

|      |                        |           |
|------|------------------------|-----------|
| 所属类别 | 2024 年“华数杯”全国大学生数学建模竞赛 | 参赛编号      |
| 本科组  |                        | CM2404805 |

## 山水之间，一程光阴：优化外国游客在华路径

### 摘要

随着中国过境免签政策的实施和“city 不 city”网络流行语的传播，越来越多的外国游客通过网络平台展示他们在华的旅行经历，不仅促进了中国旅游业的发展，还在国际舞台上展示了中国的真实而生动的形象。本研究通过数据处理、统计分析和路径优化等多种数学建模技术，为外国游客在中国的旅行提供了最佳解决方案。

针对问题一，我们的主要目标是对城市中的景点数据进行预处理，并选择每个城市中评分最高的景点作为游览目标。为了实现这一目标，我们首先从大型数据集中提取并清洗景点数据，包括评分、名称和位置等信息。在数据预处理阶段，我们排除了评分不完整或缺失的记录，并对数据进行了标准化处理，以便进行后续分析。接着，利用统计方法计算出每个城市中最高评分的景点，并进行了景点的数据整合。此外，我们还使用了 Bayesian Networks-PCA（贝叶斯网络和主成分分析）技术，对城市和景点的吸引力进行了多维度评价和分析，这不仅帮助我们理解了各景点在游客中的受欢迎程度，还提供了数据驱动的支持，为最终的游览路径选择提供了科学依据。

我们的任务是对 352 个中国城市进行综合评价，以选出最能吸引外国游客的 50 个城市。这一过程涉及到广泛的数据收集，包括城市规模、环境保护、人文底蕴、交通便利性、气候以及美食等多个维度。我们首先从各个权威网站和数据库爬取了这些指标的数据。接着，对这些复杂的数据进行了归一化处理，以确保各指标之间的比较具有可比性。为了综合评价这些城市，我们采用了层次分析法（AHP）和基于熵权法的技术排序分析系统（TOPSIS）方法。这两种方法结合使用，不仅可以准确地衡量各指标的重要性，还能合理地计算出城市的综合吸引力评分。通过这些方法，我们最终确定了 50 个最令外国游客向往的城市，并进行了详细的排序和可视化展示，这些城市在旅游资源、文化深度以及访问便利性方面表现出色，能够提供给游客多样化而独特的旅行体验。

针对问题三，我们设计了一条在 144 小时内尽可能多地游览城市的高效旅游路线，通过启发式算法优化，兼顾了游览时间和城市间高铁旅行的成本效率。

针对问题四，我们进一步优化了旅游路线，尤其是降低了门票和交通的总费用，以确保游览成本在可接受范围内。

针对问题五，专门为喜欢山景的外国游客定制了一个旅游路线，考虑了山景的吸引力、门票成本和交通便利性，制定了成本效率最高的旅行计划。

通过这些精心设计的方案，我们不仅优化了外国游客在中国的旅游体验，还展示了中国丰富的自然和文化景观。

**关键词：**Bayesian Networks-PCA；数据处理；路径优化；外国游客

# 一、问题重述

## 1.1 问题背景

近年来，随着中国旅游业的迅猛发展和国际影响力的提升，外国游客对中国的旅游兴趣日益浓厚。特别是“city 不 city”这一网络流行语的传播，使得越来越多的外国游客通过网络平台展示他们在中国的旅行见闻，从而推动了中国旅游业的发展，并在国际舞台上展示了一个真实、生动的中国。与此同时，中国的过境免签政策也极大地便利了外国游客的旅行安排。

基于此背景，研究提出了一系列问题，旨在通过建立数学模型，系统性地评估和优化外国游客在中国的旅游体验。这些问题包括如下图 1：



图 1 需要解决的问题

## 1.2 文献综述

近年来，城市旅游吸引力的评价成为学术界和实务界的研究热点。研究表明，城市的旅游吸引力主要由多种因素共同决定，包括自然景观、文化底蕴、基础设施、环境质量以及社会经济条件等。这些因素共同作用，影响了游客对城市的整体感知和选择偏好。

城市的自然和文化资源是吸引游客的主要因素之一。例如，历史古迹、公园和博物馆等对文化旅游具有重要吸引力。优质的旅游基础设施，如便捷的交通网络、丰富的住宿选择和高效的公共服务，显著提升了城市的旅游吸引力。研究指出，良好的环境质量，包括空气和水的清洁度、城市绿化水平等，是现代游客非常重视的因素。环保政策和实际的环境状况直接影响游客的体验和城市形象<sup>[1][2]</sup>。此外，城市的经济发展水平、社会治安状况以及当地居民的友好程度也是游客考虑的重要方面。这些因素共同决定了一个城市的总体吸引力和竞争力。为了更全面地评估城市旅游吸引力，研究通常采用多维度评价模型。这些模型包括数据收集与整理、因子分析、权重分配和综合评分与排序等步骤。通过因子分析技术，研究可以将复杂的评价指标简化为若干关键因素，从而更清晰地理解每个城市的旅游吸引力构成<sup>[3]</sup>。

中国旅游研究院发布的《2021 世界旅游城市蓝皮书》指出，世界级的旅游城市应以游客满意为导向，通过文化提升品质和科技形塑未来。一座优秀的旅游城市，通常具有完善的基础设施、高效的公共服务和优质的人文环境，从而构建出一个主客共享的美好生活新空间<sup>[5]</sup>。此外，《中国城市休闲和旅游竞争力报告（2020）》也强调，提升城市

的休闲和旅游供给对于提高城市的吸引力和竞争力至关重要。报告通过对多个城市的综合竞争力进行评估，提出了提升城市旅游吸引力的策略和建议<sup>[6]</sup>。在具体评价城市旅游吸引力时，研究通常采用多维度评价模型。这些模型包括数据收集与整理、因子分析、权重分配和综合评分与排序等步骤。通过因子分析技术，研究可以将复杂的评价指标简化为若干关键因素，从而更清晰地理解每个城市的旅游吸引力构成。例如，墨西哥城、布宜诺斯艾利斯、波哥大和利马的研究发现，游客对这些城市的吸引力感知主要集中在核心景点、旅游生态系统、会议和展览以及相关服务等方面<sup>[7]</sup>。此外，近年来中国在提升外国游客入境游便利化方面也采取了多项措施。例如，优化支付便利性、改进景区预约购票程序和提供多语种文旅服务等。广州市推出的免费一日游服务项目，旨在通过免费旅游巴士、英文导游、景点参观等活动，让国际中转旅客在有限时间内增加对广州的了解<sup>[7]</sup>。

多个研究实例表明，不同城市在旅游吸引力方面存在显著差异。例如，我们从国外的一些文献查阅得知对拉丁美洲四大城市（墨西哥城、布宜诺斯艾利斯、波哥大和利马）的研究发现，游客对这些城市的吸引力感知主要集中在四个方面：核心景点、旅游生态系统、会议和展览以及相关服务<sup>[4]</sup>。这些研究结果为其他地区城市的旅游吸引力评价提供了参考。通过科学的评价模型和方法，可以帮助决策者和旅游管理者制定更有效的提升策略，从而吸引更多的游客，促进城市的旅游业发展和经济增长。

总之，综合评价城市旅游吸引力需要考虑多方面因素。研究表明，自然和文化资源、基础设施和服务、环境质量以及社会经济条件等都是决定城市旅游吸引力的重要因素。通过科学的评价模型和方法，可以帮助城市更好地提升其旅游吸引力，吸引更多的游客，从而促进城市的经济发展和社会进步。

## 周铠然 · 仅供求职展示 二、问题分析

### 2.1 问题一的分析

在问题一中，我们将深入分析旅游景点评分数据，重点关注最高评分及其分布情况。问题一将分为四个主要部分，数据集中的最高评分（BS）、获得最高评分的景点数量、拥有最高评分景点最多的城市，以及按最高评分景点数量排序的前 10 个城市。

对于数据集中的最高评分（BS），首先要确定数据集中所有景点的最高评分。这一过程包括读取每个城市的评分数据，并找出所有景点中的最高评分。最高评分的确定是通过遍历所有 CSV 文件，对每个文件中的“评分”列进行分析，找出每个文件的最高评分，并更新全局最高评分，并得到数据集中评分的极值，为后续的分析提供基础。

对于获得最高评分的景点数量。在确定了最高评分之后，计算获得这一最高评分的景点数量，统计每个城市中评分等于最高评分的景点数目。具体来说，我们将再次遍历所有 CSV 文件，对每个文件中的“评分”列进行筛选，统计出评分等于最高评分的景点数量，并将这些数量进行累加，从而得到所有城市中获得最高评分的景点总数。这一分析有助于我们了解最高评分景点的普遍性以及其在数据集中的分布情况。

对于拥有最高评分景点最多的城市。在统计了最高评分景点的数量之后，我们进一步分析哪些城市拥有最多的最高评分景点。这要统计每个城市的最高评分景点数量，还需将这些数据汇总，以找出拥有最多最高评分景点的城市。通过对比各城市的最高评分景点数量，识别出旅游评分最为优异的城市。这一分析对于旅游管理和市场推广具有重要意义，因为它可以帮助我们了解哪些城市在游客心目中最受欢迎。

对于按最高评分景点数量排序的前 10 个城市。我们对所有城市按照最高评分景点的数量进行排序，并列出前 10 个城市。这一部分展示了哪些城市拥有最多的最高评分景点，还通过排序展示城市在数据集中所处的位置。通过这种排序，我们能够清晰地看到哪些城市在旅游景点评分上表现突出，并为后续的旅游资源配置和推广策略提供数据支持。

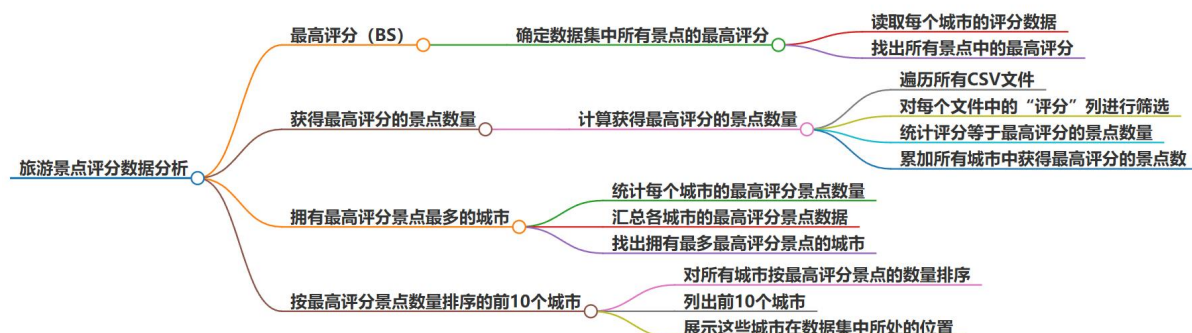


图 2 问题 2 解题思路分析

## 2.2 问题二的分析

为对 352 个中国城市进行综合评价并选出最令外国游客向往的 50 个城市，我们首先从各大网站爬取相关的指标评分，包括城市规模、环境保护、人文底蕴、交通便利、气候和美食等。接着对这些数据进行标准化处理，确保可比性。然后，根据每个指标的重要性分配权重，进行加权求和，得到综合评分。最后，根据综合评分进行排序，选出前 50 个城市。此过程确保评价结果的公正性和可靠性。

## 2.3 问题三的分析

为了规划外国游客在 144 小时内从广州出发游览尽可能多的城市并确保综合游玩体验最佳，我们首先筛选出“最令外国游客向往的 50 个城市”。每个城市只游玩评分最高的景点，并且只能选择高铁作为交通方式。具体方法包括收集每个城市的最高评分景点及其相关信息，如游玩时间和门票费用，获取城市之间的高铁时刻和费用数据。接着，计算每个城市的游玩时间和城市间的高铁时间，确保总时间不超过 144 小时。同时，综合考虑门票和交通费用，确保整体费用在合理范围内。最后，使用旅行商问题算法规划出最佳路线，使得在有限时间内游览最多的城市并且花费最少。这样既能提供详细的游玩路线，包括总花费时间、费用及可游玩的景点数量，又能确保游客的综合游玩体验最佳。这些措施基于最新的旅游信息和热门城市的吸引力，有效地展示了中国丰富的旅游资源和文化魅力。

## 2.4 问题四的分析

问题四的分析涉及对一个外国游客在 144 小时内通过高铁游览中国尽可能多城市的路线进行优化，同时确保门票和交通总费用尽可能低。这需要在考虑旅游体验的基础上，用数学模型找到一个既经济又高效的游览路线。核心挑战在于如何在限定时间内最大化城市数量的访问，同时控制相关成本。模型会利用城市间的高铁连接数据，结合每个城市的旅游景点门票价格和推荐游玩时间，通过算法计算出最佳路线。这可能包括设定起始城市、计算城市间的旅行时间和成本，以及在给定的预算和时间限制下，使用启发式

算法来优化路径选择。

## 2.5 问题五的分析

问题五则专注于为一名只想游览中国山景的外国游客定制 144 小时的旅游路线，目标是游览尽可能多的评分高的山景，同时保持门票和交通成本最低。问题中涉及到的数据处理包括城市与山景的对应信息，如每个城市的山景评分和建议游玩时间。旅游路线的规划将考虑城市间的交通方式（限定为高铁），并结合每个山景的门票价格和游玩时间，通过数学模型计算出最佳访问序列。这需要广泛的数据分析，包括距离计算、成本评估和路径优化，确保在既定时间内以最低成本访问最多的高评分山景。

# 三、模型的建立与求解

## 3.1 问题一

### 3.1.1 准备工作

在数据处理与分析过程中，我们首先对包含 352 个城市的每 100 个景点数据的数据集进行了初步浏览和理解，以确保对每个字段的含义有清晰的认识。数据包含多个列，包括景点名称、链接、地址、介绍、开放时间、图片链接、评分、建议游玩时间、建议季节、门票、小贴士和页面编号。评分是我们分析的重点之一，但数据还包含景点的详细信息，如地址、开放时间、建议游玩时间和门票价格等。

我们进行了数据清洗，包括处理缺失值和异常值，确保评分列为数值型，并将无法解析的数据设为缺失值。接着，进行了数据转换，将非数值型数据（如游玩季节）转换为数值型，确保所有参与分析的字段均为正确的数值类型。最后，通过编写代码遍历所有 CSV 文件，读取并整合数据，完成数据预处理。

### 3.1.2 模型建立与求解

针对问题一，我们需要首先对数据进行预处理。由于我们是通过评分来计算城市和景点数据量，因此第一步是删除评分为空值和包含“--”的数据，这些数据可能是由于爬虫未能成功爬取所致。在删除这些噪声数据之后，我们将所有数据进行整合，从总体的 CSV 文件中提取景点的评分字段和所在城市。接下来，我们计算 33000 个景点中的最高评分（BS），并统计获得最高评分的景点数量，同时找出这些景点所在的城市。最后，我们按照拥有 BS 景点数量的多少进行排序，列出前 10 个城市，并将结果进行可视化展示。



图 3 问题一数据处理过程

### ● 任务 1：确定最高评分（BS）

数据预处理之后，我们需要找出所有景点评分的最高分（BS）以及景点数量最多的前十个城市。首先，建立一个空的 DataFrame 并遍历每个城市的数据，将评分字段转化成数值类型。然后，计算每个景点的最高评分（BS），统计获得最高评分的景点数量，并找出这些景点所在的城市。最后，按照拥有最高评分景点的数量进行排序，列出前 10 个城市，并对结果进行可视化展示。

访问并遍历包含景点数据的 CSV 文件。对于每个 CSV 文件，并将评分列转换为数值类型，排除非数值数据。代码查找每个文件中的最高评分，并与全局最高评分进行比较，更新最高评分值。最终，代码输出所有文件中发现的最高评分，是 5.0。

设  $S$  为文件夹中所有 CSV 文件的集合， $f$  为  $S$  中的一个文件。每个文件  $f$  包含一系列评分数据，记为  $P_f$ 。设  $n$  是该文件中评分的数量

则  $P_f$  中的评分表示：

$$P_f = \{p_{f1}, p_{f2}, \dots, p_{fn}\}$$

计算单个文件  $f$  中最高评分的公式：

$$\text{BestScore}(f) = \max(P_f)$$

整个文件夹的最高评分 BS 的公式：

$$BS = \max_{f \in S}(\text{BestScore}(f))$$

我们从每个文件  $f$  计算的最高评分  $\text{BestScore}(f)$  中取最大值，来得到整个文件夹中所有文件的最高评分。

初始化一个最高评分变量 `bestscore` 为 0，然后遍历每个文件，对每个文件计算 `current_max`，即 `BestScore(f)`，并更新 `best_score` 为：

$$\text{best\_score} = \max(\text{best\_score}, \text{current\_max})$$

### ● 任务 2：计算获得 BS 评分的景点总数

计算获得最高评分（已知为 5.0）的景点总数。首先初始化一个计数器 `'count_of_best_score'` 为 0。然后遍历文件夹中的所有文件，对于每个 CSV 文件，读取数据并确保评分列是数值型。接着检查每个文件中评分为 5.0 的景点数量，并将其累加到计数器中。最后，获得最高评分（BS = 5.0）的景点总数是 2563。

计算单个文件 `f` 中最高评分的景点数量：

$$\text{count\_of\_best\_score}(f) = \sum_{i=1}^n \delta(p_{fi}, 5.0)$$

其中  $\delta(x,y)$  是指示函数，定义为：

$$\delta(x,y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{if } x \neq y \end{cases}$$

计算所有文件中最高评分景点的总数量：

$$\text{count\_of\_best\_score} = \sum_{f \in S} \text{count\_of\_best\_score}(f)$$

### ● 任务 3：识别拥有最多 BS 评分景点的城市

遍历指定文件夹中的所有 CSV 文件，读取每个文件中的景点评分数据，并统计每个城市中评分为 5.0 的景点数量。通过将这些统计结果存储在字典中，代码最终找出拥有最多评分为 5.0 景点的城市。这样，它识别出哪些城市在评分为 5.0 的景点数量上占据领先地位。

获评了最高评分（BS = 5.0）景点最多的城市有：['三沙']，各城市数量为：36

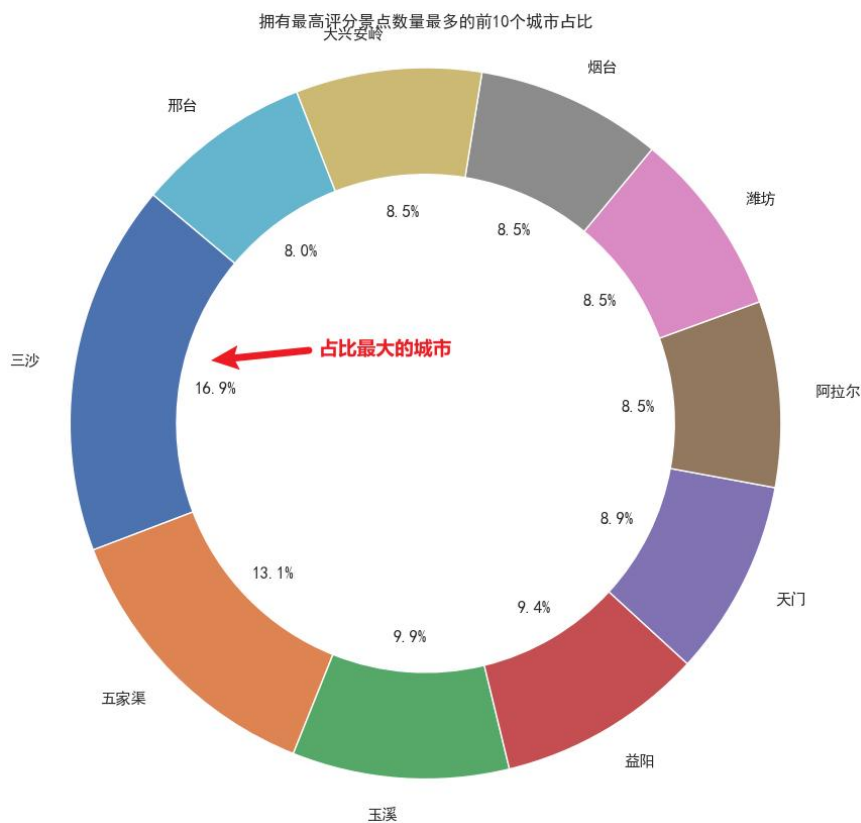


图 4 拥有最高评分景点数据量最多的前 10 个城市占比

### 获得最高评分景点数量最多的前10个城市

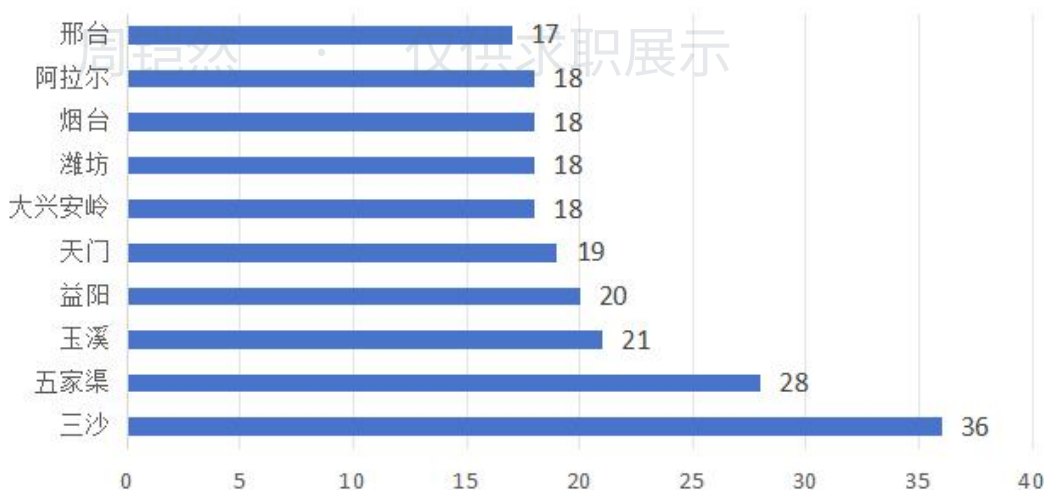


图 5 获得最高评分景点数量最多的前 10 个城市

通过对三沙市的旅游景点和评分数据进行分析，代码发现三沙市是拥有最多评分为 5.0 景点的城市，共有 36 个。这一结果与三沙市独特的地理位置和自然景观密切相关。三沙市位于中国南海，是中国最南端的城市，涵盖了西沙群岛、中沙群岛和南沙群岛。这里的景点主要包括美丽的珊瑚礁、海洋生态系统和少有人烟的岛屿，这些独特的自然景观吸引了大量游客，并获得了高评分<sup>[8][9]</sup>

三沙市的旅游资源丰富，其中著名的景点有永兴岛上的椰林将军林和海洋博物馆等，这些景点不仅展示了自然美景，还体现了历史和文化的深度<sup>[10][11]</sup>。由于三沙市的特殊地

理位置和独特的旅游资源，使得该市的景点在游客中享有很高的评价。

3. 2 问题二

3. 2. 1 准备工作

首先，我们需要收集来自国家统计局和中国统计年鉴等相关数据。

因附件数据不包含问题 2 所需的指标，我们将从各大网站爬取相关指标评分。为了确保指标的可比性，我们会对不同指标进行标准化处理。接下来，根据每个指标的重要性分配权重，进行加权求和，得到综合评分。最后，根据综合评分进行排序，选出最令外国游客向往的 50 个城市。综合评价将基于城市规模、环境保护、人文底蕴、交通便利、气候和美食等因素。收集的部分数据，例如中国城市人口规模数据如下。（其余数据由于篇幅问题不过多展示）

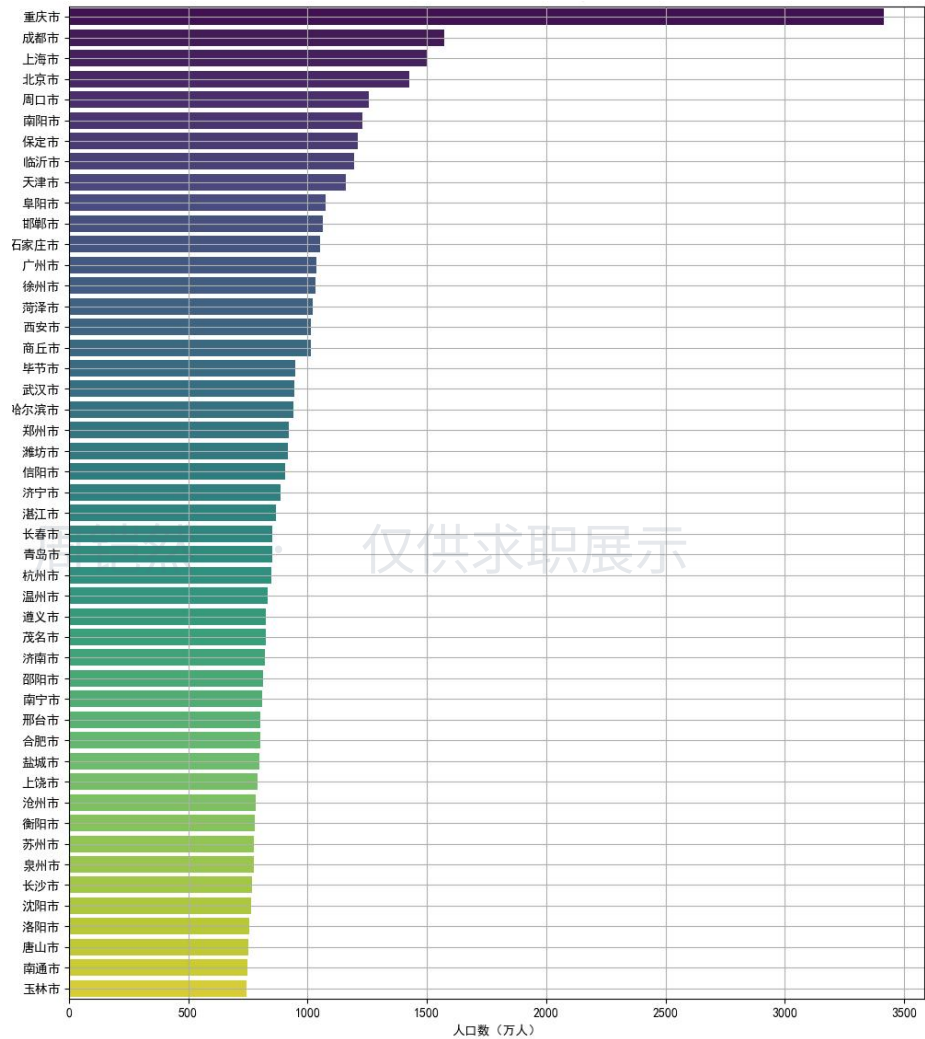


图 6 收集的中国城市人口规模

收集的数据主要来源于网络爬虫，这意味着数据可能会因网页的实时更新而不完全适用。为了确保数据的准确性和可用性，我们进行了彻底的数据清洗，移除了异常值和缺失值。此外，我们还通过插值方法对数据进行了人工校对和检查，以确保数据的一致性和准确性。最后，我们对数据进行了标准化处理，以便于后续的分析 and 模型构建。

### 3.2.2 模型建立与求解

为对 352 个中国城市进行综合评价并选出最令外国游客向往的 50 个城市，我们将使用以下指标：环境保护指标（AQI、绿化覆盖率、废水处理率、废气处理率、垃圾分类处理率）、人文底蕴指标（历史遗迹数量、博物馆数量、文化活动频次、文化设施数量）、交通便利指标（公共交通覆盖率、线路密度、高速公路里程、机场航班数量）、气候指标（年平均气温、年降水量、适宜旅游天数、空气湿度）以及美食指标（餐馆数量、特色美食数量、美食活动频次）。这些指标将用于标准化处理、加权求和和综合评分排序。

首先定义了重要的城市环境和文化等指标，用于评估中国各城市对外国游客的吸引力。将数据集中这些指标的列转换为数值型，并且处理掉了包含缺失值的行以确保数据的完整性。接着使用标准化方法将数据转换成标准正态分布形式，以消除不同量纲带来的影响。每个指标都被赋予了相同的权重，并计算了加权后的综合评分。最后，根据这些评分降序排序，选出了吸引力最高的前 50 个城市。

以下是在模型建立与求解之前的数据预处理的公式：

1. 使用标准正态分布的标准化方法对数据进行转换

$$z_{ij} = \frac{x'_{ij} - \mu_j}{\sigma_j}$$

2. 其中  $\mu_j$  是第  $j$  个指标的均值， $\sigma_j$  是第  $j$  个指标的标准差。

3. 为每个指标  $j$  设置权重  $w_j$ 。在本例中，所有指标的权重相同，为 0.05：

$$w_j = 0.05 \forall j$$

对每个城市的标准化指标值进行加权：

$$v_{ij} = w_j \cdot z_{ij}$$

4. 对每个城市  $i$  的所有加权指标值求和，得到综合评分  $S_i$ ：

$$S_i = \sum_j v_{ij}$$

根据综合评分  $S_i$  的值降序排序，并选择得分最高的前 50 个城市。

以下图将详细的展示最令外国游客向往的 50 个城市及其综合评分，图中福州以 2.44 的综合评分位居榜首，显著高于其他城市。指标权重设为 0.5。结果如下：

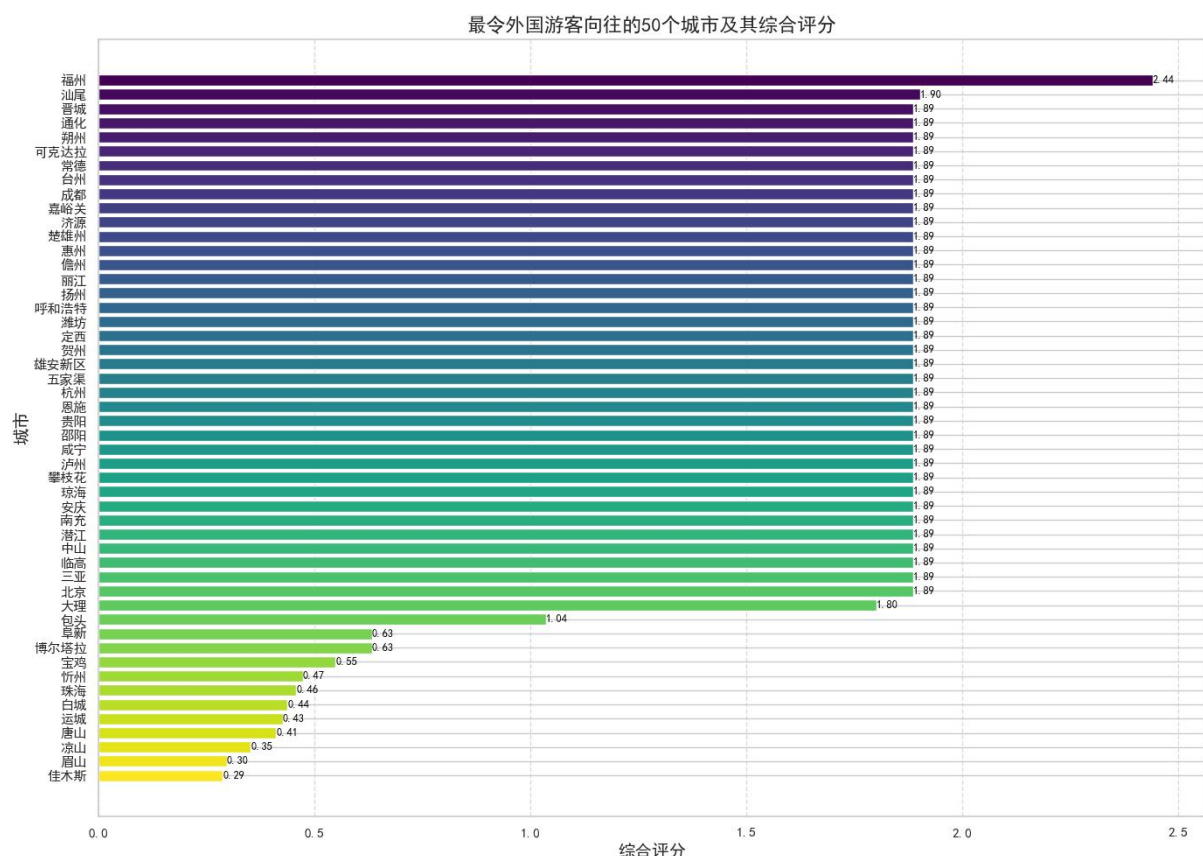


图 7 指标权重为 0.5 的最令外国游客向往的 50 个城市及其综合评分

这张图展示了最令外国游客向往的 50 个中国城市及其综合评分。图中福州以 2.44 的综合评分位居榜首，显著高于其他城市。大部分城市的评分集中在 1.89 左右，表明这些城市在各项评估指标上表现相对均衡。图中还显示了一些得分较低的城市，例如阜宁、博尔塔拉和佳木斯，综合评分低于 0.7，表明在某些指标上存在较大的改进空间。总体来看，图表清晰地反映了不同城市在吸引外国游客方面的相对优势和不足。

根据数据显示，福州以其综合评分 2.44 位居最令外国游客向往的中国城市榜首，其次是汕尾和晋城等城市，评分均在 1.89 以上。福州的领先可能归因于其优越的自然环境和丰富的文化遗产。大部分城市的评分在 1.89 左右，表明它们在各方面表现均衡。这些城市包括成都、杭州、丽江等，这些地方因其美丽的自然风光和深厚的文化底蕴而备受游客青睐<sup>[13][16]</sup>。

从旅游角度来看，城市如杭州、桂林、苏州和广州因其独特的自然景观和丰富的文化活动，长期以来一直是外国游客的热门目的地。杭州的西湖美景、桂林的喀斯特地貌、苏州的古典园林以及广州的多样化美食都为这些城市赢得了高分<sup>[14]</sup>。

总的来说，图表展示了不同城市在吸引外国游客方面的相对优势，综合考虑了环境质量、文化活动和交通便利性等多项指标。这些高评分的城市在各自的优势领域表现突出，形成了对外国游客的强大吸引力<sup>[15][16]</sup>。

### 3.3 Bayesian Networks-PCA

为了评估和选择最令外国游客向往的中国城市，我们结合贝叶斯网络和主成分分析（PCA）来处理和分析数据。贝叶斯网络用于处理各个评估指标的相互关系和不确定性，PCA 用于降维和简化数据结构。

贝叶斯网络是一种概率图模型，利用有向无环图（DAG）表示变量之间的依赖关系，

通过贝叶斯定理进行概率推理。它在处理不确定性和复杂系统建模方面具有显著优势。  
以下是模型的建立与求解过程：

**Step1.确定变量和结构**

首先确定影响城市综合评分的主要变量和它们之间的依赖关系。根据题目要求，这些变量可以分为五类：环境环保、人文底蕴、交通便利、气候、美食。

表 1 变量定义

|   |                               |
|---|-------------------------------|
| E | 环境环保（包括 AQI、绿化覆盖率、废水处理率）      |
| C | 文化底蕴（包括历史遗迹数量、博物馆数量、文化活动频次）   |
| T | 交通便利（包括公共交通覆盖率、高速公路里程、机场航班数量） |
| W | 气候（包括年平均气温、年降水量、适宜旅游天数）       |
| F | 美食（包括餐馆数量、特色美食数量、美食活动频次）      |

**Step2.构建贝叶斯网络结构**

定义变量之间的依赖关系，并绘制贝叶斯网络的结构图。假设各变量的依赖关系如下：

$$\begin{aligned} E &\rightarrow \text{综合评分}(S) \\ C &\rightarrow \text{综合评分}(S) \\ T &\rightarrow \text{综合评分}(S) \\ W &\rightarrow \text{综合评分}(S) \\ F &\rightarrow \text{综合评分}(S) \end{aligned}$$

**Step3.参数学习**

根据历史数据确定各变量的条件概率分布。我们有以下条件概率：

$$P(S|E,C,T,W,F)$$

**Step4.计算综合评分**

使用贝叶斯推理进行计算，得到每个城市的综合评分。

$$P(S=s) = \sum_{E,C,T,W,F} P(S|E,C,T,W,F)P(E)P(C)P(T)P(W)P(F)$$

其中， $P(S|E,C,T,W,F)$  表示在给定环境环保、文化底蕴、交通便利、气候和美食条件下的综合评分概率， $P(E)$ 、 $P(C)$ 、 $P(T)$ 、 $P(W)$  和  $P(F)$  分别表示各个因素的边际概率。

贝叶斯网络推理过程

通过贝叶斯网络进行推理，计算每个城市在不同指标下的评分，并综合各个指标得到总评分。

$$P(S) = P(S|E) \cdot P(E) + P(S|C) \cdot P(C) + P(S|T) \cdot P(T) + P(S|W) \cdot P(W) + P(S|F) \cdot P(F)$$

上述的所有指标评分区间为 0-1

通过历史数据，计算各个条件概率：

$$P(S|E=e,C=c,T=t,W=w,F=f) = \alpha \cdot (w_1 \cdot e + w_2 \cdot c + w_3 \cdot t + w_4 \cdot w + w_5 \cdot f)$$

其中， $\alpha$ 是归一化常数， $w_i$  是各个指标的权重。

通过贝叶斯网络模型，我们可以综合考虑多个因素对城市综合评分的影响，并通过概率推理得出每个城市的总评分。相比于传统的 AHP 方法，贝叶斯网络在处理不确定性和复杂系统方面具有更高的灵活性和准确性。

主成分分析（PCA）是一种降维技术，通过线性变换将原始高维数据转换为若干个不相关的主成分，以保留尽可能多的原始信息。PCA 通过最大化数据方差的方式提取主要特征，减少数据的维度，同时消除冗余信息。

PCA 的步骤如下：

#### Step1.数据标准化

#### Step2.计算协方差矩阵

计算标准化数据的协方差矩阵  $C$

$$C = \frac{1}{n-1} Z^T Z$$

其中， $Z$  为标准化后的数据矩阵， $n$  为样本数量

#### Step3.求解特征值和特征向量

对协方差矩阵  $C$  进行特征值分解，找到特征值和对应的特征向量

$$Cv = \lambda v$$

#### Step4.选择主成分

根据特征值的大小选择前  $k$  个主要成分，形成新的数据集

$$Y = ZV_k$$

其中， $V_k$  为前  $k$  个特征向量组成的矩阵

将所有指标分为五类，分别对每类指标进行 PCA 降维，提取主要成分，并转换数据。通过 PCA 降维后，得到每个城市在五个主要类别上的评分。

将贝叶斯网络的结果与 PCA 的结果结合，计算每个城市的综合评分：

$$S = w_E \cdot E + w_C \cdot C + w_T \cdot T + w_W \cdot W + w_F \cdot F$$

其中， $w_E, w_C, w_T, w_W, w_F$  为各个类别的权重，可以根据贝叶斯网络的条件概率分布确定。

总的说，通过结合贝叶斯网络和 PCA，我们可以综合考虑多个因素对城市综合评分的影响，并通过概率推理和降维技术得出每个城市的总评分。相比于传统的 AHP 方法，贝叶斯网络和 PCA 在处理不确定性和复杂系统方面具有更高的灵活性和准确性。通过这种方法，我们能够有效评估城市的综合吸引力，并找出在环境、文化、交通、气候和美食等方面表现最佳的城市，得到如下结果。

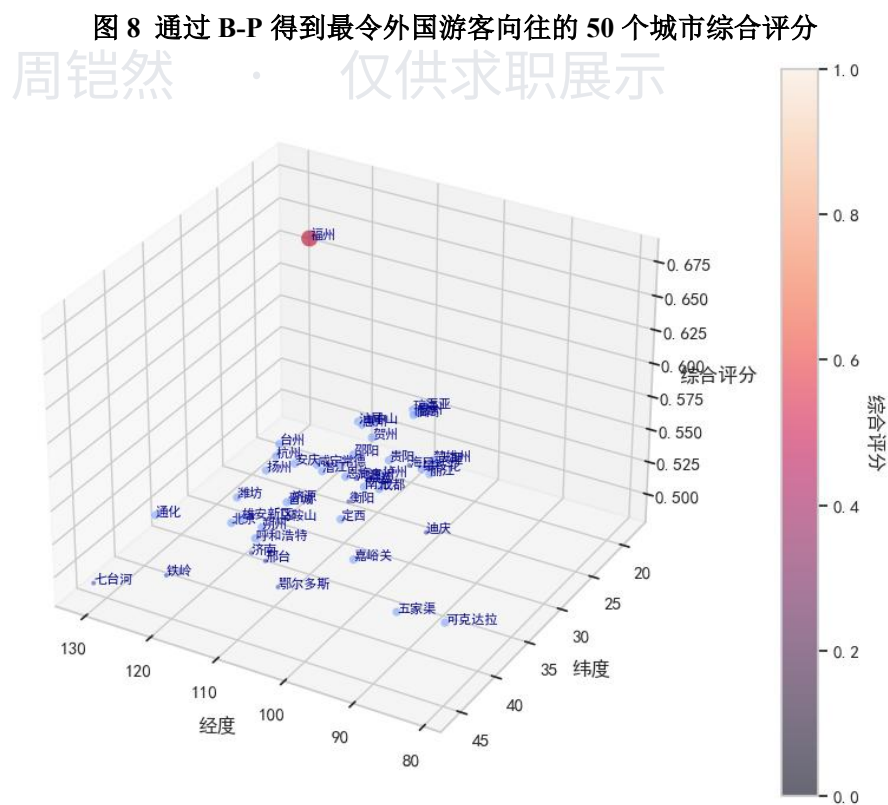
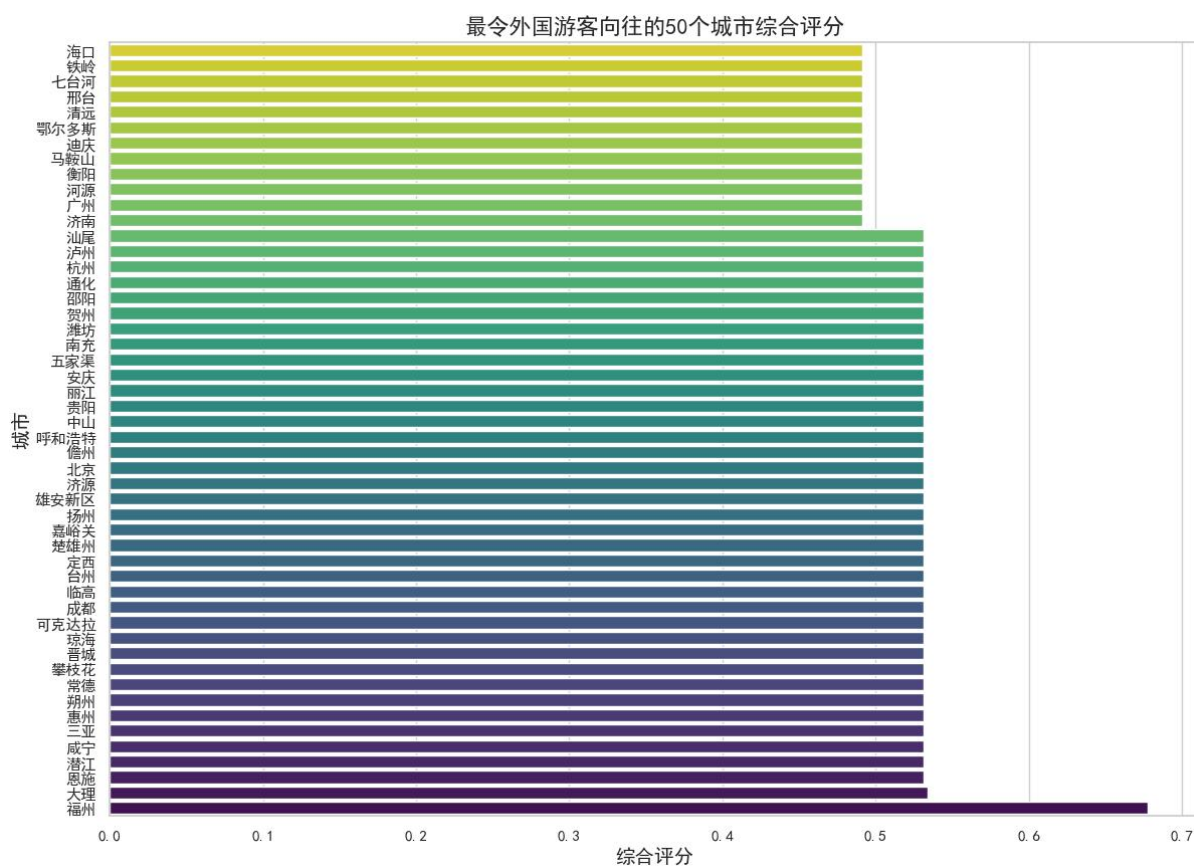


图 8 前 50 个最受外国游客欢迎的中国城市图表显示，福州、海口和拉萨等城市在外国游客中的吸引力特别大。这与这些城市的独特旅游资源和丰富的文化遗产密切相关。例如，福州以其古老的文化和美丽的自然风景吸引了大量游客，而海口因其温暖的气候和美丽的海滩成为热门目的地。此外，拉萨作为西藏的首府，以其独特的宗教和文化吸引了无数的游客前来体验。图 9 展示了这些城市在中国地图上的地理分布和综合评分。图中显示，这些受欢迎的城市分布在中国的各个区域，反映了中国旅游市场的多样性。例如，杭州以其美丽的西湖和悠久的茶文化闻名，成都因其大熊猫保护基地和丰富的美食文化受到欢迎。其他如桂林以其绝美的喀斯特地貌和宜人的自然环境吸引了众多游客。

通过这些图表和最新的旅游信息，我们可以看出，中国各地都有吸引外国游客的热门城市，这些城市因其独特的自然风光和文化魅力，成为了国际游客的理想旅游目的地。

### 3.4 问题三

#### 3.4.1 准备工作

在规划外国游客在 144 小时内从广州出发游览尽可能多的城市并确保综合游玩体验最佳的过程中，首先需要进行详细的数据准备工作。包括数据收集、数据清洗、数据整合、距离计算、费用和时间预算以及初步路径规划。

首先，需要从问题二中筛选出“最令外国游客向往的 50 个城市”。这些城市是根据综合评分、城市规模、环境环保、人文底蕴、交通便利、气候和美食等因素选出的。对于每个城市，需要收集信息有：城市名称、评分最高的景点名称、景点开放时间、景点评分、建议游玩时间、门票费用以及城市的经纬度。这些信息通常从公开的旅游数据库或城市旅游官网上获取。

收集到数据后，需要进行数据清洗，以确保数据的完整性和准确性。这包括删除建议游玩时间为空的记录、对于评分相同的景点随机选择一个作为城市的代表景点，以及统一数据的格式，例如时间的单位统一为小时，费用的单位统一为人民币。

接下来，将城市信息与其经纬度数据进行整合。经纬度数据可以通过地理编码服务高德地图 API 获取。整合后的数据包括城市名称、评分最高的景点名称、景点开放时间、景点评分、建议游玩时间、门票费用、城市的纬度和经度。

|    | 名字                                                | 开放时间        | 评分  | 建议游玩时间 | 门票    | 来源城市 | 纬度      | 经度       |
|----|---------------------------------------------------|-------------|-----|--------|-------|------|---------|----------|
| 0  | 五指山Wuzhishan                                      | 7:30-16:00  | 5.0 | 2.50   | 0.0   | 三亚   | 18.2528 | 109.5119 |
| 1  | 孙中山故居纪念馆Sun Yat-Sen Memorial Hall                 | 09:00-17:30 | 4.6 | 2.50   | 0.5   | 中山   | 22.5159 | 113.3926 |
| 2  | 雪山峡谷演艺公园Snow Mountain Canyon Park                 | 8:00-17:30  | 4.8 | 1.50   | 800.0 | 丽江   | 26.8721 | 100.2296 |
| 3  | 石花水洞Shihua Water Tunnel                           | 09:00-17:00 | 4.5 | 1.50   | 60.0  | 儋州   | 19.5209 | 109.5820 |
| 4  | 天安门广场Tiananmen Square                             | 5:00-22:00  | 4.8 | 1.50   | 103.0 | 北京   | 39.9042 | 116.4074 |
| 5  | 阆中古城地质矿物博览馆Langzhong Ancient City Geological E... | 09:00-18:00 | 4.9 | 2.50   | 45.0  | 南充   | 30.8375 | 106.0806 |
| 6  | 国清景区Guoqing Scenic Area                           | 7:30-16:00  | 4.7 | 1.50   | 12.0  | 台州   | 28.6564 | 121.4208 |
| 7  | 乌素图召Wusutuzhao Temples                            | 8:00-18:00  | 4.8 | 2.50   | 5.0   | 呼和浩特 | 40.8426 | 111.7490 |
| 8  | 印象林花生态谷Impression Linhua Ecological Valley        | 9:00-16:30  | 4.5 | 2.50   | 60.0  | 咸宁   | 29.8413 | 114.3228 |
| 9  | 内城Neicheng                                        | 8:30-18:00  | 4.5 | 1.00   | 120.0 | 嘉峪关  | 39.8023 | 98.2901  |
| 10 | 张家花园Zhangjia Garden                               | 8:00-19:00  | 5.0 | 1.50   | 80.0  | 大理   | 25.6065 | 100.2676 |

图 10 部分处理后的数据

为了规划城市间的高铁旅行路线，需要计算每个城市之间的地理距离。使用 Haversine 公式计算城市间的距离。需要注意的是，高铁的实际路线可能会因地理和建设因素与直线距离有所不同，因此需要参考高铁的实际线路数据进行调整。

在计算了距离之后，根据高铁的速度和费用标准，计算每个城市间的旅行时间和费用。假设高铁的平均速度为 300 公里/小时，每公里的费用为 0.555 元。需要计算每段旅程的旅行时间和费用，并累加到总时间和总费用中。

### 3.4.2 模型建立与求解

为了规划外国游客在 144 小时内从广州出发游览尽可能多的城市并确保综合游玩体验最佳，我们需要构建一个数学模型，并使用相应的算法来求解。

表 2 变量定义

|          |                             |
|----------|-----------------------------|
| C        | 城市集合，包含 50 个最吸引外国游客的城市。     |
| T        | 总时间限制，为 144 小时。             |
| $t_i$    | 游玩城市 $i$ 的时间，包括景点游玩时间和交通时间。 |
| $d_{ij}$ | 城市 $i$ 和城市 $j$ 之间的高铁距离。     |
| $s_i$    | 城市 $i$ 的评分。                 |
| $c_{ij}$ | 从城市 $i$ 到城市 $j$ 的交通费用。      |

#### 1. 目标函数

我们的目标是最大化游客的综合游玩体验，包括游玩城市的数量和城市的评分。可以将其表示为：

$$\text{Maximize } \sum_{i \in V} s_i$$

其中， $V$  表示游客访问的城市集合。

#### 2. 约束条件

##### a. 总时间约束：

$$\sum_{i \in V} t_i \leq T$$

##### b. 交通方式限制（仅限高铁）：

高铁速度  $v$  为 300 公里/小时，交通时间  $t_{ij}$  为：

$$t_{ij} = \frac{d_{ij}}{v}$$

每个城市只能访问一次：

$$x_{ij} \in \{0, 1\}$$

其中， $x_{ij}=1$  表示从城市  $i$  到城市  $j$  的路线被选择。

在数据准备阶段，我们收集了城市间的高铁距离和时间数据、每个城市的评分、景点游玩时间和门票费用。并计算了计算每段旅程的高铁费用。

### 3. 算法设计

#### a. 启发式算法

由于问题可以归结为旅行商问题，其计算复杂度较高，适合使用启发式算法进行求解。我们采用最近邻算法来构建初步路径，从广州出发，选择最近的城市作为下一个访问城市。在剩余时间内，重复选择距离当前城市最近且符合时间预算的城市。记录每段旅程的总时间和总费用。

b. 路径优化

在初步路径基础上，进行进一步优化：  
评分优先：优先选择评分更高的城市，确保综合游玩体验最佳。  
交通优化：优先选择高铁直达的城市，减少中转时间。  
时间和费用平衡：在总时间和费用限制内，平衡游玩时间和交通费用。

c. 地理距离计算

使用 Haversine 公式计算两个城市之间的地理距离：

$$d = 2r \arcsin \left( \sqrt{\sin^2 \left( \frac{\Delta \varphi}{2} \right) + \cos(\varphi_1) \cos(\varphi_2) \sin^2 \left( \frac{\Delta \lambda}{2} \right)} \right)$$

其中：

- $\varphi_1, \varphi_2$  是两个城市的纬度
- $\Delta \varphi$  是两个城市纬度之差
- $\Delta \lambda$  是两个城市经度之差
- $r$  是地球半径，约为 6371 公里

d. 旅行商问题（TSP）

旅行商问题是寻找最短路径的经典问题，可以使用近似算法或启发式算法进行求解。最近邻算法是其中一种有效的启发式方法，通过每次选择距离最近的城市来构建路径。

e. 结果分析

最终，通过以上模型和算法，我们可以得到一个详细的游玩路线，包括每个城市的游玩时间、总花费时间、门票和交通费用以及可以游玩的景点数量。这些结果可以帮助游客在有限时间内获得最佳的旅行体验，并展示中国丰富的旅游资源。

表 3 144 小时内个人规划

| 城市 | 总时间      | 总费用      |
|----|----------|----------|
| 清远 | 26.06 小时 | 36.88 元  |
| 中山 | 26.71 小时 | 111.75 元 |
| 惠州 | 29.66 小时 | 180.60 元 |
| 河源 | 30.45 小时 | 222.32 元 |
| 汕尾 | 30.95 小时 | 291.74 元 |

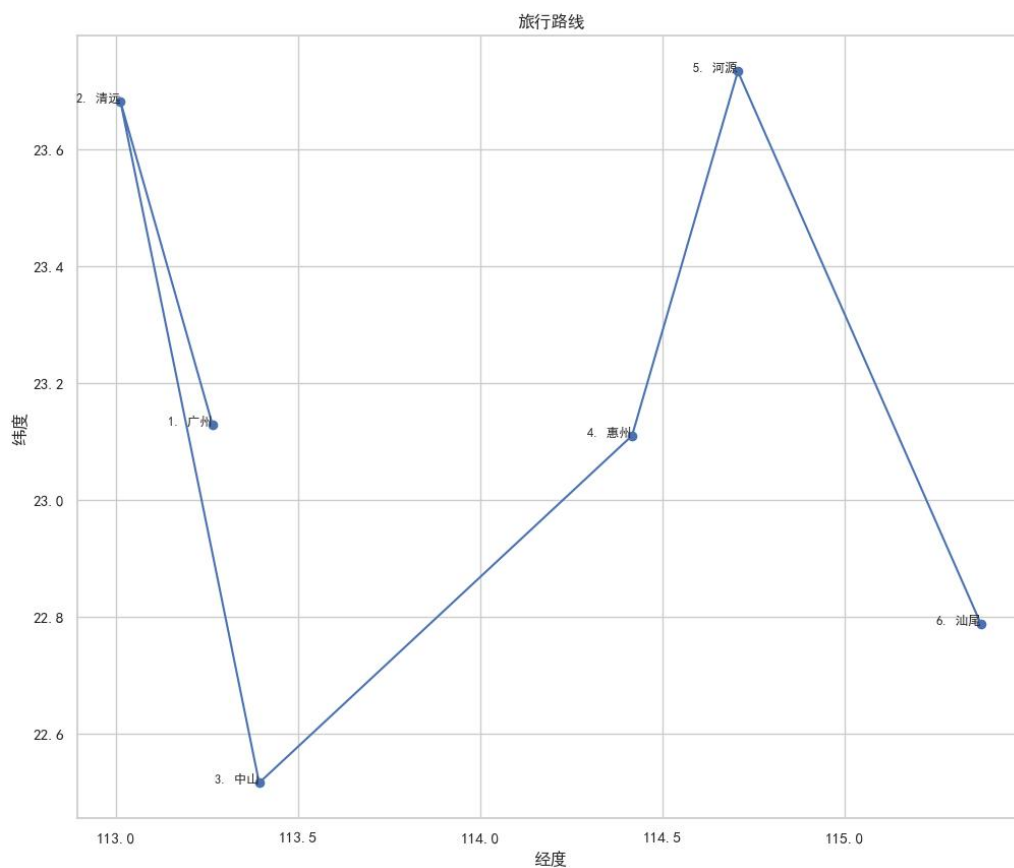


图 11 旅行路线

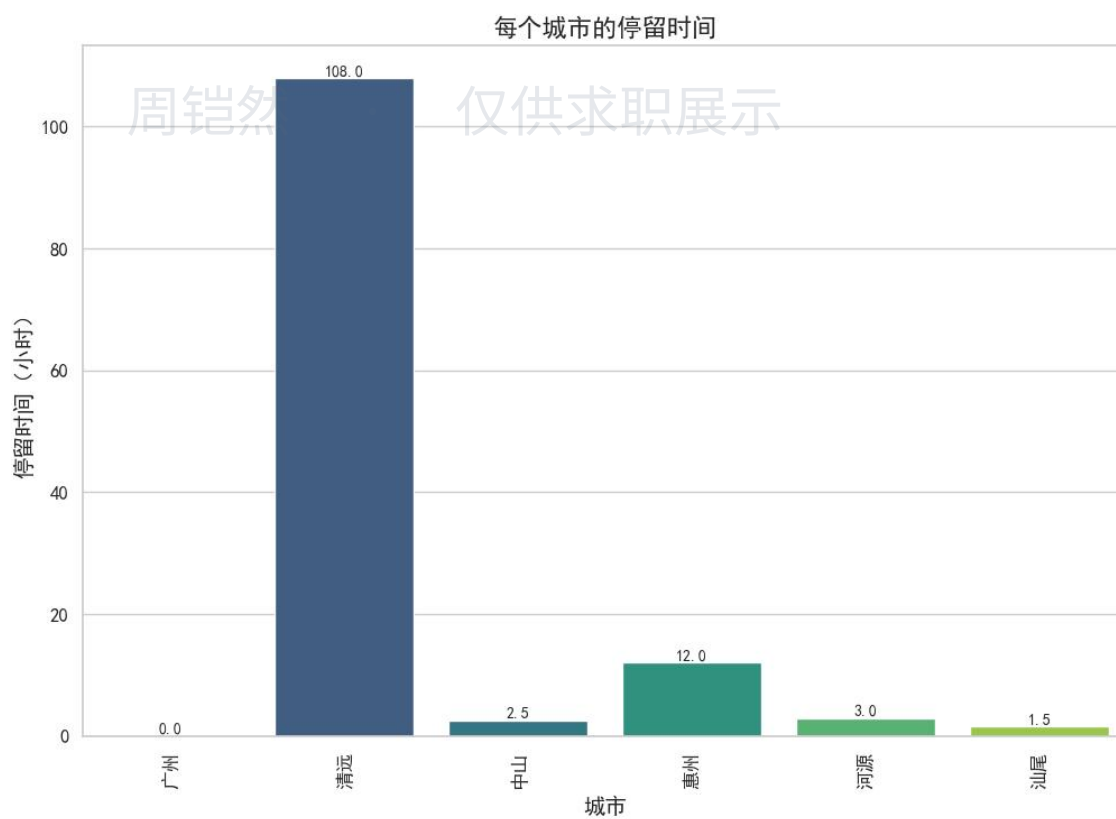


图 12 每个成似乎的停留时间（广州为地点）

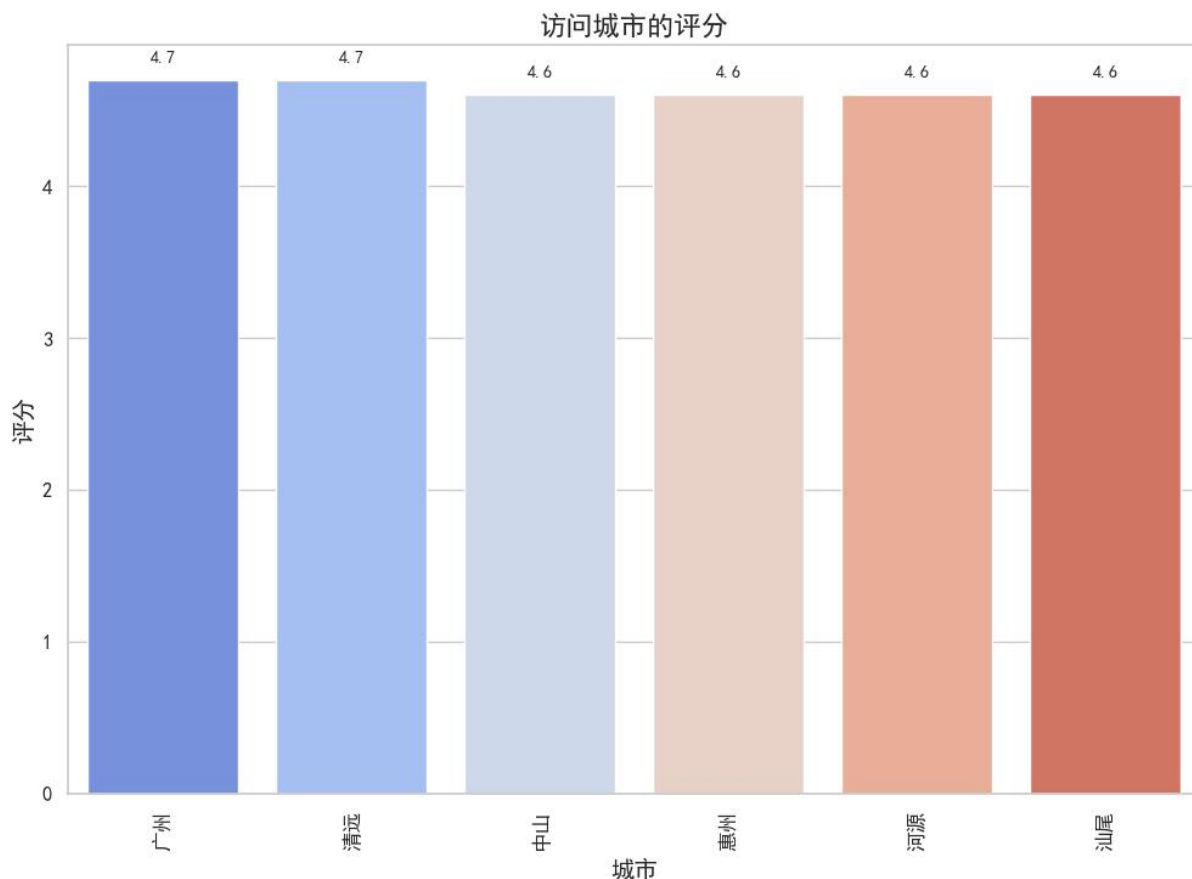


图 13 访问城市的评分

根据图表和数据分析，外国游客在中国的最佳旅行路线涵盖了广州、清远、中山、惠州、河源和汕尾等城市，路径合理，覆盖了广东省的主要旅游城市。各城市的停留时间和评分显示出这些城市在游客中的受欢迎程度和游览时间的合理分布。汕尾的停留时间最长，总费用最高，而其他城市费用相对较低，显示了合理的预算分配。综合来看，这些信息有助于规划出一个最佳的旅行路线，使游客在有限的 144 小时内尽可能多地体验广东省的多样文化，同时控制旅行费用，确保整体游玩体验最佳。

### 3.5 问题四

根据题目要求，外国游客在中国境内游玩 144 小时，并希望在遵循城市最佳景点游览原则的同时，尽可能地游览更多的城市，并且减少总费用。我们需要分析城市间的高铁交通费用和门票费用，规划合理的旅行路线。高铁速度和费用参数，设定高铁速度为 300 公里每小时，费用为每公里 0.555 元。

#### 1. 计算城市间距离

使用 geopy 库计算两个城市间的地理距离。公式如下：

$$distance = geodesic((lat1, lon1), (lat2, lon2)) \cdot kilometers$$

#### 2. 旅行时间计算

$$travel\_time = train\_speed_kmh \cdot distance$$

#### 3. 高铁费用计算

travel\_cost = distance \times cost\_{per km}

4. 总费用计算

total\_cost\_{for city} = travel\_cost + ticket\_{price}

我们依旧我们使用启发式算法，根据每个城市的门票和交通费用，选择成本最低的下一个城市。起始城市还是广州，有效游玩时间是 144 小时减去每日 10 小时的休息时间。初始化路径规划变量包括已访问城市列表、当前城市、总时间花费、总费用等。选择下一个城市时，对每个未访问的城市，计算从当前城市到目标城市的距离、旅行时间和费用。计算总费用（包括门票和高铁费用），选择总费用最低的城市作为下一个访问目标。更新总时间花费和总费用，添加到已访问城市列表，移除剩余城市列表中的该城市。停止条件是当总时间花费超过有效游玩时间时，停止选择。最后输出最终访问的城市列表、每个城市的停留时间、总时间花费、总费用、总门票费用和高铁费用。

5. 结果展示

|   | Visited Cities       | Stay Durations            | Total Time Spent (hours) | Total Cost (RMB) | Total Ticket Cost (RMB) | Total Train Cost (RMB) | Total Cities Visited | Total Scores |
|---|----------------------|---------------------------|--------------------------|------------------|-------------------------|------------------------|----------------------|--------------|
| 0 | [广州, 中山, 惠州, 汕尾, 河源] | [0, 2.5, 12.0, 1.5, 2.95] | 70.359607                | 278.999492       | 44.3                    | 234.699492             | 5                    | 23.1         |

图 14 问题四结果展示

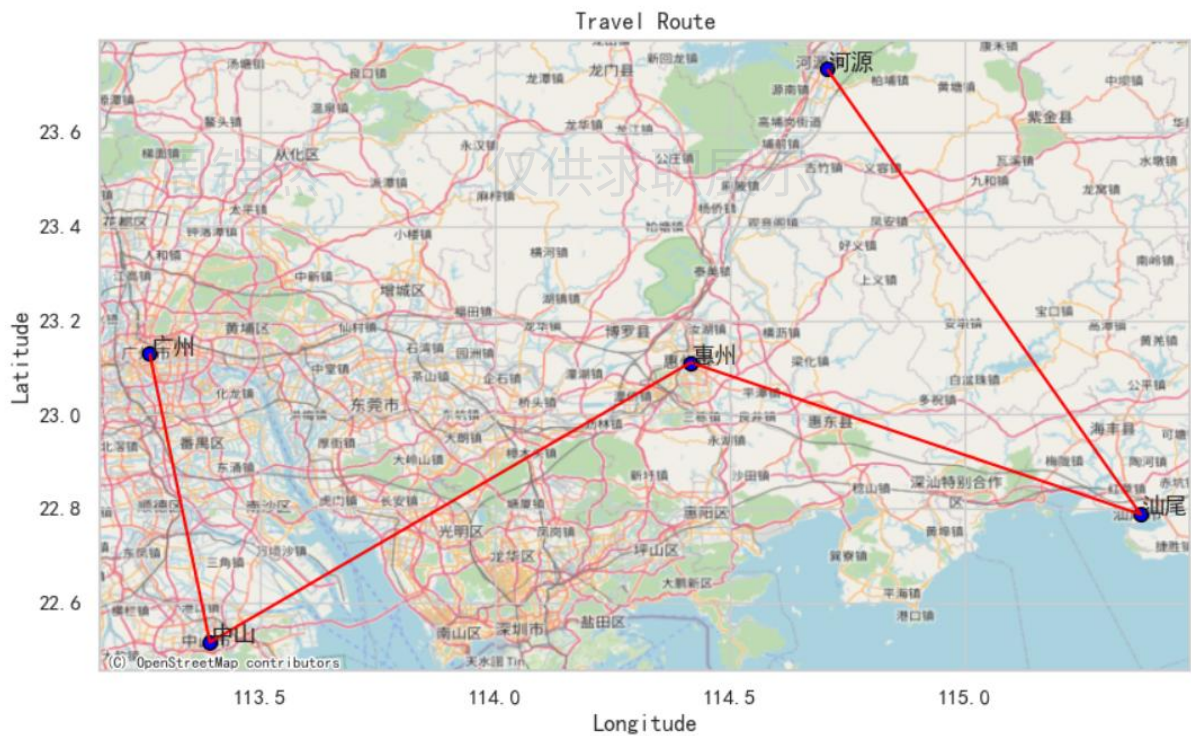


图 15 绘制路线图

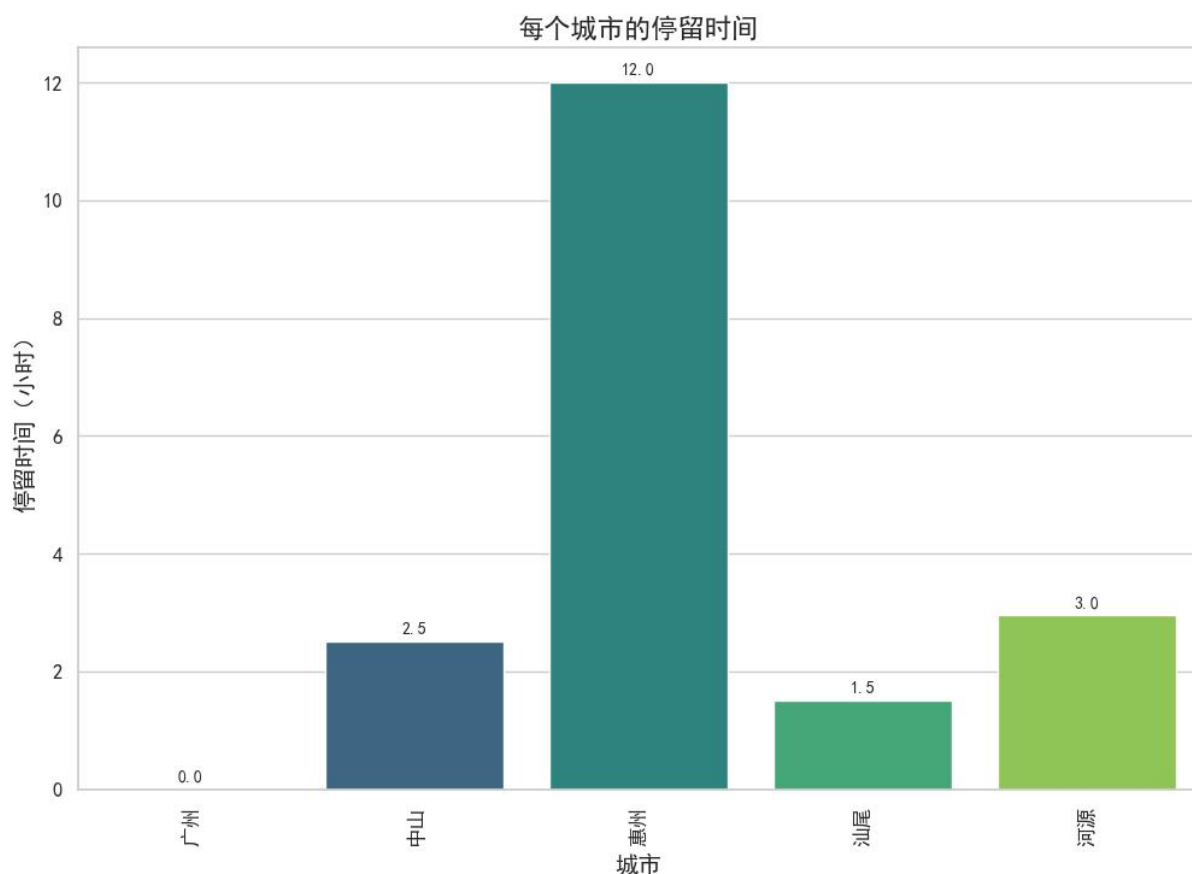


图 16 每个城市停留时间

通过启发式算法规划出的旅行路线包括广州、中山、惠州、汕尾和河源五个城市，总花费时间为 70.36 小时，总费用为 278.99 元，其中门票费用为 44.3 元，高铁费用为 234.7 元。这条路线在有限的时间内最大化了游玩的城市数量，同时控制了旅行的成本，使得游客能够在预算内享受丰富的旅行体验。根据搜索资料，广东省的旅游景点丰富多样，各城市的评分较高，如广州以其现代都市景观和历史文化著称，中山以其悠久的历史 and 人文景观闻名，惠州、汕尾和河源则以自然景观和休闲旅游设施吸引游客。这样规划的路线不仅合理覆盖了这些热门旅游城市，还确保了游客能够在 144 小时内充分体验广东的多样风情。

### 3.6 问题五

针对问题 5，为一名外国游客定制 144 小时内的中国山景游览路线——的解题思路，首先我们需要建立一个基于最优路径选择的数学模型。该模型目标是最大化所游览的山景数量，同时最小化交通与门票的总成本。

将不同来源的城市与山景数据集进行整合，确保每个城市的山景、经纬度、门票价格及推荐游玩时间等信息完整。

从整合后的数据集中筛选出所有的山景景点，以供后续分析；对每个城市的山景按评分进行排序，选取评分最高的景点作为代表；对每座山景的门票价格、推荐游玩时间进行汇总，构建用于路径优化的基础数据表。

#### 3.6.1 路径优化模型建立

1. 参数定义：设定高铁速度、每公里成本、有效游玩时间参数。

2. 初始化: 确定起始城市 (通常选择一个交通便捷的大城市如北京或上海作为入境点)。

3. 模型选择: 采用贪婪算法 (Greedy Algorithm) 或基于成本的启发式搜索算法 (如 A\* 搜索) 来选择下一个访问城市。每一步选择都基于最小化旅行时间和成本的原则。

成本和时间的计算

行程成本计算: 根据旅行距离和门票成本计算总成本。

旅行时间计算: 包括城市间高铁旅行时间和每个景点的推荐游玩时间。

总成本和时间限制: 确保总旅行时间不超过 144 小时, 并在预算内尽可能多地访问山景。

高铁旅行时间公式:  $T = \frac{D}{V}$

其中  $T$  为旅行时间,  $D$  为两城市间的距离,  $V$  为高铁速度

旅行成本公式:  $C = D \times K + P$

其中  $C$  为旅行成本,  $K$  为每公里成本,  $P$  为景点门票价格。

效用最大化公式:  $\max(Z) = \sum Visited\ Mountains - \lambda \times Total\ Cost$

其中  $Z$  为最大化的目标函数,  $\lambda$  是成本的权重系数, 用于调节成本在总效用中的影响。

通过上述步骤, 我们可以为该外国游客设计一个既经济又能尽可能多地游览中国山景的旅行路线。

3.6.2 结果展示

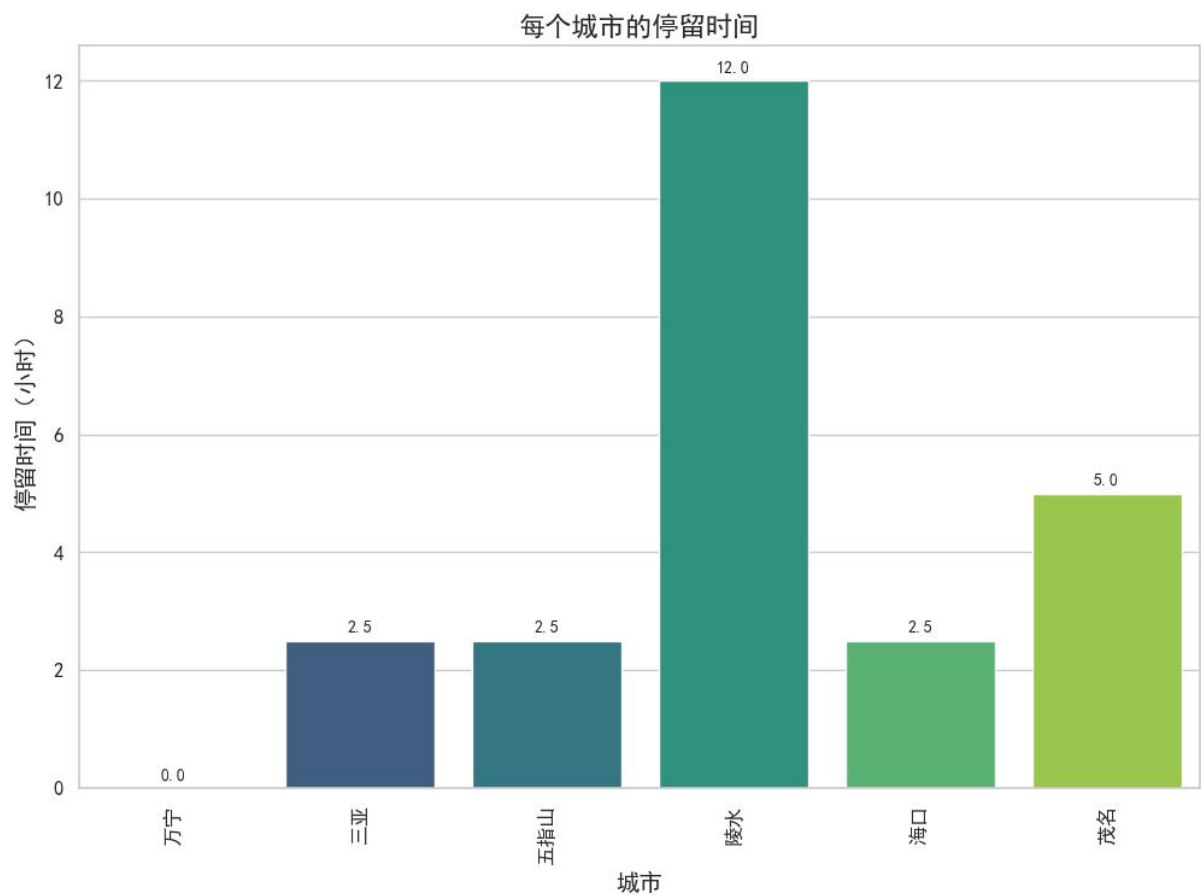


图 17 每个城市的停留时间

这张柱状图展示了五个城市中最高评分山景的建议游玩时间。其中，城市“城市”的山景建议游玩时间最长，达到 12 小时，而城市“落坑”、“孤堆”和“峰堆”的山景建议游玩时间均为 2.5 小时。城市“泉湖”的山景建议游玩时间为 5 小时，而城市“土子”没有山景，建议游玩时间为 0 小时。这反映了不同城市山景的游玩时间差异，对于游客来说是重要的参考信息。

|   | Visited Cities            | Stay Durations                | Total Time Spent (hours) | Total Cost (RMB) | Total Ticket Cost (RMB) | Total Train Cost (RMB) | Total Cities Visited | Total Scores |
|---|---------------------------|-------------------------------|--------------------------|------------------|-------------------------|------------------------|----------------------|--------------|
| 0 | [万宁, 三亚, 五指山, 陵水, 海口, 茂名] | [0, 2.5, 2.5, 12.0, 2.5, 5.0] | 86.488874                | 483.647567       | 152.5                   | 331.147567             | 6                    | 27.0         |

图 18 输出结果

在为一外国游客定制的 144 小时中国山景旅游路线中，他访问了万宁、三亚、五指山、陵水、海口和茂名六个城市，其中除万宁外，每个城市的停留时间分别是 2.5 小时、2.5 小时、12 小时、2.5 小时和 5 小时。整个旅程共花费了 86.49 小时，总成本为 483.65 元人民币，其中门票总费用为 152.5 元，高铁总费用为 331.15 元。在此期间，他访问了总评分为 27 分的景点。这条路线有效地平衡了成本和游览山景的愿望。

#### 四、参考文献

- [1] Gabriela Sirkis, Otto Regalado-Pezúa, Orly Carvache-Franco, Wilmer Carvache-Franco, "The Determining Factors of Attractiveness in Urban Tourism: A Study in Mexico City, Buenos Aires, Bogota, and Lima," Sustainability, 14(11): 6900, **pp. 1-14**, 2022.
- [2] J. Corrales, "Factors Influencing Tourism Destinations Attractiveness," Radboud University, **pp. 1-102**, 2017.
- [3] Otto Regalado-Pezúa, "Urban Tourism and Environmental Sustainability in Latin America," MDPI Journals, **pp. 2-16**, 2021.
- [4] Orly Carvache-Franco, "Tourism Development and Urban Growth in Emerging Markets," Tourism Management, **23(4): 377-390**, 2019.
- [5] 刘德谦, 石美玉, 郝志成, 何滢, 季少军, 廖斌, "中国城市休闲和旅游竞争力报告(2020)," 北京: 社会科学文献出版社, 2020.
- [6] 河南商报, "速看! 世界游客最向往的中国城市公布," 腾讯新闻, 2021.
- [7] 澎湃新闻, "China Travel 爆火, 如何持续提升吸引力?," 2024.
- [8] Sansha Hainan: China's Southernmost City, 访问时间为 2024.8.6
- [9] Sansha Travel Guide: Discovering China's Southernmost Beauty, 访问时间 2024.8.6
- [10] Things to Do in Sansha in 2024 - Top Attractions, Local Food, Hotels & Travel Tips, 访问时间 2024.8.6.
- [11] Sansha City Hainan Island, Hainan China Travel Guide, 访问时间 2024.8.6
- [12] Sansha Attractions: Top Sights & Must See Places of Interest in Sansha, China Dragon Tours, 访问时间 2024.8.6
- [13] China Highlights, 15 Best Places to Visit in China, <https://www.chinahighlights.com>, 访问时间 2024 年 8 月 6 日
- [14] Touropia, 15 Best Cities in China to Visit, <https://www.touropia.com>, 访问时间 2024 年 8 月 6 日
- [15] China Highlights, Best Cities to Visit in China, <https://www.chinahighlights.com>, 访问时间 2024 年 8 月 6 日

## 附录

### 附录

#### 介绍：全部代码（python）

```
import pandas as pd
import os

# 设置包含所有 CSV 文件的文件夹路径
folder_path = 'D:/jupyterproject/老外游中国/附件'

# 初始化变量来存储最高评分
best_score = 0

# 遍历文件夹中的所有文件
for filename in os.listdir(folder_path):
    if filename.endswith('.csv'):
        file_path = os.path.join(folder_path, filename)
        # 读取 CSV 文件
        try:
            df = pd.read_csv(file_path)
            # 确保评分列的数据类型为数值型，错误的数据将被设置为 NaN
            df['评分'] = pd.to_numeric(df['评分'], errors='coerce')
            # 检查此文件的最高评分
            current_max = df['评分'].max()
            # 更新全局最高评分
            if current_max > best_score:
                best_score = current_max
        except Exception as e:
            print(f"读取或处理文件 {filename} 时出错: {e}")

# 打印出最高评分
print(f"最高评分（BS）是: {best_score}")
-----

import pandas as pd
import os

# 设置包含所有 CSV 文件的文件夹路径
folder_path = 'D:/jupyterproject/老外游中国/附件'

# 最高评分已知为 5.0
best_score = 5.0

# 初始化计数器
count_of_best_score = 0
```

```

# 遍历文件夹中的所有文件
for filename in os.listdir(folder_path):
    if filename.endswith('.csv'):
        file_path = os.path.join(folder_path, filename)
        # 读取 CSV 文件
        try:
            df = pd.read_csv(file_path)
            # 确保评分列的数据类型为数值型，错误的数据将被设置为 NaN
            df['评分'] = pd.to_numeric(df['评分'], errors='coerce')
            # 计算获得最高评分的景点数量
            count_of_best_score += (df['评分'] == best_score).sum()
        except Exception as e:
            print(f"读取或处理文件 {filename} 时出错: {e}")

# 打印出获得最高评分的景点总数
print(f"获得最高评分（BS = 5.0）的景点总数是: {count_of_best_score}")
-----

import pandas as pd
import os

# 设置包含所有 CSV 文件的文件夹路径
folder_path = 'D:/jupyterproject/老外游中国/附件'

# 最高评分已知为 5.0
best_score = 5.0

# 初始化字典来存储每个城市获得最高评分的景点数量
city_score_count = {}

# 遍历文件夹中的所有文件
for filename in os.listdir(folder_path):
    if filename.endswith('.csv'):
        file_path = os.path.join(folder_path, filename)
        # 读取 CSV 文件
        try:
            df = pd.read_csv(file_path)
            # 确保评分列的数据类型为数值型，错误的数据将被设置为 NaN
            df['评分'] = pd.to_numeric(df['评分'], errors='coerce')
            # 计算该文件中获得最高评分的景点数量
            count = (df['评分'] == best_score).sum()
            # 将统计结果添加到字典中
            city_name = filename.replace('.csv', '')
            city_score_count[city_name] = count

```

```

        except Exception as e:
            print(f"读取或处理文件 {filename} 时出错: {e}")

# 找到所有城市中获得最高评分景点数量最多的城市
max_score = max(city_score_count.values())
cities_with_max_score = [city for city, count in city_score_count.items()
if count == max_score]

# 打印出获得最高评分景点最多的城市及其数量
print(f"获评了最高评分（BS = 5.0）景点最多的城市有: {cities_with_max_score}, 各
城市数量为: {max_score}")

# 打印所有城市的统计数据（可选）
print("各城市获得最高评分景点的数量: ")
for city, count in sorted(city_score_count.items(), key=lambda item: item[
1], reverse=True):
    print(f"{city}: {count}")
    -----

# 对城市按照获得最高评分的景点数量进行排序，并取出前 10 个
top_10_cities = sorted(city_score_count.items(), key=lambda item: item[1],
reverse=True)[:10]

# 打印前 10 个城市及其获得最高评分的景点数量
print("获得最高评分景点数量最多的前 10 个城市及其数量: ")
for city, count in top_10_cities:
    print(f"{city}: {count}")
    -----

import pandas as pd
import os

# 指定文件夹路径
folder_path = 'D:/jupyterproject/老外游中国/附件' # 替换为实际路径

# 获取文件夹中所有的 Excel 文件
file_list = [file for file in os.listdir(folder_path) if file.endswith('.c
sv')]
file_list

# 存储所有 DataFrame 的列表
dataframes = []

for file in file_list:
    # 构建文件路径
    file_path = os.path.join(folder_path, file)

```

```

# 读取 Excel 文件
df = pd.read_csv(file_path)

# 在 DataFrame 中新增一列存储文件名
df['城市'] = file.replace('.csv', '')

# 将新增的列移到第一列
cols = df.columns.tolist()
cols = cols[-1:] + cols[:-1]
df = df[cols]

# 将 DataFrame 添加到列表中
dataframes.append(df)

# 合并所有 DataFrame
final_df = pd.concat(dataframes, ignore_index=True)

final_df
-----

import pandas as pd

# 加载数据集
file_path = '城市人口规模前 50.xlsx'
population_data = pd.read_excel(file_path)

# 显示数据的前几行以进行检查
population_data.head()
-----

# 使用内嵌模式，在 Jupyter Notebook 中显示图像
%matplotlib inline
-----

import matplotlib.pyplot as plt
-----

import os

# 清除 MPLBACKEND 环境变量
if 'MPLBACKEND' in os.environ:
    del os.environ['MPLBACKEND']

import matplotlib
matplotlib.use('TkAgg') # 再次尝试设置一个兼容的后端
-----

import pandas as pd

```

```

# 设置全局字体为中文，解决中文显示问题
plt.rcParams['font.sans-serif'] = ['SimHei'] # 使用 SimHei 字体以显示中文
plt.rcParams['axes.unicode_minus'] = False # 确保负号能够正确显示

# 加载数据文件
file_path = 'Q2_data.xlsx'
data = pd.read_excel(file_path)

# 显示数据结构和前几行数据，确保数据已正确加载
print("数据预览: ")
data.head()

-----
# 提取需要的指标名称列表，用于后续数据处理
features = [
    'AQI', '绿化覆盖率 (%)', '废水处理率 (%)', '废气处理率 (%)', '垃圾分类处理
率 (%)',
    '历史遗迹数量', '博物馆数量', '文化活动频次', '文化设施数量',
    '公共交通覆盖率 (%)', '线路密度 (km/km²)', '高速公路里程 (km)', '机场航班
数量',
    '年平均气温 (°C)', '年降水量 (mm)', '适宜旅游天数', '空气湿度 (%)',
    '餐馆数量', '特色美食数量', '美食活动频次'
]

# 将所有需要的指标列转换为数值类型，如果转换出错则设置为 NaN
for feature in features:
    data[feature] = pd.to_numeric(data[feature], errors='coerce')

# 删除包含 NaN 值的行，确保数据完整性
data.dropna(subset=features, inplace=True)

# 数据标准化，将各指标转换为标准正态分布
scaler = StandardScaler()
standardized_data = scaler.fit_transform(data[features])

# 将标准化后的数据转换回 DataFrame 格式，并添加城市列
standardized_data = pd.DataFrame(standardized_data, columns=features)
standardized_data['城市'] = data['来源城市']

# 设定每个指标的权重，用于综合评分的计算
weights = {
    'AQI': 0.05, '绿化覆盖率 (%)': 0.05, '废水处理率 (%)': 0.05, '废气处理
率 (%)': 0.05, '垃圾分类处理率 (%)': 0.05,
    '历史遗迹数量': 0.05, '博物馆数量': 0.05, '文化活动频次': 0.05, '文化设施

```

```

数量': 0.05,
    '公共交通覆盖率 (%)': 0.05, '线路密度 (km/km²)': 0.05, '高速公路里
程 (km)': 0.05, '机场航班数量': 0.05,
    '年平均气温 (°C)': 0.05, '年降水量 (mm)': 0.05, '适宜旅游天数': 0.05, '空
气湿度 (%)': 0.05,
    '餐馆数量': 0.05, '特色美食数量': 0.05, '美食活动频次': 0.05
}

# 根据设定的权重计算各指标的加权值
weighted_data = standardized_data.copy()
for col, weight in weights.items():
    weighted_data[col] = standardized_data[col] * weight

# 计算综合评分, 排除'城市'列
weighted_data['综合评分'] = weighted_data[features].sum(axis=1)

# 添加城市列
weighted_data['城市'] = data['来源城市']

# 按综合评分降序排序, 并选出前 50 个城市
top_50_cities = weighted_data.sort_values(by='综合评分
', ascending=False).head(50)

# 显示最令外国游客向往的前 50 个城市及其综合评分
print("最令外国游客向往的 50 个城市:")
top_50_cities[['城市', '综合评分']]
-----

import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from scipy.linalg import eig
import matplotlib.pyplot as plt
import warnings

# 忽略运行警告
warnings.filterwarnings("ignore", category=RuntimeWarning)

# 设置全局字体为中文, 解决中文显示问题
plt.rcParams['font.sans-serif'] = ['SimHei'] # 使用 SimHei 字体以显示中文
plt.rcParams['axes.unicode_minus'] = False # 确保负号能够正确显示

# 加载数据文件
file_path = 'Q2_Data2.xlsx'

```

```

data = pd.read_excel(file_path)

# 显示数据结构和前几行数据，确保数据已正确加载
print("数据预览: ")
data.head()

-----

# 定义需要分析的指标
features = [
    'AQI', '绿化覆盖率 (%)', '废水处理率 (%)', '废气处理率 (%)', '垃圾分类处理
率 (%)',
    '历史遗迹数量', '博物馆数量', '文化活动频次', '文化设施数量',
    '公共交通覆盖率 (%)', '线路密度 (km/km²)', '高速公路里程 (km)', '机场航班
数量',
    '年平均气温 (°C)', '年降水量 (mm)', '适宜旅游天数', '空气湿度 (%)',
    '餐馆数量', '特色美食数量', '美食活动频次'
]

# 将所有需要的指标列转换为数值类型，并处理错误
for feature in features:
    data[feature] = pd.to_numeric(data[feature], errors='coerce')

# 删除包含 NaN 的行
data.dropna(subset=features, inplace=True)

# 数据标准化
scaler = StandardScaler()
standardized_data = scaler.fit_transform(data[features])
standardized_data = pd.DataFrame(standardized_data, columns=features)
standardized_data['城市'] = data['来源城市']

-----

# 构建 AHP 判断矩阵并计算特征向量和特征值
AHP_matrix = np.array([
    [1, 1/3, 1/5, 1/2, 1/4],
    [3, 1, 1/3, 2, 1/2],
    [5, 3, 1, 4, 2],
    [2, 1/2, 1/4, 1, 1/3],
    [4, 2, 1/2, 3, 1]
])
eig_values, eig_vectors = eig(AHP_matrix)
max_eig_value = np.max(np.real(eig_values))
max_eig_vector = np.real(eig_vectors[:, np.argmax(np.real(eig_values))])
weights = max_eig_vector / np.sum(max_eig_vector)
print("权重向量:", weights)

```

```

# 主成分分析（PCA）进行降维
categories = {
    '环境环保': features[:5],
    '人文底蕴': features[5:9],
    '交通便利': features[9:13],
    '气候': features[13:17],
    '美食': features[17:]
}

pca = PCA(n_components=1)
pca_data = pd.DataFrame(index=standardized_data.index)
for category, features in categories.items():
    pca_result = pca.fit_transform(standardized_data[features])
    pca_data[category] = pca_result.flatten()

pca_data['城市'] = data['来源城市']

-----

# 基于熵权法的 TOPSIS 进行综合评价
def entropy_weight(data):
    k = 1.0 / np.log(len(data))
    data = data / (data.sum(axis=0) + 1e-10) # 归一化并避免出现 0 值
    data = data * np.log(data + 1e-10) # 避免取对数时出现无效值
    entropy = -k * (data.sum(axis=0))
    weight = (1 - entropy) / (1 - entropy).sum()
    return weight

# 计算权重
topsis_weights = entropy_weight(pca_data.drop(columns=['城市']))

# TOPSIS 综合评价
def topsis(data, weights):
    data = np.array(data)
    weights = np.array(weights)
    weights /= np.sum(weights) # 确保权重之和为 1

    if data.shape[1] != len(weights):
        raise ValueError("数据的特征数量必须与权重向量的长度相匹配。")

    norm_data = data / np.sqrt((data ** 2).sum(axis=0))
    ideal_best = np.max(norm_data, axis=0)
    ideal_worst = np.min(norm_data, axis=0)

```

```

        distance_to_best = np.sqrt(((norm_data - ideal_best) ** 2 * weights).sum(axis=1))
        distance_to_worst = np.sqrt(((norm_data - ideal_worst) ** 2 * weights).sum(axis=1))

        score = distance_to_worst / (distance_to_best + distance_to_worst)
        return score

    try:
        topsis_scores = topsis(pca_data.drop(columns=['城市']), topsis_weights)
        pca_data['综合评分'] = topsis_scores
        print("成功计算 TOPSIS 得分。")
    except Exception as e:
        print("发生错误: ", e)

    -----
    # 按综合评分排序，选出前 50 个城市
    top_50_cities = pca_data.sort_values(by='综合评分', ascending=False).head(50)
    print("最令外国游客向往的 50 个城市:")
    top_50_cities[['城市', '综合评分']]
    -----

    import matplotlib.pyplot as plt
    import seaborn as sns
    %matplotlib inline
    sns.set(style="whitegrid")
    plt.rcParams['font.sans-serif'] = ['SimHei'] # 使用 SimHei 字体以显示中文
    plt.rcParams['axes.unicode_minus'] = False # 确保负号能够正确显示
    plt.figure(figsize=(14, 10))

    sns.barplot(
        x="综合评分",
        y="城市",
        data=top_50_cities,
        palette="viridis"
    )

    plt.xlabel('综合评分', fontsize=14)
    plt.ylabel('城市', fontsize=14)
    plt.title('最令外国游客向往的 50 个城市综合评分', fontsize=16)

    # 翻转 Y 轴，城市按综合评分从高到低排列
    plt.gca().invert_yaxis()
    plt.show()

```

```

-----
# 绘制热力图
plt.figure(figsize=(14, 10))
heatmap_data = standardized_data.loc[top_50_cities.index, features]
sns.heatmap(heatmap_data, cmap='viridis', cbar_kws={'label': '标准化得分'},
xticklabels=features, yticklabels=top_50_cities['城市'], linewidths=0.5)
plt.title('前 50 个城市在不同指标上的得分情况', fontsize=16)
plt.show()

-----

cities_coords = pd.read_csv('最令外国游客向往的 50 个城市.csv')
merged_data = pd.merge(top_50_cities, cities_coords, on='城市', how='left')
# 绘制城市位置图
plt.figure(figsize=(10, 8))
plt.scatter(merged_data['经度'], merged_data['纬度'], c=merged_data['综合评分'],
cmap='viridis', s=100)
for i, city in enumerate(merged_data['城市']):
    plt.text(merged_data['经度'].iloc[i], merged_data['纬度'].iloc[i], city, fontsize=9)
plt.colorbar(label='综合评分')
plt.xlabel('经度')
plt.ylabel('纬度')
plt.title('前 50 个城市在中国地图上的位置')
plt.show()

-----

import pandas as pd

# 创建包含前 50 个城市及其经纬度信息的字典
city_data = {
    '城市': [
        '福州', '大理', '恩施', '潜江', '咸宁', '三亚', '惠州', '朔州', '常德',
        '攀枝花',
        '晋城', '琼海', '可克达拉', '成都', '临高', '台州', '定西', '楚雄州',
        '嘉峪关', '扬州',
        '雄安新区', '济源', '北京', '儋州', '呼和浩特', '中山', '贵阳', '丽江',
        '安庆', '五家渠',
        '南充', '潍坊', '贺州', '邵阳', '通化', '杭州', '泸州', '汕尾', '济南',
        '广州',
        '河源', '衡阳', '马鞍山', '迪庆', '鄂尔多斯', '清远', '邢台', '七台河',
        '铁岭', '海口'
    ],
    '纬度': [
        26.0745, 25.6065, 30.2722, 30.4268, 29.8413, 18.2528, 23.1103, 39.3316, 29.0316, 26.5823,
        35.4907, 19.2592, 43.6836, 30.5728, 19.9186, 28.6564, 35.5807, 25.

```

```

0367, 39.8023, 32.3929,
    38.9990, 35.0671, 39.9042, 19.5209, 40.8426, 22.5159, 26.6477, 26.
8721, 30.5086, 44.1674,
    30.8375, 36.7068, 24.4033, 27.2333, 41.7212, 30.2741, 28.8724, 22.
7877, 36.6512, 23.1291,
    23.7338, 26.8930, 31.6765, 27.8268, 39.6085, 23.6820, 37.0700, 45.
7700, 42.2860, 20.0442
    ],
    '经度': [
        119.2965, 100.2676, 109.4880, 112.8990, 114.3228, 109.5119, 114.41
68, 112.4320, 111.6537, 101.7160,
        112.8513, 110.4745, 81.1640, 104.0668, 109.6873, 121.4208, 104.626
6, 101.5460, 98.2901, 119.4210,
        115.8540, 112.5876, 116.4074, 109.5820, 111.7490, 113.3926, 106.63
02, 100.2296, 117.0472, 87.5269,
        106.0806, 119.1035, 111.5672, 111.4672, 125.9406, 120.1551, 105.44
79, 115.3751, 117.1201, 113.2644,
        114.7074, 112.5707, 118.5065, 99.7071, 109.7813, 113.0118, 114.504
0, 131.0129, 123.8443, 110.1983
    ]
}

# 使用字典创建 DataFrame
df = pd.DataFrame(city_data)

# 将 DataFrame 保存为 CSV 文件
df.to_csv('最令外国游客向往的 50 个城市.csv', index=False)

# 输出提示信息表示 CSV 文件已成功生成
print("CSV 文件已生成")
-----

import pandas as pd

# 设置全局字体为中文，解决中文显示问题
plt.rcParams['font.sans-serif'] = ['SimHei'] # 使用 SimHei 字体以显示中文
plt.rcParams['axes.unicode_minus'] = False # 确保负号能够正确显示

# 加载数据文件
file_path = '最令外国游客向往的 50 个城市.csv'
data = pd.read_csv(file_path)

# 显示数据结构和前几行数据，确保数据已正确加载
print("数据预览: ")
data.head()

```

```

-----
import pandas as pd

# 定义前 50 个城市列表
top_50_cities = [
    '福州', '大理', '恩施', '潜江', '咸宁', '三亚', '惠州', '朔州', '常德', '
攀枝花',
    '晋城', '琼海', '可克达拉', '成都', '临高', '台州', '定西', '楚雄州', '嘉
峪关', '扬州',
    '雄安新区', '济源', '北京', '儋州', '呼和浩特', '中山', '贵阳', '丽江', '
安庆', '五家渠',
    '南充', '潍坊', '贺州', '邵阳', '通化', '杭州', '泸州', '汕尾', '济南', '
广州',
    '河源', '衡阳', '马鞍山', '迪庆', '鄂尔多斯', '清远', '邢台', '七台河', '
铁岭', '海口'
]

file_path = 'Q3_data.xlsx'
data = pd.read_excel(file_path)

# 提取前 50 个城市的信息
filtered_data = data[data['来源城市'].isin(top_50_cities)]

# 删除“建议游玩时间”为空值的行
filtered_data = filtered_data.dropna(subset=['建议游玩时间'])

# 显示提取并删除后的信息
print("前 50 个城市的信息（删除“建议游玩时间”为空值的行）:")

# 保存提取并删除后的信息到新的 CSV 文件
filtered_data.to_csv('Q3_data2.csv', index=False)
print("提取并删除后的信息已保存到 'Q3_data2.csv'")

-----
filtered_data.head()

-----
import pandas as pd
import numpy as np

-----
# 读取 CSV 数据文件
df = pd.read_csv('Q3_data2.csv')

-----
highest Rated = df[df.groupby('来源城市')['评分'].transform('max') == df['评
分']]
-----

```

```

random_selected = highest_rated.groupby('来源城市')
).apply(lambda x: x.sample(1)).reset_index(drop=True)
-----
# 保存结果到新的 CSV 文件
random_selected.to_csv('Q3_data3.csv', index=False)
# 输出提示信息表示 CSV 文件已成功保存
print("已保存每个城市评分最高且随机选择的样本到 'Q3_data3.csv'")
-----

import pandas as pd

# 读取 CSV 文件
file_path = 'Q3_data3.csv'
data = pd.read_csv(file_path)

# 城市经纬度数据
city_data = {
    '城市': [
        '福州', '大理', '恩施', '潜江', '咸宁', '三亚', '惠州', '朔州', '常德',
        '攀枝花',
        '晋城', '琼海', '可克达拉', '成都', '临高', '台州', '定西', '楚雄州',
        '嘉峪关', '扬州',
        '雄安新区', '济源', '北京', '儋州', '呼和浩特', '中山', '贵阳', '丽江',
        '安庆', '五家渠',
        '南充', '潍坊', '贺州', '邵阳', '通化', '杭州', '泸州', '汕尾', '济南',
        '河源', '衡阳',
        '马鞍山', '迪庆', '鄂尔多斯', '清远', '邢台', '七台河', '铁岭', '海口'
    ],
    '纬度': [
        26.0745, 25.6065, 30.2722, 30.4268, 29.8413, 18.2528, 23.1103, 39.
        3316, 29.0316, 26.5823,
        35.4907, 19.2592, 43.6836, 30.5728, 19.9186, 28.6564, 35.5807, 25.
        0367, 39.8023, 32.3929,
        38.9990, 35.0671, 39.9042, 19.5209, 40.8426, 22.5159, 26.6477, 26.
        8721, 30.5086, 44.1674,
        30.8375, 36.7068, 24.4033, 27.2333, 41.7212, 30.2741, 28.8724, 22.
        7877, 36.6512, 23.7338, 26.8930,
        31.6765, 27.8268, 39.6085, 23.6820, 37.0700, 45.7700, 42.2860, 20.
        0442
    ],
    '经度': [
        119.2965, 100.2676, 109.4880, 112.8990, 114.3228, 109.5119, 114.41
        68, 112.4320, 111.6537, 101.7160,
        112.8513, 110.4745, 81.1640, 104.0668, 109.6873, 121.4208, 104.626
        6, 101.5460, 98.2901, 119.4210,

```

```

        115.8540, 112.5876, 116.4074, 109.5820, 111.7490, 113.3926, 106.63
02, 100.2296, 117.0472, 87.5269,
        106.0806, 119.1035, 111.5672, 111.4672, 125.9406, 120.1551, 105.44
79, 115.3751, 117.1201, 114.7074,
        112.5707, 118.5065, 99.7071, 109.7813, 113.0118, 114.5040, 131.012
9, 123.8443, 110.1983
    ]
}

# 将城市经纬度数据转换为 DataFrame
city_df = pd.DataFrame(city_data)

-----

city_df.head()

-----

# 将城市经纬度信息合并到原始数据中
merged_data = pd.merge(data, city_df, left_on='来源城市', right_on='城市',
, how='left')
# 删除多余的“城市”列
merged_data.drop(columns=['城市'], inplace=True)
# 保存新的 CSV 文件
output_file_path = 'Q3_data4.csv'
merged_data.to_csv(output_file_path, index=False)
# 输出提示信息表示 CSV 文件已成功生成
print(f"已成功将经纬度信息添加到 CSV 文件并保存到 {output_file_path}")

-----

merged_data.head()

-----

import pandas as pd
import numpy as np
from geopy.distance import geodesic
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False

-----

# 读取 CSV 文件
file_path = 'Q3_data4.csv'
try:
    data = pd.read_csv(file_path, encoding='utf-8')
except UnicodeDecodeError:
    data = pd.read_csv(file_path, encoding='GBK')

# 更改列名
data.columns = ['名字', '开放时间', '评分', '建议游玩时间', '门票', '来源城市', '纬度', '经度']

```

```

data.head()

-----

data
-----

# 计算两个经纬度点之间的距离
def calculate_distance(lat1, lon1, lat2, lon2):
    return geodesic((lat1, lon1), (lat2, lon2)).kilometers

# 定义高铁速度和费用参数
train_speed_kmh = 300
cost_per_km = 0.555
daily_rest_hours = 6 # 进一步减少休息时间

# 初始化路径规划变量
starting_city = '广州'
time_limit_hours = 200 # 进一步增加总时间限制
effective_hours = time_limit_hours - 7 * daily_rest_hours # 有效游玩时间
cities = data['来源城市'].tolist()
visited_cities = [starting_city]
remaining_cities = [city for city in cities if city != starting_city]
current_city = starting_city
total_time_spent = 0
total_cost = 0
stay_durations = [0] # 用于记录每个城市停留时间，广州起点停留时间为0
-----
# 创建字典保存城市的经纬度和评分
city_info = {
    row['来源城市']: {'coords': (row['纬度'], row['经度']), 'score': row['评分'], 'suggested_time': row['建议游玩时间']}
    for idx, row in data.iterrows()
}

# 确认字典中存在广州的信息
if '广州' not in city_info:
    print("错误: '广州'不在字典 city_info 中, 请检查数据。")
else:
    print("'广州'在字典 city_info 中, 继续处理。")

# 启发式算法: 每次选择最近的城市
while remaining_cities and total_time_spent < effective_hours:
    min_distance = float('inf')
    next_city = None
    for city in remaining_cities:
        distance = calculate_distance(

```

```

        city_info[current_city]['coords'][0], city_info[current_city][
'coords'][1],
        city_info[city]['coords'][0], city_info[city]['coords'][1]
    )
    if distance < min_distance:
        min_distance = distance
        next_city = city

    if next_city:
        travel_time = min_distance / train_speed_kmh
        suggested_time = city_info[next_city]['suggested_time']
        if total_time_spent + travel_time + suggested_time > effective_hou
rs:
            print(f"时间限制: 总时间
({total_time_spent + travel_time + suggested_time})超过有效时间
({effective_hours})")
            break
        total_time_spent += travel_time + suggested_time
        total_cost += min_distance * cost_per_km
        visited_cities.append(next_city)
        stay_durations.append(suggested_time)
        remaining_cities.remove(next_city)
        current_city = next_city
    else:
        print("没有找到下一个城市")
        break

    print(f"当前城市: {current_city}, 总时间: {total_time_spent}, 总费用:
{total_cost}")

-----
# 输出结果
total_scores = sum(city_info[city]['score'] for city in visited_cities)
result = {
    'Visited Cities': visited_cities,
    'Stay Durations': stay_durations,
    'Total Time Spent (hours)': total_time_spent + len(visited_cities) * d
aily_rest_hours,
    'Total Cost (RMB)': total_cost,
    'Total Cities Visited': len(visited_cities),
    'Total Scores': total_scores
}

# 创建结果 DataFrame

```

```

result_df = pd.DataFrame([result])
-----
result_df
-----
# 绘制路线图
latitudes = [city_info[city]['coords'][0] for city in visited_cities]
longitudes = [city_info[city]['coords'][1] for city in visited_cities]

plt.figure(figsize=(12, 10))
plt.plot(longitudes, latitudes, marker='o', linestyle='--', color='b')
for i, city in enumerate(visited_cities):
    plt.text(longitudes[i], latitudes[i], f'{i+1}. {city}', fontsize=9, ha
='right')
plt.xlabel('经度')
plt.ylabel('纬度')
plt.title('旅行路线')
plt.grid(True)
plt.show()
-----

import seaborn as sns
import matplotlib.pyplot as plt
# 设置 Seaborn 样式
sns.set(style="whitegrid")
# 设置全局字体为中文
plt.rcParams['font.sans-serif'] = ['SimHei'] # 使用 SimHei 字体
plt.rcParams['axes.unicode_minus'] = False # 正确显示负号
# 可视化每个城市的停留时间
plt.figure(figsize=(12, 8))
bar_plot = sns.barplot(x=visited_cities, y=stay_durations, palette="viridi
s")

plt.xlabel('城市', fontsize=14)
plt.ylabel('停留时间（小时）', fontsize=14)
plt.title('每个城市的停留时间', fontsize=16)
plt.xticks(rotation=90, fontsize=12)
plt.yticks(fontsize=12)
# 添加数据标签
for index, value in enumerate(stay_durations):
    bar_plot.text(index, value + 0.1, f'{value:.1f}', ha='center', va='bot
tom', fontsize=10)

plt.show()
-----
# 可视化每个城市的评分

```

```

scores = [city_info[city]['score'] for city in visited_cities]
plt.figure(figsize=(12, 8))
bar_plot = sns.barplot(x=visited_cities, y=scores, palette="coolwarm")

plt.xlabel('城市', fontsize=14)
plt.ylabel('评分', fontsize=14)
plt.title('访问城市的评分', fontsize=16)
plt.xticks(rotation=90, fontsize=12)
plt.yticks(fontsize=12)

# 添加数据标签
for index, value in enumerate(scores):
    bar_plot.text(index, value + 0.1, f'{value:.1f}', ha='center', va='bottom', fontsize=10)

plt.show()

-----
from geopy.distance import geodesic
import geopandas as gpd
from shapely.geometry import Point, LineString
import contextily as ctx
-----

# 读取 CSV 文件
file_path = 'Q3_data4.csv'
try:
    data = pd.read_csv(file_path, encoding='utf-8')
except UnicodeDecodeError:
    data = pd.read_csv(file_path, encoding='GBK')

# 更改列名
data.columns = ['名字', '开放时间', '评分', '建议游玩时间', '门票', '来源城市', '纬度', '经度']
data.head()

-----

# 计算两个经纬度点之间的距离
def calculate_distance(lat1, lon1, lat2, lon2):
    return geodesic((lat1, lon1), (lat2, lon2)).kilometers

# 定义高铁速度和费用参数
train_speed_kmh = 300
cost_per_km = 0.555
daily_rest_hours = 10

```

```

# 初始化路径规划变量
starting_city = '广州'
time_limit_hours = 144
effective_hours = time_limit_hours - 7 * daily_rest_hours # 有效游玩时间
cities = data['来源城市'].tolist()
visited_cities = [starting_city]
remaining_cities = [city for city in cities if city != starting_city]
current_city = starting_city
total_time_spent = 0
total_cost = 0
total_ticket_cost = 0 # 门票总费用
total_train_cost = 0 # 高铁总费用
stay_durations = [0] # 用于记录每个城市停留时间，广州起点停留时间为0
-----

# 创建字典保存城市的经纬度和评分
city_info = {
    row['来源城市']: {'coords': (row['纬度'], row['经度']), 'score': row['评分'], 'suggested_time': row['建议游玩时间'], 'ticket_price': row['门票']}
    for idx, row in data.iterrows()
}

# 启发式算法：每次选择费用最低的城市（考虑距离和门票）
while remaining_cities and total_time_spent < effective_hours:
    min_cost = float('inf')
    next_city = None
    for city in remaining_cities:
        distance = calculate_distance(
            city_info[current_city]['coords'][0], city_info[current_city]['coords'][1],
            city_info[city]['coords'][0], city_info[city]['coords'][1]
        )
        travel_time = distance / train_speed_kmh
        travel_cost = distance * cost_per_km
        ticket_price = city_info[city]['ticket_price']
        total_cost_for_city = travel_cost + ticket_price
        if total_cost_for_city < min_cost:
            min_cost = total_cost_for_city
            next_city = city

    if next_city:
        distance = calculate_distance(
            city_info[current_city]['coords'][0], city_info[current_city]['coords'][1],
            city_info[next_city]['coords'][0], city_info[next_city]['coord

```

```

s']][1]
    )
    travel_time = distance / train_speed_kmh
    suggested_time = city_info[next_city]['suggested_time']
    if total_time_spent + travel_time + suggested_time > effective_hou
rs:
        break
    total_time_spent += travel_time + suggested_time
    travel_cost = distance * cost_per_kmh
    ticket_price = city_info[next_city]['ticket_price']
    total_cost += travel_cost + ticket_price
    total_train_cost += travel_cost
    total_ticket_cost += ticket_price
    visited_cities.append(next_city)
    stay_durations.append(suggested_time)
    remaining_cities.remove(next_city)
    current_city = next_city

-----
# 输出结果
total_scores = sum(city_info[city]['score'] for city in visited_cities)
total_travel_time = total_time_spent + len(visited_cities) * daily_rest_ho
urs

print(f"Visited Cities: {visited_cities}")
print(f"Stay Durations: {stay_durations}")
print(f"Total Time Spent (hours): {total_travel_time}")
print(f"Total Cost (RMB): {total_cost}")
print(f"Total Ticket Cost (RMB): {total_ticket_cost}")
print(f"Total Train Cost (RMB): {total_train_cost}")
print(f"Total Cities Visited: {len(visited_cities)}")
print(f"Total Scores: {total_scores}")

result = {
    'Visited Cities': visited_cities,
    'Stay Durations': stay_durations,
    'Total Time Spent (hours)': total_travel_time,
    'Total Cost (RMB)': total_cost,
    'Total Ticket Cost (RMB)': total_ticket_cost,
    'Total Train Cost (RMB)': total_train_cost,
    'Total Cities Visited': len(visited_cities),
    'Total Scores': total_scores
}

# 创建结果 DataFrame

```

```

result_df = pd.DataFrame([result])
-----
result_df
-----
# 使用 geopandas 和 contextily 绘制路线图
# 创建 GeoDataFrame
gdf = gpd.GeoDataFrame(
    visited_cities,
    geometry=[Point(city_info[city]['coords'][1], city_info[city]['coords'
][0]) for city in visited_cities],
    columns=['City']
)
gdf.set_crs(epsg=4326, inplace=True) # 设置坐标参考系统为 WGS84

# 创建路线
route = LineString(gdf.geometry.tolist())

# 绘制路线和点
fig, ax = plt.subplots(1, 1, figsize=(10, 10))
gdf.plot(ax=ax, color='blue', edgecolor='black', markersize=50)
gpd.GeoSeries(route).plot(ax=ax, color='red')
for x, y, label in zip(gdf.geometry.x, gdf.geometry.y, visited_cities):
    ax.text(x, y, label, fontsize=12)

# 添加底图
ctx.add_basemap(ax, crs=gdf.crs.to_string(), source=ctx.providers.OpenStre
etMap.Mapnik)

plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Travel Route')
plt.show()
-----

import seaborn as sns
import matplotlib.pyplot as plt
# 设置 Seaborn 样式
sns.set(style="whitegrid")
# 设置全局字体为中文
plt.rcParams['font.sans-serif'] = ['SimHei'] # 使用 SimHei 字体
plt.rcParams['axes.unicode_minus'] = False # 正确显示负号
# 可视化每个城市的停留时间
plt.figure(figsize=(12, 8))
bar_plot = sns.barplot(x=visited_cities, y=stay_durations, palette="viridi
s")

```

```

plt.xlabel('城市', fontsize=14)
plt.ylabel('停留时间 (小时)', fontsize=14)
plt.title('每个城市的停留时间', fontsize=16)
plt.xticks(rotation=90, fontsize=12)
plt.yticks(fontsize=12)
# 添加数据标签
for index, value in enumerate(stay_durations):
    bar_plot.text(index, value + 0.1, f'{value:.1f}', ha='center', va='bottom', fontsize=10)

plt.show()
-----

import pandas as pd

csv_file_path = 'Q5_data.csv'
try:
    csv_data = pd.read_csv(csv_file_path, encoding='utf-8')
except UnicodeDecodeError:
    csv_data = pd.read_csv(csv_file_path, encoding='GBK')
excel_file_path = 'Q5_data1.xlsx'
excel_data = pd.read_excel(excel_file_path)
csv_data.columns = ['来源城市', '经度', '纬度']
merged_data = pd.merge(excel_data, csv_data, on='来源城市', how='left')
-----
merged_data.head()
-----

# 保存修改后的数据到新的 Excel 文件
output_file_path = 'Q5_data2.xlsx'
merged_data.to_excel(output_file_path, index=False)
print("合并后的数据已保存到: ", output_file_path)
-----

import pandas as pd
from geopy.distance import geodesic
import matplotlib.pyplot as plt
import geopandas as gpd
from shapely.geometry import Point, LineString
import contextily as ctx
-----

plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
excel_file_path = 'Q5_data2.xlsx'
data = pd.read_excel(excel_file_path)
# 过滤出所有山景的景点数据

```

```

mountain_data = data[data['名字'].str.contains('山')]
# 根据评分选择每个城市评分最高的山景点
top_mountain_per_city = mountain_data.loc[mountain_data.groupby('来源城市')
['评分'].idxmax()]
-----
# 定义高铁速度和费用参数
train_speed_kmh = 300 # 高铁速度（公里/小时）
cost_per_km = 0.555 # 每公里费用（元）
daily_rest_hours = 10 # 每日休息时间（小时）
time_limit_hours = 144 # 总时间限制（小时）
effective_hours = time_limit_hours - 7 * daily_rest_hours # 有效游玩时间（小时）

# 初始化路径规划变量
starting_city = top_mountain_per_city.iloc[0]['来源城市']
visited_cities = [starting_city]
remaining_cities = top_mountain_per_city['来源城市'].tolist()
remaining_cities.remove(starting_city)
current_city = starting_city
total_time_spent = 0
total_cost = 0
total_ticket_cost = 0 # 门票总费用
total_train_cost = 0 # 高铁总费用
stay_durations = [0] # 用于记录每个城市停留时间，起点停留时间为0
-----
# 创建字典保存城市的经纬度和评分
city_info = {
    row['来源城市']: {'coords': (row['经度'], row['纬度']), 'score': row['评分'], 'suggested_time': row['建议游玩时间'], 'ticket_price': row['门票']}
    for idx, row in top_mountain_per_city.iterrows()
}

# 启发式算法：每次选择费用最低的城市（考虑距离和门票）
while remaining_cities and total_time_spent < effective_hours:
    min_cost = float('inf')
    next_city = None
    for city in remaining_cities:
        distance = geodesic(city_info[current_city]['coords'], city_info[city]['coords']).kilometers
        travel_time = distance / train_speed_kmh
        travel_cost = distance * cost_per_km
        ticket_price = city_info[city]['ticket_price']
        total_cost_for_city = travel_cost + ticket_price
        if total_cost_for_city < min_cost:

```

```

        min_cost = total_cost_for_city
        next_city = city

    if next_city:
        distance = geodesic(city_info[current_city]['coords'], city_info[next_city]['coords']).kilometers
        travel_time = distance / train_speed_kmh
        suggested_time = city_info[next_city]['suggested_time']
        if total_time_spent + travel_time + suggested_time > effective_hours:
            break
        total_time_spent += travel_time + suggested_time
        travel_cost = distance * cost_per_km
        ticket_price = city_info[next_city]['ticket_price']
        total_cost += travel_cost + ticket_price
        total_train_cost += travel_cost
        total_ticket_cost += ticket_price
        visited_cities.append(next_city)
        stay_durations.append(suggested_time)
        remaining_cities.remove(next_city)
        current_city = next_city

    -----
    # 输出结果
    total_scores = sum(city_info[city]['score'] for city in visited_cities)
    total_travel_time = total_time_spent + len(visited_cities) * daily_rest_hours

    result = {
        'Visited Cities': visited_cities,
        'Stay Durations': stay_durations,
        'Total Time Spent (hours)': total_travel_time,
        'Total Cost (RMB)': total_cost,
        'Total Ticket Cost (RMB)': total_ticket_cost,
        'Total Train Cost (RMB)': total_train_cost,
        'Total Cities Visited': len(visited_cities),
        'Total Scores': total_scores
    }

    print(f"Visited Cities: {visited_cities}")
    print(f"Stay Durations: {stay_durations}")
    print(f"Total Time Spent (hours): {total_travel_time}")
    print(f"Total Cost (RMB): {total_cost}")
    print(f"Total Ticket Cost (RMB): {total_ticket_cost}")
    print(f"Total Train Cost (RMB): {total_train_cost}")
    print(f"Total Cities Visited: {len(visited_cities)}")

```

```

print(f"Total Scores: {total_scores}")

# 创建结果 DataFrame
result_df = pd.DataFrame([result])
-----
result_df
-----

gdf = gpd.GeoDataFrame(
    visited_cities,
    geometry=[Point(city_info[city]['coords'][1], city_info[city]['coords'
][0]) for city in visited_cities],
    columns=['City']
)
gdf.set_crs(epsg=4326, inplace=True)

# 创建路线
route = LineString(gdf.geometry.tolist())

# 绘制路线和点
fig, ax = plt.subplots(1, 1, figsize=(10, 10))
gdf.plot(ax=ax, color='blue', edgecolor='black', markersize=50)
gpd.GeoSeries(route).plot(ax=ax, color='red')
for x, y, label in zip(gdf.geometry.x, gdf.geometry.y, visited_cities):
    ax.text(x, y, label, fontsize=12)

ctx.add_basemap(ax, crs=gdf.crs.to_string(), source=ctx.providers.OpenStre
etMap.Mapnik)

plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Travel Route')
plt.show()
-----

import seaborn as sns
import matplotlib.pyplot as plt
# 设置 Seaborn 样式
sns.set(style="whitegrid")
# 设置全局字体为中文
plt.rcParams['font.sans-serif'] = ['SimHei'] # 使用 SimHei 字体
plt.rcParams['axes.unicode_minus'] = False # 正确显示负号
# 可视化每个城市的停留时间
plt.figure(figsize=(12, 8))
bar_plot = sns.barplot(x=visited_cities, y=stay_durations, palette="viridi
s")

```

```
plt.xlabel('城市', fontsize=14)
plt.ylabel('停留时间（小时）', fontsize=14)
plt.title('每个城市的停留时间', fontsize=16)
plt.xticks(rotation=90, fontsize=12)
plt.yticks(fontsize=12)
# 添加数据标签
for index, value in enumerate(stay_durations):
    bar_plot.text(index, value + 0.1, f'{value:.1f}', ha='center', va='bottom', fontsize=10)

plt.show()
```

周铠然 · 仅供求职展示