 D 到各个顶点的最短距离就计算出来了**: A(22) B(13) C(3) D(0) E(4) F(6) G(12)**。

4.2.3 数据分析

通过对收集到的数据分析发现，我们采用 Dijkstra 算法需要先确定起始点和终点，在本题中即为出发机场和到达机场。由于本题的性质，我们要确定的出发机场和到达机场不仅仅是机场名字，更要包括航班出发日期（起飞日期）、航班到达日期（降落日期）以及机场所在州与城市。

首先，我们先调用 numpy、pandas、networkx、matplotlib 库以及附件给出的数据表。这里我们先对调用的四个库进行介绍：

①numpy 是一个开源的 Python 扩展库，它是 Python 科学计算的基础包。 NumPy 用来支持大数据量的高维数组和矩阵运算，比 Python 自身的嵌套列表（该结构也可以用来表示矩阵）结构要高效的多。 NumPy 模拟了 MATLAB 的常用功能，开发 NumPy 库是为了让 Python 也具有和 MATLAB 一样强大的数值计算能力。

②Pandas 是 python 的核心数据分析支持库，提供了快速、灵活、明确的数据结构，旨在简单、直观地处理关系型、标记型数据。Pandas 的目标是成为 python 数据分析与实战的必备高级工具，其长远目标是成为最强大、最灵活、可以支持任何语言的开源数据分析工具。

④networkx 是一个用 python 语言开发的图论与复杂网络建模工具，内置了常用的图与复杂网络分析算法，可以方便的进行复杂网络数据分析、仿真建模等工作。networkx 支持创建简单无向图、有向图和多重图；内置许多标准的图论算法，节点可为任意数据；支持任意的边值维度，功能丰富，简单易用。利用 networkx 可以以标准化和非标准化的数据格式存储网络、生成多种随机网络和经典网络、分析网络结构、建立网络模型、设计新的网络算法、进行网络绘制等。

④matplotib 是一款可以数据可视化的库。由各种可视化的类构成。matplotib .pyplot 是绘制各类可视化 Q 图形的命令子库。通常别名为 plt 命令如 import matplotlib.pyplot as plt。matplotib 通常和 numpy 结合使用。

详细代码如下：

```
import numpy as np
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
import matplotlib

# 机场数据
airport_data = pd.read_csv("./数据集/airports.csv", encoding='unicode_escape', low_memory=False)
# 天气数据
wheather = pd.read_csv("./数据集/rawweatherdata.csv", encoding='unicode_escape', low_memory=False)
# flight_data = pd.read_csv('./数据集/2001.csv', encoding='unicode_escape', low_memory=False)
# flight_data1 = pd.read_csv('./数据集/2002.csv', encoding='unicode_escape', low_memory=False)
# flight_data2 = pd.read_csv('./数据集/2003.csv', encoding='unicode_escape', low_memory=False)
# print(flight_data.shape)
# print(flight_data1.shape)
# print(flight_data2.shape)
```

其次，由于我们的数据是根据年份分为三个表格，所以需先确定数据年份，来获取当前年的所有飞行数据。我们仍然用 python 代码实现。详细代码如下：

```
# 函数名：read_data
```

```
# 参数说明：
#  flight_year:要获取的数据年份
# 作用：获取当前年所有飞行数据
def read_data(flight_year):
  year = flight_year.split('-')[0]
  flight_data = pd.read_csv('./数据集/{0}.csv'.format(year), encoding='unicode_escape', low_memory=False)
  return flight_data
```

确定数据年份后，我们还需进一步确定出发时间和到达时间，来获取该时间段内的所有飞行数据。同时在这里我们还进行了航班是否取消的判别，详细代码如下：

```
# 函数名：flightday_data
# 参数说明：
#  clean_data：清洗完成的数据
#  outset_time：出发日期
#  arrive_time：到达日期
# 作用：返回时间段内的所有飞行数据
def flightday_data(clean_data, outset_time, arrive_time):
  twoday_data = clean_data.loc[(clean_data['起飞日期'] == '{0}'.format(outset_time)) | (clean_data['起飞日期'] == '{0}'.format(arrive_time))][clean_data['航班是否取消'] == 0]
  return twoday_data
```

在确定完航班的具体出发日期和到达日期后，我们还需要确定机场。由于年份表中出现的机场名称是机场缩写，所以我们决定用机场所在州与城市来确定机场名称，然后再将机场名称替换为机场缩写。但是在我们分析的过程中发现，airports 表中存在同名城市分属两个不同的州以及同一个城市有多个机场的现象。最终我们采用了在替换为机场缩写后从当前出发日期和到达日期的航班数据中判别该机场是否存在，如不存在则重新获取另一个机场，如存在则获取无误的方法解决了这一问题。具体代码如下：

```
# 函数名：departure
# 参数说明：
#  city：城市名
#  state：州名
# 作用：输入地名返回机场缩写
def departure(city, state):
  num = []
  real_num = []
```

```
    airports = airport_data.loc[airport_data['state'] == state]
    airports = airports.loc[airport_data['city'] == city]
    # 获取当前城市所有机场缩写
    for i in range(0, len(airports)):
        if airports.iat[i, 0] not in twoday_data['出发机场'].values:
            if airports.iat[i, 0] not in twoday_data['到达机场'].values:
                print('{0}机场不存在'.format(airports.iat[i, 0]))
                h = airports[airports.iata == airports.iat[i, 0]].index.tolist()[0]
                num.append(h)
            else：
                print('{0}机场存在'.format(airports.iat[i, 0]))
                h = airports[airports.iata == airports.iat[i, 0]].index.tolist()[0]
                real_num.append(h)
    airports = airports.drop(num)
    airports_name = airports.iat[0, 0]
    return airports_name
```

在我们需要的信息都已经获取完毕后，我们就可以求出当前时间段内最短时间的路径，并通过图的方式将这段时间所有的航班路线展现出来。在这一步中我们就借用了上文提到几种库。具体代码如下：

```
# 函数名：shortest_path
# 参数说明：
#   twoday_data: 出发日期到到达日期段内所有数据
#   outset_name：出发机场
#   arrive_name：到达机场
# 作用：求出当前时间段内最短时间的路径并画图
def shortest_path(twoday_data, outset_name, arrive_name):
    FG = nx.from_pandas_edgelist(twoday_data, source='出发机场', target='到达机场', edge_attr='实际到达时间与实际起飞时间之差')
    nx.draw_networkx(FG, with_labels=True, font_size=5, node_size=50, arrows=True)
    dijpath = nx.dijkstra_path(FG, source='{0}'.format(outset_name), target='{0}'.format(arrive_name), weight='实际到达时间与实际起飞时间之差')
    print('从{0}到{1}的最短飞行时间转机方案为{2}'.format(outset_name, arrive_name, dijpath))
    plt.title('{0}到{1}飞行路线有向图'.format(outset_time, arrive_time))
```

```
    plt.savefig('./{0}-{1}.png'.format(outset_name, arrive_name))
    plt.show()
```

最后，在每一部分的作用都可以实现后，我们编写了主程序，统一调用上述封装的函数来实现航班转机功能。同时主函数也产生出一个运行窗口，方便我们输入出发时间、到达时间、出发州与城市以及到达州与城市。具体代码如下：

```
if __name__ == '__main__':
    # name = departure('Wrangell', 'AK')
    outset_time = input('请输入出发时间(xxxx-x-x)：')
    arrive_time = input('请输入到达时间(xxxx-x-x)：')
    outset_state = input('请输入出发州：')
    outset_city = input('请输入出发城市：')
    arrive_state = input('请输入到达州：')
    arrive_city = input('请输入到达城市：')
    flight_data = read_data(outset_time)
    clean_data = data_process(flight_data)
    twoday_data = flightday_data(clean_data, outset_time, arrive_time)
    outset_name = departure(outset_city, outset_state)
    arrive_name = departure(arrive_city, arrive_state)
    shortest_path(twoday_data, outset_name, arrive_name)
```

4.3 结果分析

经过以上努力，我们通过 python 语言运用 Dijkstra 算法，编写程序实现了 2001-2003 年从 A 地到 B 地的多种转机方案中找到在包含航班延误的情况下时间最短方案的航班转机功能，具体代码见附录 II 所示。

现在我们需使用该转机功能，来查询 2003 年 7 月 4 日出发 7 月 5 日到达，从 CVG 机场到 ANC 机场最短时间方案。在主程序产生的运行窗口的相应位置依次输入 2003-7-4、2003-7-5、KY、Covington、AK、Anchorage，我们就得到了从 CVG 机场到 ANC 机场的最短时间转机方案是 CVG-ORD-ANC，其中 ORD 是位于 TX 州 Orange 城市的 Orange County 机场。

```
请输入出发时间(xxxx-x-x): 2003-7-4
请输入到达时间(xxxx-x-x): 2003-7-5
请输入出发州: KY
请输入出发城市: Covington
请输入到达州: AK
请输入到达城市: Anchorage
CVG机场存在
F:\数模\2022年首届钉钉杯大学生大数据挑战赛初赛题目\B题\ddb1.p
  twoday_data = clean_data.loc[(clean_data['起飞日期']
[1202]
CVG
ANC机场存在
LHD机场不存在
MRI机场不存在
[839]
ANC
从CVG到ANC的最短飞行时间转机方案为['CVG', 'ORD', 'ANC']
```

图 5 运行窗口与运行结果

同时我们还得到了 2003 年 7 月 4 日出发到 2003 年 7 月 5 日到达的所有机场航班的飞行路线有向图。如下图：

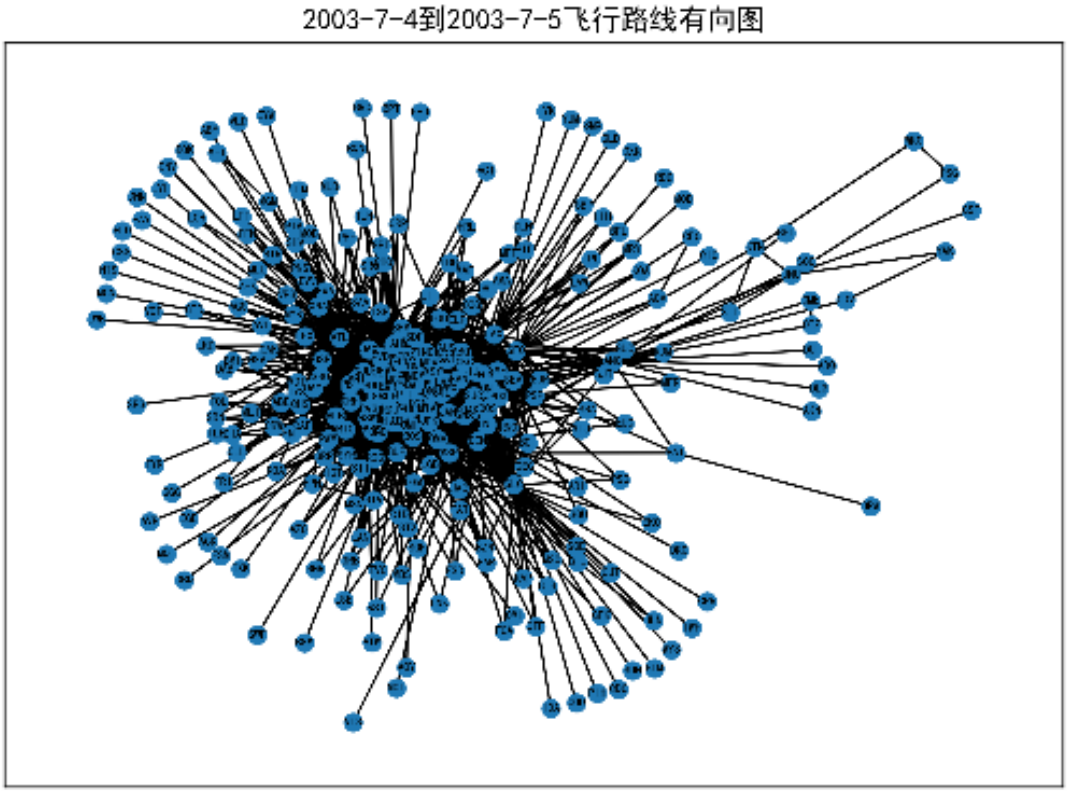


图 6 2003-7-4 到 2003-7-5 飞行路线有向图

五、 问题二的求解

对于问题二的求解，在解决问题之前，我们需要先明晰航班延误的含义，才能知道我们需要哪些数据和指标，找到方式方法，从而进行接下来的求解。

5.1 航班延误的相关定义

“航班延误”这一说法最早是在 1929 年第一部国际民用航空运输公约即《华沙公约》中出现的，其第 19 条规定“航空承运人应为航空运输途中发生的旅客、行李和货物未按时送达的产生的损失提供补偿”，但是其并没有给出航班延误的明确定义。随后 1955 年的《海牙议定书》和 1999 年的《蒙特利尔公约》两部民航公约都未给出关于航班延误的明确界定标准。因为世界各国的民航发展水平不同和各国的民航业运输体系差异的存在，导致各国对于航班延误的定义也不尽相同[5]。下面给出中美欧以及世界航空运输协会关于航班延误的相关定义，具体见表 1。

表 1 各组织航班延误相关规定对比 [7]

官方组织	航班延误相关规定
中国民航局（CCAC）	航班延误是指航班实际到港机务人员挡上轮档的时间晚于计划到港时间大于 15 分钟的情形；航班出港延误是指航班实际出港机务人员撤去轮档的时间晚于计划出港撤轮档时间大于 15 分钟的情形[6]
美国联邦航空管理局（FAA）	准时航班的定义是航班在计划到达时间的 15 分钟内抵达的情形；对航班延误的定义是航班到达或者离开机位时间比计划到达或离开时间晚于 15 分钟以上的情形
欧洲航行安全组织（CFMU）	航班延误是指航班的实际运行所用的时间长度超过计划航班运行的时间长度；航班延误正常率是指航班实际运行的时间长度超过计划航班运行的时间长度 15 分钟以内的航班量占据总的航班数据量的比例，即不正常航班是指航班的延误时间超过 15 分钟的情况
国际航空运输协会（IATA）	航班出港延误是指航班在预计离港时间后 15 分钟外推出的情形

通过对比表 1 中各国民航管理部门和国际航空运输协会对于航班延误的规定，不难发现对于航班延误的定义往往是通过比较实际进（离）港时间与计划进（离）港时间加上一定的缓冲时间值后是否一致作为评判航班是否延误的标准。

由于在本次研究中，绝大多数机场地理位置都在美国，所以结合美国联邦航空管理局关于航班延误的相关规定，现在将本文中航班延误的相关定义如下：延误航班是指航

班实际起飞（到达）时间晚于计划起飞（到达）时间超过 15 分钟的情形。以下我们就分别对这两种情况进行求解。

5.2 问题二（1）的数据预处理

5.2.1 数据介绍

在题目中，我们得知需以 2001-2003 年的航班数据作为训练集，以附件 2004-2005 年的航班数据作为测试集，以从迈阿密（MIA）到洛杉矶（LAX）和从迈阿密（MIA） 到纽约（JFK）这两组航班数据作为研究对象。

5.2.2 特征值和目标值的选取

目标值为航班是否延误，特征值为实际起飞时间、实际到达时间、预计到达时间与预计起飞时间之差、实际起飞与预计起飞之差、实际到达与预计到达之差以及最低能见度。

5.2.3 数据处理

在确定数据以后，我们首先对数据进行处理。首先分别将测试集和训练集数据进行合并，然后进行数据处理后再对从 MIA 到 LAX 和从 MIA 到 JFK 这两组航班数据进行提取。①合并，具体代码如下：

```
#函数名：hb()
#作用：分别将 2001-2003 和 2004-2005 的数据合并为一个 csv 文件
defhb():
  list1_3=[]
  list4_5=[]
  foriinrange(1,4):
    df=pd.read_csv("./数据集/200{0}.csv".format(str(i)),encoding='unicode_escape')#读取文件
    list1_3.append(df)#将数据添加到数组中
  forxinrange(4,6):
    df1=pd.read_csv("./数据集/200{0}.csv".format(str(x)),encoding='unicode_escape')#读取文件
    list4_5.append(df1)#将数据添加到数组中
  result=pd.concat(list1_3)#合并文件
  result1=pd.concat(list4_5)#合并文件
  result.to_csv('2001_3.csv',index=False,encoding='unicode_escape')#保存合并文件
  result1.to_csv('2004_5.csv',index=False,encoding='unicode_escape')#保存合并文件
```

②合并完数据后，我们还对数据进行了数据转换、数据融合以及数据清洗，同上文

第一问类似。具体代码如下：

```
wheather.columns=['Year','Month','Day','最高气温','平均气温','最低气温','最高露点','平均露点','最低露点','最大湿度','平均湿度','最小湿度','最高海平面气压','平均海平面气压','最低海平面气压','最高能见度','平均能见度','最低能见度','最大风速','平均风速','瞬时风速','降水量','云量','活动','风向','机场','机场所在城市']#列名
wheather['日期']=pd.to_datetime(wheather[['Year','Month','Day']])#生成日期
wheather=wheather.drop(columns=['Month','Day','最高气温','平均气温','最低气温','最高露点','平均露点','最低露点','最大湿度','平均湿度','最小湿度','最高海平面气压','平均海平面气压','最低海平面气压','最高能见度','平均能见度','最大风速','机场所在城市','平均风速','瞬时风速','降水量','云量','活动','风向',])#删除不需要的列
```

同时在下面的 data_process 函数中还有以下处理：

```
flight_data['Day']=flight_data['DayofMonth']
flight_data['日期']=pd.to_datetime(flight_data[['Year','Month','Day']])
#删除年月日和不需要的数据数据，节约内存
clean_data=flight_data.drop(columns=['Year','Month','DayofMonth','DayOfWeek','FlightNum','CRSDepTime','Diverted','LateAircraftDelay','UniqueCarrier','Day','CRSArrTime','FlightNum','TailNum','ActualElapsedTime','AirTime','Distance','TaxiIn','TaxiOut','CancellationCode','SecurityDelay','NASDelay','WeatherDelay','CarrierDelay'])
#修改列名
clean_data.rename(columns={'Cancelled':'航班是否取消','Origin':'出发机场','Dest':'到达机场','DepDelay':'实际起飞与预计起飞之差','ArrDelay':'实际到达与预计到达之差','TaxiIn':'起飞滑行','TaxiOut':'降落滑行','DepTime':'实际起飞','CRSElapsedTime':'预计到达与预计起飞之差','ArrTime':'实际到达'},inplace=True)
```

③对从 MIA 到 LAX 和从 MIA 到 JFK 这两组航班数据进行提取。具体代码如下：

```
clean_data=clean_data.loc[clean_data['出发机场']=='MIA']
#从数据中筛选出出发机场为 MIA，航班没有取消，到达机场为 JFK 和 LAX 的数据
clean_data=clean_data.loc[clean_data['航班是否取消']==0]
clean_data_J=clean_data.loc[clean_data['到达机场']=='JFK']
clean_data_L=clean_data.loc[clean_data['到达机场']=='LAX']
L_J.append(clean_data_J)#将数据添加到数组中
```

```
L_J.append(clean_data_L)
result=pd.concat(L_J)#合并文件
```

④另外，针对缺失值和异常值，我们也进行了处理。由于数据量较大，有缺失值的行采取了直接过滤的方法。异常值的话，则将不是数据的值转换为 NAN。具体代码如下：

```
result['实际起飞']=pd.to_numeric(result['实际起飞'],'coerce')#将此列中不是数字的值转换为 NAN
result['实际到达']=pd.to_numeric(result['实际到达'],'coerce')
result['预计到达与预计起飞之差']=pd.to_numeric(result['预计到达与预计起飞之差'],'coerce')
result['实际到达与预计到达之差']=pd.to_numeric(result['实际到达与预计到达之差'],'coerce')
result['实际起飞与预计起飞之差']=pd.to_numeric(result['实际起飞与预计起飞之差'],'coerce')
result=result[~result['实际起飞'].isnull()]#过滤掉缺失值的行
result=result[~result['实际到达'].isnull()]
result=result[~result['预计到达与预计起飞之差'].isnull()]
result=result[~result['实际到达与预计到达之差'].isnull()]
result=result[~result['实际起飞与预计起飞之差'].isnull()]
```

5.3 问题二（1）的数据分析

首先，我们先调用 pandas、numpy、matplotlib、sklearn 库以及附件给出的机场数据和天气数据，具体代码见附录III。

然后我们创建了一个用于返回天气数据的函数 wheather _process，当 train_date 为真时，返回训练集天气数据，将 2001-2003 年天气数据合并，并选择出 MIA 机场的数据。当 test_date 为真时，则返回测试集。具体代码如下：

```
#函数名：wheather _process
#参数说明：
#两者只能一个为真
#train_date:为真时，返回训练集天气数据
#test_date：为真时，返回测试集天气数据
#作用：返回天气数据
def wheather_process(train_date=None,test_date=None):
  wheather2001_3=[]#2001-2003 年天气数据
```

```
  wheather2004_5=[]#2004-2005 年天气数据
  if train_date:
      wheather2001=wheather.loc[wheather['Year']==2001]#按条件选择数据
      wheather2002=wheather.loc[wheather['Year']==2002]
      wheather2003=wheather.loc[wheather['Year']==2003]
      wheather2001_3.append(wheather2001)#将数据添加到数组中
      wheather2001_3.append(wheather2002)
      wheather2001_3.append(wheather2003)
      wheather2001_3=pd.concat(wheather2001_3)#合并文件
      train_wheather=wheather2001_3.loc[wheather2001_3['机场']=='MIA']#按条件选择数
据
      returntrain_wheather
  if test_date:
      wheather2004=wheather.loc[wheather['Year']==2004]#按条件选择数据
      wheather2005=wheather.loc[wheather['Year']==2005]
      wheather2004_5.append(wheather2004)
      wheather2004_5.append(wheather2005)
      wheather2004_5=pd.concat(wheather2004_5)#合并文件
      test_wheather=wheather2004_5.loc[wheather2004_5['机场']=='MIA']#按条件选择数据
      returntest_wheather
```

接着，我们又创建了 data_process 函数用于返回特征值和目标值，从而形成特征集和目标集。将数据分为了测试集与训练集和以实际到达与预计到达之差视为延误与实际起飞与预计起飞之差视为延误。具体代码如下：

```
#函数名：data_process
#参数说明：
#两者只能一个为真
#train_date:为真时，返回训练集数据
#test_date：为真时，返回测试集数据
#两者只能一个为真
#arrive：为真时，以实际到达与预计到达之差统计目标数据
#take_off：为真时，以实际起飞与预计起飞之差统计目标数据
#作用：返回特征值和目标值
def data_process(train_date=None,test_date=None,arrive=None,take_off=None):
  target=[]#目标数据
```

```
L_J=[]
if train_date:#按条件选择数据
    flight_data=pd.read_csv('./2001_3.csv',encoding='unicode_escape',low_memory=False)
if test_date:
    flight_data=pd.read_csv('./2004_5.csv',encoding='unicode_escape',low_memory=False)
```

在这里我们分别以实际到达与预计到达之差和实际起飞与预计起飞之差统计目标数据，在大于 15 的数组中添加一个目标值 1（延误），在小于等于 15 的数组中添加一个目标值 0（未延误）。

```
#以实际到达与预计到达之差统计目标数据
if arrive:
  for i in range(0,len(result)):
    if result.iat[i,3]>15:
      target.append(1)#将数据添加到数组中
    elif result.iat[i,3]<=15:
      target.append(0)#将数据添加到数组中
#以实际起飞与预计起飞之差统计目标数据
  if take_off:
    for i in range(0,len(result)):
      if result.iat[i,4]>15:
        target.append(1)#将数据添加到数组中
    elif result.iat[i,4]<=15:
      target.append(0)#将数据添加到数组中
```

下面我们将将航班信息表与天气表用日期连接了起来，具体代码如下：

```
if train_date:
  result = pd.merge(result, wheather_process(train_date=1), on='日期')
if test_date:
  result = pd.merge(result, wheather_process(test_date=1), on='日期')
```

最后还需要进行数据类型转换，防止在后面的步骤出错。具体代码如下：

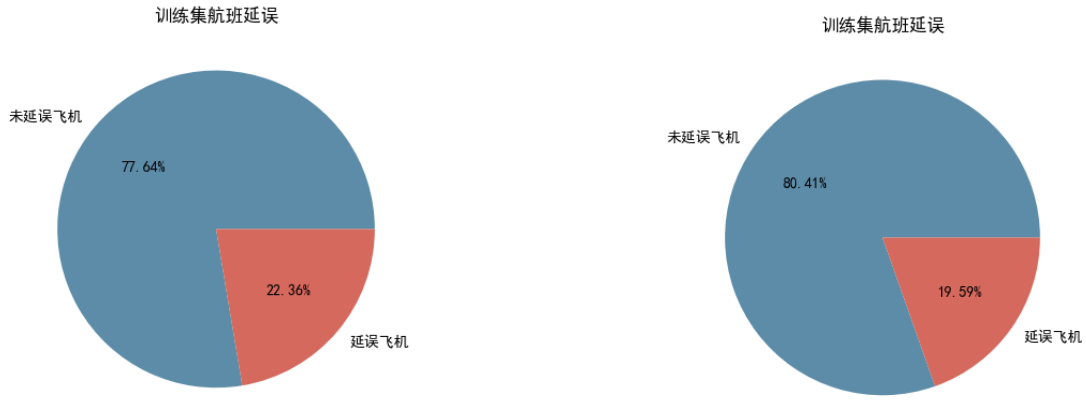
```
result['实际起飞']=result['实际起飞'].astype('int64')#数据类型转换
result['实际到达']=result['实际到达'].astype('int64')
result['预计到达与预计起飞之差']=result['预计到达与预计起飞之差'].astype('int64')
result['实际到达与预计到达之差']=result['实际到达与预计到达之差'].astype('int64')
result['实际起飞与预计起飞之差']=result['实际起飞与预计起飞之差'].astype('int64')
result['最低能见度']=result['最低能见度'].astype('int64')
```

```
train=result.values#将 DataFrame 类型的数据转换成 array 类型
target=np.array(target)#将数组类型转换成 array 类型
returntrain,target
```

5.4 问题二（1）数据建模

5.4.1 训练集航班是否延误对比

在数据分析结束以后，我们还得到航班是否延误数据统计对比，以饼状图的形式输出。如下图：



（a）实际到达与预计到达之差为航班延误　　（b）实际起飞与预计起飞之差为航班延误

图 7 训练集航班延误与未延误占比饼状图

5.4.2 Logistic 回归分析模型介绍

logistic 回归分析模型，是一种广义的线性回归分析模型，常用于二分类问题的数据挖掘，疾病自动诊断，经济预测等领域。例如，探讨引发疾病的危险因素，并根据危险因素预测疾病发生的概率等。以胃癌病情分析为例，选择两组人群，一组是胃癌组，一组是非胃癌组，两组人群必定具有不同的体征与生活方式等。因此因变量就为是否胃癌，值为“是”或“否”，自变量就可以包括很多了，如年龄、性别、饮食习惯、幽门螺杆菌感染等。自变量既可以是连续的，也可以是分类的。然后通过 logistic 回归分析，可以得到自变量的权重，从而可以大致了解到底哪些因素是胃癌的危险因素。同时根据该权值可以根据危险因素预测一个人患癌症的可能性。刚好使用于本题。

5.4.3 基于 Logistic 回归分析模型的预测

我们使用 Logistic 回归分析模型来实现预测。在上文中，我们对数据进行了预处理与分析，最终分别得到了训练集与测试集的特征集和目标集。现在首先建立 Logistic 回归分析模型，然后将训练集的特征集和目标集传输到模型中，对模型进行了 1000 次训练。模型训练完成后，再分别将训练集与测试集的特征集传输到模型中来预测结果，得

到预测结果后再与其的目标集进行比较，从而得到训练集与测试集的分数，进一步计算出准确率。具体代码如下：

```
def delay_model(arrive=None, take_off=None):
    data_train, data_target = data_process(train_date=1, arrive=arrive, take_off=take_off)  # 获取训练集数据
    test_train, test_target = data_process(test_date=1, arrive=arrive, take_off=take_off)  # 获取训练集数据
    pie(data_target)  # 生成延误航班饼形图
    model = LR(max_iter=1000)  # 创建模型,max_iter 为训练次数
    model.fit(data_train, data_target)  # 训练模型
    target_pred = model.predict(test_train)  # 预测结果
    target_pred_data = model.predict(data_train)
    print("航班训练集准确率为 %2.3f" % accuracy_score(data_target, target_pred_data))
    print("航班测试集准确率为 %2.3f" % accuracy_score(test_target, target_pred))
    print('航班延误训练集的分数:{0}'.format(model.score(data_train, data_target)))  # 精度
    print('航班延误测试集的分数:{0}'.format(model.score(test_train, test_target)))
    print('航班延误分类报告:')
    print(classification_report(test_target, target_pred))
```

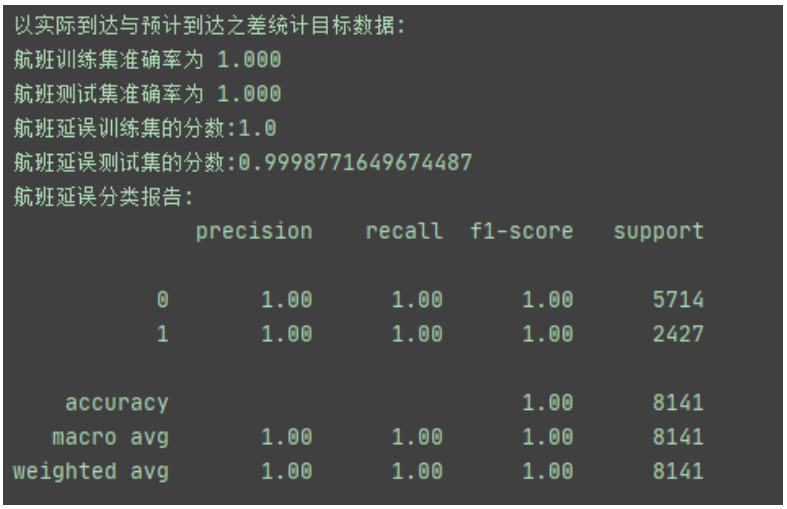
具体结果如下图：

```
以实际起飞与预计起飞之差统计目标数据
航班训练集准确率为 1.000
航班测试集准确率为 1.000
航班延误训练集的分数:1.0
航班延误测试集的分数:0.9997543299348974
航班延误分类报告:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5961
           1       1.00      1.00      1.00      2180

    accuracy                           1.00      8141
   macro avg       1.00      1.00      1.00      8141
weighted avg       1.00      1.00      1.00      8141
```

（a）以实际起飞与预计起飞之差为航班延误的预测准确率



（b）以实际到达与预计到达之差为航班延误的预测准确率

图 8 航班延误预测准确率

航班延误的分数为决定系数，决定系数反应了 y 的波动有多少百分比能被 x 的波动所描述，即表征依变数 Y 的变异中有多少百分比可由控制的自变数 X 来解释。即目标集被预测结果描述的百分比。由图可得，以实际起飞与预计起飞之差为航班延误的测试集分数约为 0.9997，以实际到达与预计到达之差为航班延误的测试集分数约为 0.9998，均表示目标集被测试集预测结果基本完全描述，测试集准确率也都为 1。

另外，分类报告中各参数含义如下：

①Precision，为精确度，表示在所有预测结果为 1 的样例数中，实际为 1 的样例数所占比重。精确度越低，意味着 01 比重很大，则代表模型对多数类 0 误判率越高，误伤了过多的多数类。为了避免对多数类的误伤，需要追求高精确度。

②Recall，为召回率，表示所有真实为 1 的样本中，被我们预测正确的样本所占的比例。召回率越高，代表我们尽量捕捉出了越多的少数类。

③为了同时兼顾精确度和召回率，我们创造了两者的调和平均数作为考量两者平衡的综合性指标，称之为 F1-score。两个数之间的调和平均倾向于靠近两个数中比较小的那一个数，因此我们追求尽量高的 F1-score，能够保证我们的精确度和召回率都比较高。F1-score 在[0,1]之间分布，越接近 1 越好。

④Support，支持度，是指原始的真实数据中属于该类的个数。

⑤accuracy，准确率，准确率(accuracy)即全部样本里被分类正确的比例。

⑥macro avg =上面类别各分数的直接平均。

⑦weighted avg=上面类别各分数的加权(权值为 support)平均。

由以上可得，分类报告中各数据参数也都为 1，表示本次预测都比较准确，预测结果理想。

5.5 问题二（2）数据预处理

5.5.1 数据介绍

这一小问所使用数据与（1）相同，只是特征值和目标值的选取略有变化。

5.5.2 特征值和目标值的选取

目标值为航班是否延误，特征值为航空公司延误、天气延误、国家航空系统延误、安全延误、晚飞延误。

5.5.3 数据处理

这一小问所进行的数据处理也与前文类似，只是把特征值和目标值进行替换，做法与步骤基本相同，主要体现在 delayreason_data 函数中。具体代码见附录III，在这里就不再具体展示。

5.6 问题二（2）数据分析

对于问题二（1）与问题二（2），代码是放在一起编写的。所以它们数据分析的方法与步骤是一致的，只是把特征值和目标值进行替换，两小问分别针对各自的特征值和目标值进行分析。

5.7 问题二（2）数据建模

5.7.1 训练集航班延误原因时长对比

在数据分析结束以后，我们还得到各航班延误原因的延误总时长数据，其中因航空公司延误时长为 13101 分钟，因天气延误时长为 6602 分钟，因国家航空系统延误时长为 7088 分钟，因安全延误时长为 121 分钟，因晚飞延误时长为 5244 分钟。经过对比，以饼状图的形式输出。如下图：

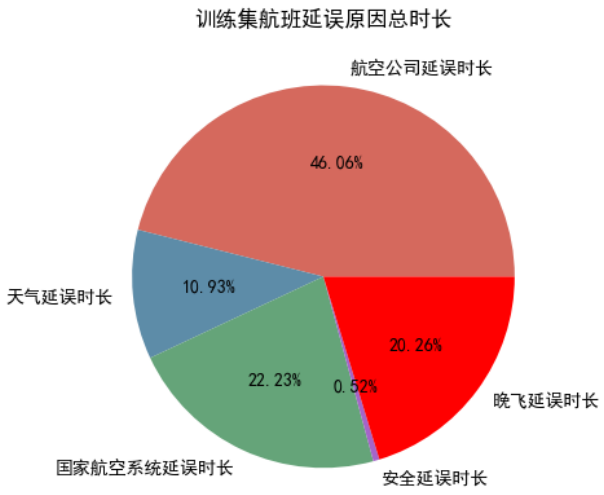


图 9 训练集航班延误原因总时长占比图

5.7.2 基于 Logistic 回归分析模型的预测

在这一问中，我们仍然使用 Logistic 回归分析模型来实现预测，做法与前文相同。具体代码及结果如下：

```
def delayreason_model(arrive=None, take_off=None):
    data_train, data_target = delayreason_data(train_date=1, arrive=arrive, take_off=take_off) # 获取训练集数据
    test_train, test_target = delayreason_data(test_date=1, arrive=arrive, take_off=take_off)  # 获取训练集数据
    model = LR(max_iter=1000)  # 创建模型,max_iter 为训练次数
    model.fit(data_train, data_target)  # 训练模型
    target_pred = model.predict(test_train)  # 预测结果
    target_pred_data = model.predict(data_train)
    print("航班原因训练集准确率为 %2.3f" % accuracy_score(data_target, target_pred_data))
    print("航班原因测试集准确率为 %2.3f" % accuracy_score(test_target, target_pred))
    print('航班延误原因训练集的分数:{0}'.format(model.score(data_train, data_target)))  # 精度
    print('航班延误原因测试集的分数:{0}'.format(model.score(test_train, test_target)))
    print('航班延误原因分类报告:')
    print(classification_report(test_target, target_pred))
```

```
航班原因训练集准确率为 1.000
航班原因测试集准确率为 0.998
航班延误原因训练集的分数:1.0
航班延误原因测试集的分数:0.9984031445768333
航班延误原因分类报告:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5961
           1       1.00      0.99      1.00      2180

    accuracy                           1.00      8141
   macro avg       1.00      1.00      1.00      8141
weighted avg       1.00      1.00      1.00      8141
```

（a）以实际起飞与预计起飞之差为航班延误的延误原因预测准确率

```
航班原因训练集准确率为 1.000
航班原因测试集准确率为 1.000
航班延误原因训练集的分数:1.0
航班延误原因测试集的分数:1.0
航班延误原因分类报告:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5714
           1       1.00      1.00      1.00      2427

    accuracy                           1.00      8141
   macro avg       1.00      1.00      1.00      8141
weighted avg       1.00      1.00      1.00      8141
```

（b）以实际到达与预计到达之差为航班延误的延误原因预测准确率

图 10 训练集航班延误原因预测准确率

由图可得，以实际起飞与预计起飞之差为航班延误的延误原因测试集分数约为 0.9984，以实际到达与预计到达之差为航班延误的延误原因测试集分数为 1.0，均表示目标集被测试集预测结果基本完全描述，测试集准确率分别为 0.998 与 1。

分类报告中各数据参数也基本都为 1，表示本次预测都比较准确，预测结果较为理想。

六、 总结

随着科技的发展，乘坐飞机出行为人们的生活带来了极大的便利，航空交通管理影响着人们的工作和生活效率。同时这也给航天监控技术提出了更高的要求，给空中资源分配、流量管制带来了更大的压力。与此同时，航班延误频发已经成为阻碍航空业进一步发展的重要因素。因此分析与预测航空延误有助于提高资源的优化管理，提高生产生活效率，可以为乘客提供更优质的服务，对航天事业发展有着重要意义。

本文以实现最短时间航班转机功能、对航班延误以及延误原因的预测为目标，根据附件给出的数据集展开研究。首先是利用图论中的经典算法 Dijkstra 算法来实现最短时间航班转机功能。先通过数据融合、数据清洗、数据转换对原始数据进行预处理，然后再对数据展开一系列分析，通过调用 python 库，结合 Dijkstra 算法实现功能，最后使用该功能查询 2003 年 7 月 4 日出发 7 月 5 日到达，从 CVG 机场到 ANC 机场最短时间方案，确保转机功能可以实现。

还有利用 Logistic 回归分析模型来对航班延误以及延误原因进行预测。本文先明确航班延误的含义，分两种情况进行研究。对附件给定的数据进行数据预处理，展开数据分析，确定对航班延误预测影响的特征，进行数学建模后，对模型进行训练，再将测试集特征值带入训练完成的预测模型中，得到预测结果。训练过的模型可以预测出近 100%

的准确率，表现出了较高的预测精度，可以为航班延误的预测问题提供较为准确的参考。

最后由于现实中导致航班延误的原因更为复杂，本文在特征参数选择上还有待进一步优化，比如将飞机机型、空管控流、电子故障等潜在因素列入特征值中，未来希望能够融合量化更多的潜在因素，从而更趋近现实情况以提高预测准确性。还可以增加计算能力，使得基于更多历史数据的分析与预测成为可能。

参考文献

[1]丁建立，孙玥.基于 LightGBM 的航班延误多分类预测［J］.南京航空航天大学学报，2021，53（6）：847-854。

[2]郭沛亮.基于多维数据源的动态交通流信息平台研究[D].北京：北京交通大学,2010。

[3]刘小玲,李辉,郭治国.基于狄克斯特拉算法的车间动态生产能力评估与实现[J].微计算机信息,2006(12):96-98．CNKI。

[4]平凡的 L 同学.Dijkstra 算法原理.https://blog.csdn.net/yalishadaa/article/details/55827681.2022-7-29。

[5]付振宇.基于多维数据的终端区航班延误预测研究[D].中国民用航空飞行学院,2020.DOI:10.27722/d.cnki.gzgmh.2020.000077。

[6]中国民用航空局.民航航班正常统计办法（征求意见稿）[EB/OL].http://pic.carnoc.com/file/160318/16031804593067.pdf,2022-8-3。

[7]魏嵩.关于航班正常统计办法的思考——中外航班正常统计之比较[J].中国民用航空,2012(07):43-46。

附录

附录 I 使用软件

Pycharm

附录 II 第一题完整代码

```
import numpy as np
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
import matplotlib

# 画图时显示中文
matplotlib.rcParams['font.sans-serif'] = ['SimHei']
matplotlib.rcParams['axes.unicode_minus'] = False
```

```
# 机场数据
airport_data = pd.read_csv("./数据集/airports.csv", encoding='unicode_escape', low_memory=False)

# 天气数据
wheather = pd.read_csv("./数据集/rawweatherdata.csv", encoding='unicode_escape', low_memory=False)

# pd.set_option('display.max_columns', None) # 显示所有列
# pd.set_option('display.max_rows', None) # 显示所有行
plt.rcParams['savefig.dpi'] = 400 # 图片像素
plt.rcParams['figure.dpi'] = 400 # 分辨率

# 函数名：data_process
# 参数说明：
#  flight_data：需要清洗的数据
# 作用：数据清理和数据转换，删除不需要的数据，将英文列名替换为中文列名，方便阅读理解
def data_process(flight_data):
  flight_data['Day'] = flight_data['DayofMonth']
  flight_data['起飞日期'] = pd.to_datetime(flight_data[['Year','Month','Day']])
  # 删除年月日和不需要的数据数据，节约内存
  clean_data = flight_data.drop(columns=['Year', 'Month', 'DayofMonth', 'LateAircraftDelay',
                                         'SecurityDelay', 'NASDelay', 'WeatherDelay', 'CarrierDelay',
                                         'Distance', 'CRSElapsedTime', 'DayOfWeek', 'FlightNum'])
  # 修改列名
  clean_data.rename(columns={'Cancelled': '航班是否取消', 'TaxiIn': '飞机起飞时滑行时间', 'TaxiOut': '飞机降落时滑行时间', 'Origin': '出发机场', 'Dest': '到达机场', 'ActualElapsedTime': '实际到达时间与实际起飞时间之差', 'DepTime': '实际起飞时间', 'airport': '机场名称', 'UniqueCarrier': 'iata代码', 'AirTime': '空中飞行时间'}, inplace = True)

# 函数名：read_data
```

```
# 参数说明：
#  flight_year:要获取的数据年份
# 作用：获取当前年所有飞行数据
def read_data(flight_year):
    year = flight_year.split('-')[0]
    flight_data = pd.read_csv('./数据集/{0}.csv'.format(year), encoding='unicode_escape', low_memory=False)
    return flight_data

# 函数名：flightday_data
# 参数说明：
#  clean_data：清洗完成的数据
#  outset_time：出发日期
#  arrive_time：到达日期
# 作用：返回时间段内的所有飞行数据
def flightday_data(clean_data, outset_time, arrive_time):
    twoday_data = clean_data.loc[(clean_data['起飞日期'] == '{0}'.format(outset_time)) | (clean_data['起飞日期'] == '{0}'.format(arrive_time))][clean_data['航班是否取消'] == 0]
    return twoday_data

# 函数名：departure
# 参数说明：
#  city：城市名
#  state：州名
# 作用：输入地名返回机场缩写
def departure(city, state):
    num = []
    real_num = []
    airports = airport_data.loc[airport_data['state'] == state]
    airports = airports.loc[airport_data['city'] == city]
    # 获取当前城市所有机场缩写
    for i in range(0, len(airports)):
        if airports.iat[i, 0] not in twoday_data['出发机场'].values:
            if airports.iat[i, 0] not in twoday_data['到达机场'].values:
```

```
            print('{0}机场不存在'.format(airports.iat[i, 0]))
            h = airports[airports.iata == airports.iat[i, 0]].index.tolist()[0]
            num.append(h)
        else：
            print('{0}机场存在'.format(airports.iat[i, 0]))
            h = airports[airports.iata == airports.iat[i, 0]].index.tolist()[0]
            real_num.append(h)
    airports = airports.drop(num)
    airports_name = airports.iat[0, 0]
    return airports_name

# 函数名：shortest_path
# 参数说明：
#   twoday_data: 出发日期到到达日期段内所有数据
#   outset_name：出发机场
#   arrive_name：到达机场
# 作用：求出当前时间段内最短时间的路径并画图
def shortest_path(twoday_data, outset_name, arrive_name):
    FG = nx.from_pandas_edgelist(twoday_data, source='出发机场', target='到达机场',
edge_attr='实际到达时间与实际起飞时间之差')
    nx.draw_networkx(FG, with_labels=True, font_size=5, node_size=50, arrows=True)
    dijpath = nx.dijkstra_path(FG, source='{0}'.format(outset_name), target='{0}'.format(arrive
_name), weight='实际到达时间与实际起飞时间之差')
    print('从{0}到{1}的最短飞行时间转机方案为{2}'.format(outset_name, arrive_name,
dijpath))
    plt.title('{0}到{1}飞行路线有向图'.format(outset_time, arrive_time))
    plt.savefig('./{0}-{1}.png'.format(outset_name, arrive_name))
    plt.show()

if __name__ == '__main__':
    # name = departure('Wrangell', 'AK')
    outset_time = input('请输入出发时间(xxxx-x-x)：')
    arrive_time = input('请输入到达时间(xxxx-x-x)：')
    outset_state = input('请输入出发州：')
```

```
    outset_city = input('请输入出发城市：')
    arrive_state = input('请输入到达州：')
    arrive_city = input('请输入到达城市：')
    flight_data = read_data(outset_time)
    clean_data = data_process(flight_data)
    twoday_data = flightday_data(clean_data, outset_time, arrive_time)
    outset_name = departure(outset_city, outset_state)
    arrive_name = departure(arrive_city, arrive_state)
    shortest_path(twoday_data, outset_name, arrive_name)

# flight_data = pd.read_csv('./数据集/2001.csv', encoding='unicode_escape', low_memory=
False)
# flight_data1 = pd.read_csv('./数据集/2002.csv', encoding='unicode_escape', low_memory=
False)
# flight_data2 = pd.read_csv('./数据集/2003.csv', encoding='unicode_escape', low_memory=
False)
# print(flight_data.shape)
# print(flight_data1.shape)
# print(flight_data2.shape)
```

附录Ⅲ 第二题完整代码

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
from sklearn.linear_model import LogisticRegression as LR
from sklearn.metrics import classification_report
from sklearn.model_selection import cross_val_score

# 画图时显示中文
matplotlib.rcParams['font.sans-serif'] = ['SimHei']
matplotlib.rcParams['axes.unicode_minus'] = False

pd.set_option('display.max_columns', None)  # 显示所有列
```

```
pd.set_option('display.max_columns', None)  # 显示所有列

# 机场数据
airport_data = pd.read_csv("./数据集/airports.csv", encoding='unicode_escape', low_memory=False)  # 读取文件
# 天气数据
wheather = pd.read_csv("./数据集/rawweatherdata.csv", encoding='unicode_escape', low_memory=False)  # 读取文件

#函数名：hb()
#作用：分别将 2001-2003 和 2004-2005 的数据合并为一个 csv 文件
defhb():
  list1_3=[]
  list4_5=[]
  foriinrange(1,4):
    df=pd.read_csv("./数据集/200{0}.csv".format(str(i)),encoding='unicode_escape')#读取文件
    list1_3.append(df)#将数据添加到数组中
  forxinrange(4,6):
    df1=pd.read_csv("./数据集/200{0}.csv".format(str(x)),encoding='unicode_escape')#读取文件
    list4_5.append(df1)#将数据添加到数组中
  result=pd.concat(list1_3)#合并文件
  result1=pd.concat(list4_5)#合并文件
  result.to_csv('2001_3.csv',index=False,encoding='unicode_escape')#保存合并文件
  result1.to_csv('2004_5.csv',index=False,encoding='unicode_escape')#保存合并文件

wheather.columns=['Year','Month','Day','最高气温','平均气温','最低气温','最高露点','平均露点','最低露点','最大湿度','平均湿度','最小湿度','最高海平面气压','平均海平面气压','最低海平面气压','最高能见度','平均能见度','最低能见度','最大风速','平均风速','瞬时风速','降水量','云量','活动','风向','机场','机场所在城市']#列名
wheather['日期']=pd.to_datetime(wheather[['Year','Month','Day']])#生成日期
wheather=wheather.drop(columns=['Month','Day','最高气温','平均气温','最低气温','最高露
```

```
                点','平均露点','最低露点','最大湿度','平均湿度','最小湿度','最高海平面
                气压','平均海平面气压','最低海平面气压','最高能见度','平均能见度','
                最大风速','机场所在城市','平均风速','瞬时风速','降水量','云量','活动','
                风向',])#删除不需要的列


#函数名：wheather _process
#参数说明：
#两者只能一个为真
#train_date:为真时，返回训练集天气数据
#test_date：为真时，返回测试集天气数据
#作用：返回天气数据
def wheather_process(train_date=None,test_date=None):
  wheather2001_3=[]#2001-2003 年天气数据
  wheather2004_5=[]#2004-2005 年天气数据
  if train_date:
     wheather2001=wheather.loc[wheather['Year']==2001]#按条件选择数据
     wheather2002=wheather.loc[wheather['Year']==2002]
     wheather2003=wheather.loc[wheather['Year']==2003]
     wheather2001_3.append(wheather2001)#将数据添加到数组中
     wheather2001_3.append(wheather2002)
     wheather2001_3.append(wheather2003)
     wheather2001_3=pd.concat(wheather2001_3)#合并文件
     train_wheather=wheather2001_3.loc[wheather2001_3['机场']=='MIA']#按条件选择数
据
     returntrain_wheather
  if test_date:
     wheather2004=wheather.loc[wheather['Year']==2004]#按条件选择数据
     wheather2005=wheather.loc[wheather['Year']==2005]
     wheather2004_5.append(wheather2004)
     wheather2004_5.append(wheather2005)
     wheather2004_5=pd.concat(wheather2004_5)#合并文件
     test_wheather=wheather2004_5.loc[wheather2004_5['机场']=='MIA']#按条件选择数据
     returntest_wheather
```

```
#函数名：data_process
#参数说明：
#两者只能一个为真
#train_date:为真时，返回训练集数据
#test_date：为真时，返回测试集数据
#两者只能一个为真
#arrive：为真时，以实际到达与预计到达之差统计目标数据
#take_off：为真时，以实际起飞与预计起飞之差统计目标数据
#作用：返回特征值和目标值
def data_process(train_date=None,test_date=None,arrive=None,take_off=None):
  target=[]#目标数据
  L_J=[]
  if train_date:#按条件选择数据
     flight_data=pd.read_csv('./2001_3.csv',encoding='unicode_escape',low_memory=False)
  if test_date:
     flight_data=pd.read_csv('./2004_5.csv',encoding='unicode_escape',low_memory=False)

   flight_data['Day']=flight_data['DayofMonth']
   flight_data['日期']=pd.to_datetime(flight_data[['Year','Month','Day']])
   #删除年月日和不需要的数据数据，节约内存
   clean_data=flight_data.drop(columns=['Year','Month','DayofMonth','DayOfWeek','FlightNu
                  m','CRSDepTime','Diverted','LateAircraftDelay','UniqueCarrier','Day','CR
                  SArrTime','FlightNum','TailNum','ActualElapsedTime','AirTime','Distanc
                  e','TaxiIn','TaxiOut','CancellationCode','SecurityDelay','NASDelay','Weath
                  erDelay','CarrierDelay'])
   #修改列名
   clean_data.rename(columns={'Cancelled':'航班是否取消','Origin':'出发机场','Dest':'到达
                  机场','DepDelay':'实际起飞与预计起飞之差','ArrDelay':'实际到达与预
                  计到达之差','TaxiIn':'起飞滑行','TaxiOut':'降落滑行','DepTime':'实际起
                  飞','CRSElapsedTime':'预计到达与预计起飞之差','ArrTime':'实际到达
                  '},inplace=True)
   clean_data=clean_data.loc[clean_data['出发机场']=='MIA']
   #从数据中筛选出出发机场为 MIA，航班没有取消，到达机场为 JFK 和 LAX 的数据
   clean_data=clean_data.loc[clean_data['航班是否取消']==0]
```

```
clean_data_J=clean_data.loc[clean_data['到达机场']=='JFK']
clean_data_L=clean_data.loc[clean_data['到达机场']=='LAX']
L_J.append(clean_data_J)#将数据添加到数组中
L_J.append(clean_data_L)
result=pd.concat(L_J)#合并文件
result['实际起飞']=pd.to_numeric(result['实际起飞'],'coerce')#将此列中不是数字的值转换为 NAN
result['实际到达']=pd.to_numeric(result['实际到达'],'coerce')
result['预计到达与预计起飞之差']=pd.to_numeric(result['预计到达与预计起飞之差'],'coerce')
result['实际到达与预计到达之差']=pd.to_numeric(result['实际到达与预计到达之差'],'coerce')
result['实际起飞与预计起飞之差']=pd.to_numeric(result['实际起飞与预计起飞之差'],'coerce')
result=result[~result['实际起飞'].isnull()]#过滤掉缺失值的行
result=result[~result['实际到达'].isnull()]
result=result[~result['预计到达与预计起飞之差'].isnull()]
result=result[~result['实际到达与预计到达之差'].isnull()]
result=result[~result['实际起飞与预计起飞之差'].isnull()]
result = result.drop(columns=['出发机场', '到达机场', '航班是否取消', '日期', 'Year', '机场']) # 删除不需要的列
#以实际到达与预计到达之差统计目标数据
if arrive:
    for i in range(0,len(result)):
        if result.iat[i,3]>15:
            target.append(1)#将数据添加到数组中
        elif result.iat[i,3]<=15:
            target.append(0)#将数据添加到数组中
#以实际起飞与预计起飞之差统计目标数据
    if take_off:
        for i in range(0,len(result)):
            if result.iat[i,4]>15:
                target.append(1)#将数据添加到数组中
        elif result.iat[i,4]<=15:
```

```
            target.append(0)#将数据添加到数组中
    #将航班信息表与天气表用日期连接起来
    if train_date:
      result = pd.merge(result, wheather_process(train_date=1), on='日期')
    if test_date:
      result = pd.merge(result, wheather_process(test_date=1), on='日期')
    result['实际起飞']=result['实际起飞'].astype('int64')#数据类型转换
    result['实际到达']=result['实际到达'].astype('int64')
    result['预计到达与预计起飞之差']=result['预计到达与预计起飞之差'].astype('int64')
    result['实际到达与预计到达之差']=result['实际到达与预计到达之差'].astype('int64')
    result['实际起飞与预计起飞之差']=result['实际起飞与预计起飞之差'].astype('int64')
    result['最低能见度']=result['最低能见度'].astype('int64')
    train=result.values#将 DataFrame 类型的数据转换成 array 类型
    target=np.array(target)#将数组类型转换成 array 类型
    returntrain,target

def delayreason_data(train_date = None, test_date = None, arrive= None, take_off=None):
    target = [] #   目标数据
    L_J = []
    if train_date: #   按条件选择数据
      flight_data = pd.read_csv('./2001_3.csv', encoding='unicode_escape', low_memory=False)
    if test_date:
      flight_data = pd.read_csv('./2004_5.csv', encoding='unicode_escape', low_memory=False)
    clean_data = flight_data.drop(columns=['Year', 'Month', 'DayofMonth', 'DayOfWeek',
                    'FlightNum', 'CRSDepTime', 'Diverted', 'UniqueCarrier',
                    'ActualElapsedTime', 'DepTime', 'CRSElapsedTime','CRSArrTime',
                    'FlightNum', 'TailNum', 'ActualElapsedTime', 'AirTime', 'Distance',
                    'TaxiIn', 'TaxiOut', 'CancellationCode', 'ArrTime'])
    #修改列名
    clean_data.rename(columns={'Cancelled': '航班是否取消', 'Origin': '出发机场', 'Dest': '到
                    达机场', 'ArrDelay': '实际到达与预计到达之差', 'DepDelay': '实际起
                    飞与预计起飞之差', 'CarrierDelay': '航空公司延误时长', 'WeatherDelay':
                    '天气延误时长', 'NASDelay': '国家航空系统延误时长', 'SecurityDelay': '
                    安全延误时长', 'LateAircraftDelay': '晚飞延误时长'}, inplace = True)
```

```
clean_data = clean_data.loc[clean_data['出发机场'] == 'MIA']  # 从数据中筛选出出发机场为 MIA，航班没有取消，到达机场为 JFK 和 LAX 的数据
clean_data = clean_data.loc[clean_data['航班是否取消'] == 0]
clean_data_J = clean_data.loc[clean_data['到达机场'] == 'JFK']
clean_data_L = clean_data.loc[clean_data['到达机场'] == 'LAX']
L_J.append(clean_data_J)  # 将数据添加到数组中
L_J.append(clean_data_L)
result = pd.concat(L_J)  # 合并文件
result['航空公司延误时长'] = pd.to_numeric(result['航空公司延误时长'], 'coerce')  # 将此列中不是数字的值转换为 NAN
result['天气延误时长'] = pd.to_numeric(result['天气延误时长'], 'coerce')
result['国家航空系统延误时长'] = pd.to_numeric(result['国家航空系统延误时长'], 'coerce')
result['安全延误时长'] = pd.to_numeric(result['安全延误时长'], 'coerce')
result['晚飞延误时长'] = pd.to_numeric(result['晚飞延误时长'], 'coerce')
result['实际到达与预计到达之差'] = pd.to_numeric(result['实际到达与预计到达之差'], 'coerce')
result['实际起飞与预计起飞之差'] = pd.to_numeric(result['实际起飞与预计起飞之差'], 'coerce')
result = result[~result['航空公司延误时长'].isnull()]  # 过滤掉缺失值的行
result = result[~result['天气延误时长'].isnull()]
result = result[~result['国家航空系统延误时长'].isnull()]
result = result[~result['安全延误时长'].isnull()]
result = result[~result['晚飞延误时长'].isnull()]
result = result[~result['实际到达与预计到达之差'].isnull()]
result = result[~result['实际起飞与预计起飞之差'].isnull()]
result = result.drop(columns=['出发机场', '到达机场', '航班是否取消'])  # 删除不需要的列
# 以实际到达与预计到达之差统计目标数据
if arrive:
  for i in range(0, len(result)):
    if result.iat[i, 0] > 15:
      target.append(1)  # 将数据添加到数组中
    elif result.iat[i, 0] <= 15:
```

```
            target.append(0)  # 将数据添加到数组中
    # 以实际起飞与预计起飞之差统计目标数据
    if take_off:
      for i in range(0, len(result)):
          if result.iat[i, 1] > 15:
              target.append(1)  # 将数据添加到数组中
          elif result.iat[i, 1] <= 15:
              target.append(0)  # 将数据添加到数组中
    result['航空公司延误时长'] = result['航空公司延误时长'].astype('int64')  # 数据类型转换
    result['天气延误时长'] = result['天气延误时长'].astype('int64')
    result['国家航空系统延误时长'] = result['国家航空系统延误时长'].astype('int64')
    result['安全延误时长'] = result['安全延误时长'].astype('int64')
    result['晚飞延误时长'] = result['晚飞延误时长'].astype('int64')
    delay_time = result[['航空公司延误时长', '天气延误时长', '国家航空系统延误时长', '安全延误时长', '晚飞延误时长']].sum()  # 计算每种原因的延误总时长
    delay_time = np.array(delay_time)
    pie_reason(delay_time)
    train = result.values #将 DataFrame 类型的数据转换成 array 类型
    target = np.array(target) #将数组类型转换成 array 类型
    return train, target


def pie(array):
    x=[]
    for i in np.unique(array):
        x.append(np.sum(array==i))
    x=np.array(x)
    plt.pie(x,
            labels=['未延误飞机', '延误飞机'],  # 设置饼图标签
            colors=['#5d8ca8', "#d5695d"],  # 设置饼图颜色
            autopct='%.2f%%',  # 格式化输出百分比
            )
    plt.title("训练集航班延误")
    plt.savefig('训练集航班延误.png')
```

```
    plt.show()

def pie_reason(array):
    plt.pie(
        array,
        labels=['航空公司延误时长', '天气延误时长', '国家航空系统延误时长', '安全延误时长', '晚飞延误时长'],  # 设置饼图标签
        colors=["#d5695d", "#5d8ca8", "#65a479", "#a564c9", '#FF0000'],  # 设置饼图颜色
        autopct='%.2f%%',  # 格式化输出百分比
            )
    plt.title("训练集航班延误原因总时长")
    plt.savefig('训练集航班延误原因.png')
    plt.show()

def delay_model(arrive=None, take_off=None):
    data_train, data_target = data_process(train_date=1, arrive=arrive, take_off=take_off)  # 获取训练集数据
    test_train, test_target = data_process(test_date=1, arrive=arrive, take_off=take_off)  # 获取训练集数据
    pie(data_target)  # 生成延误航班饼形图
    model = LR(max_iter=1000)  # 创建模型,max_iter 为训练次数
    model.fit(data_train, data_target)  # 训练模型
    target_pred = model.predict(test_train)  # 预测结果
    target_pred_data = model.predict(data_train)
    print("航班训练集准确率为 %2.3f" % accuracy_score(data_target, target_pred_data))
    print("航班测试集准确率为 %2.3f" % accuracy_score(test_target, target_pred))
    print('航班延误训练集的分数:{0}'.format(model.score(data_train, data_target)))  # 精度
    print('航班延误测试集的分数:{0}'.format(model.score(test_train, test_target)))
    print('航班延误分类报告:')
    print(classification_report(test_target, target_pred))

def delayreason_model(arrive=None, take_off=None):
    data_train, data_target = delayreason_data(train_date=1, arrive=arrive, take_off=take_off)
# 获取训练集数据
```

```
    test_train, test_target = delayreason_data(test_date=1, arrive=arrive, take_off=take_off)  # 获取训练集数据
    model = LR(max_iter=1000)  # 创建模型,max_iter 为训练次数
    model.fit(data_train, data_target)  # 训练模型
    target_pred = model.predict(test_train)  # 预测结果
    target_pred_data = model.predict(data_train)
    print("航班原因训练集准确率为 %2.3f" % accuracy_score(data_target, target_pred_data))
    print("航班原因测试集准确率为 %2.3f" % accuracy_score(test_target, target_pred))
    print('航班延误原因训练集的分数:{0}'.format(model.score(data_train, data_target)))  # 精度
    print('航班延误原因测试集的分数:{0}'.format(model.score(test_train, test_target)))
    print('航班延误原因分类报告:')
    print(classification_report(test_target, target_pred))

if __name__ == '__main__':
#在训练模型前需先运行 hb()函数以生成 2001_3.csv 和 2004_5.csv 两个文件
    #hb()
    print('以实际到达与预计到达之差统计目标数据:')
    delay_model(arrive=1)
    delayreason_model(arrive=1)
    print('以实际起飞与预计起飞之差统计目标数据')
    delay_model(take_off=1)
    delayreason_model(take_off=1)
```