# 第十五章 Trae智能分析

随着大模型的兴起，尤其DeepSeek的出现，彻底改变了人们工作方式，对于80%的程序员来说，Trae彻底改变了他们编写代码的方式。

Trae 是字节跳动于 2025 年 3 月 3 日正式推出的国内首个 AI 原生集成开发环境（AI IDE）。作为中国首个实现端到端开发能力的 AI 编程工具，Trae 通过深度整合人工智能技术，为开发者提供了从项目构建到代码优化的全流程智能协作体验，支持切换至满血版 DeepSeek-R1 和 V3 模型。

TRAE与 AI 深度集成，提供智能问答、代码自动补全以及基于智能体的 AI 自动编程能力。使用 TRAE 开发项目时，可以与 AI 灵活协作，提升开发效率。目前TRAE CN版本为 v2.0.11 。

## 15.1安装Trae

同Anaconda一样，选择跟自己机器匹配的版本下载，如图15-1所示。Trae官方下载网址为https://www.trae.com.cn/download。

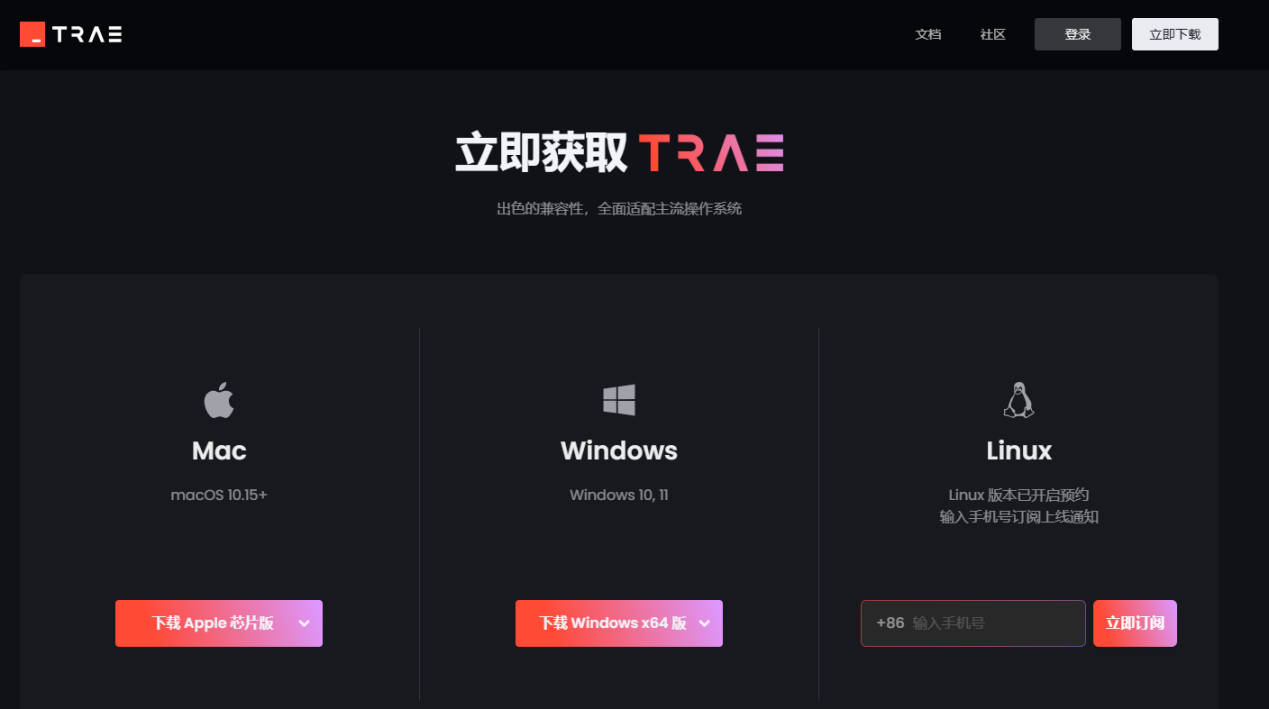


图15-1 Trae下载界面

Trae 首次启动时，需要联网注册，注册成功后可进入主界面。点击“打开文件”，如打开D:\trae\yubg1目录下python代码文件yubg.py，如图15-2所示。



图15-2 首次启动界面

点击图中“打开文件”，再点击“打开”即出现图15-3。

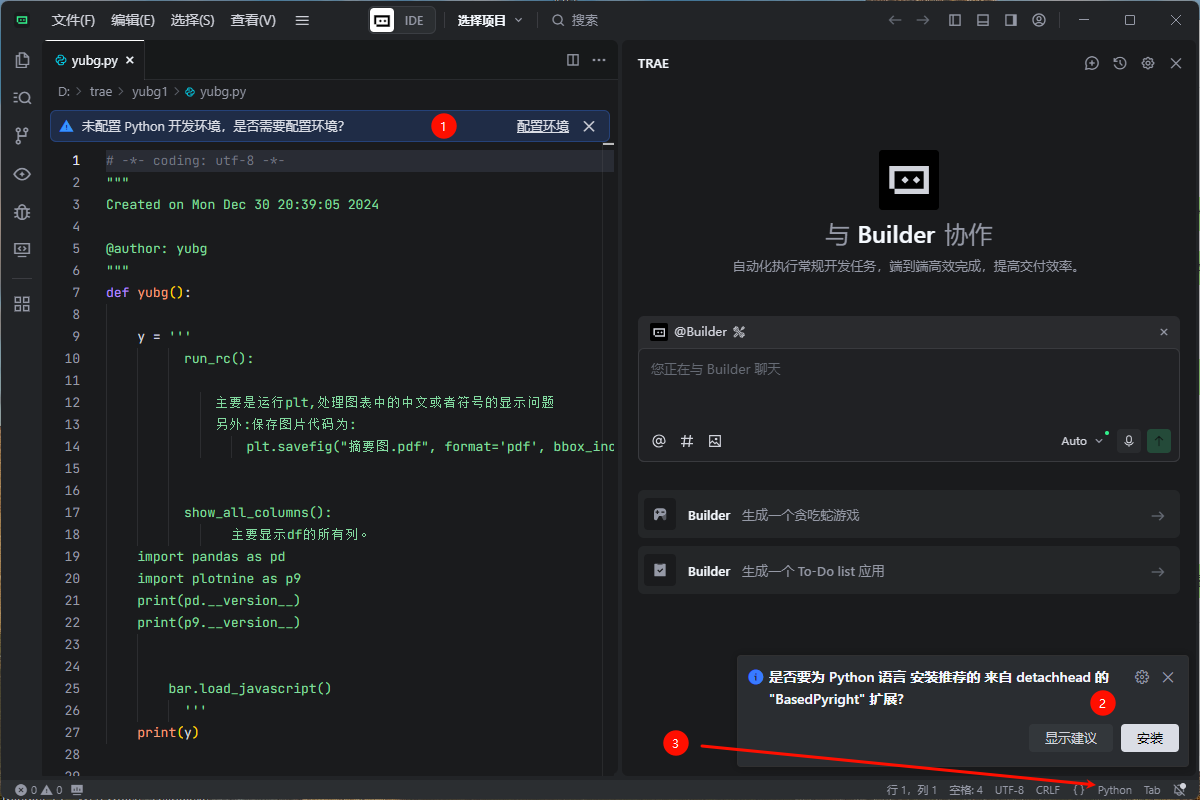


图15-3 配置环境

在运行代码前需要先配置环境，点击图15-3中的即可安装配置，或者点击安装扩展。之后可点击位置，弹出选择交互运行环境，如图15-4。如此处选择Python 3.11.5 ('base')E:soft-app\anaconda\python.exe。

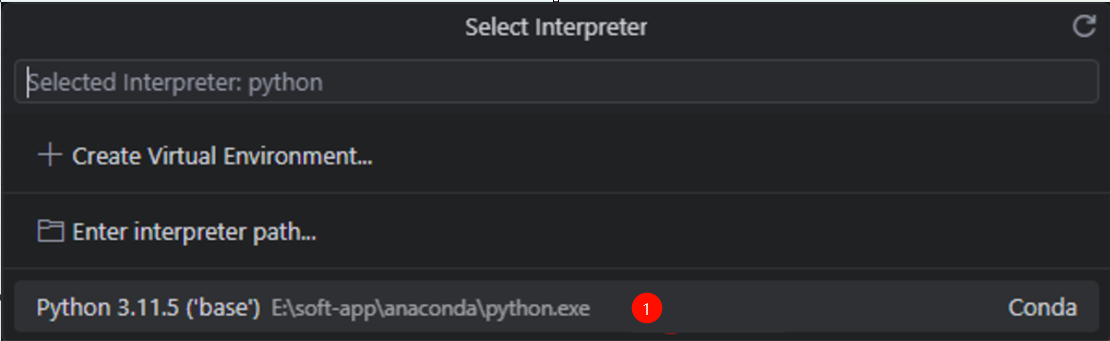


图15-4 交互环境设置

配置好运行环境后，在图15-5的处出现运行按钮，即可执行代码，结果出现在处，当出现有错误时，选择错误，会出现处的提示信息，点击或者按Ctrl+U即可在处出现添加错误信息，再点击处的AUTO旁的执行按钮，让Trae自行处理错误，并按照提示操作即可。

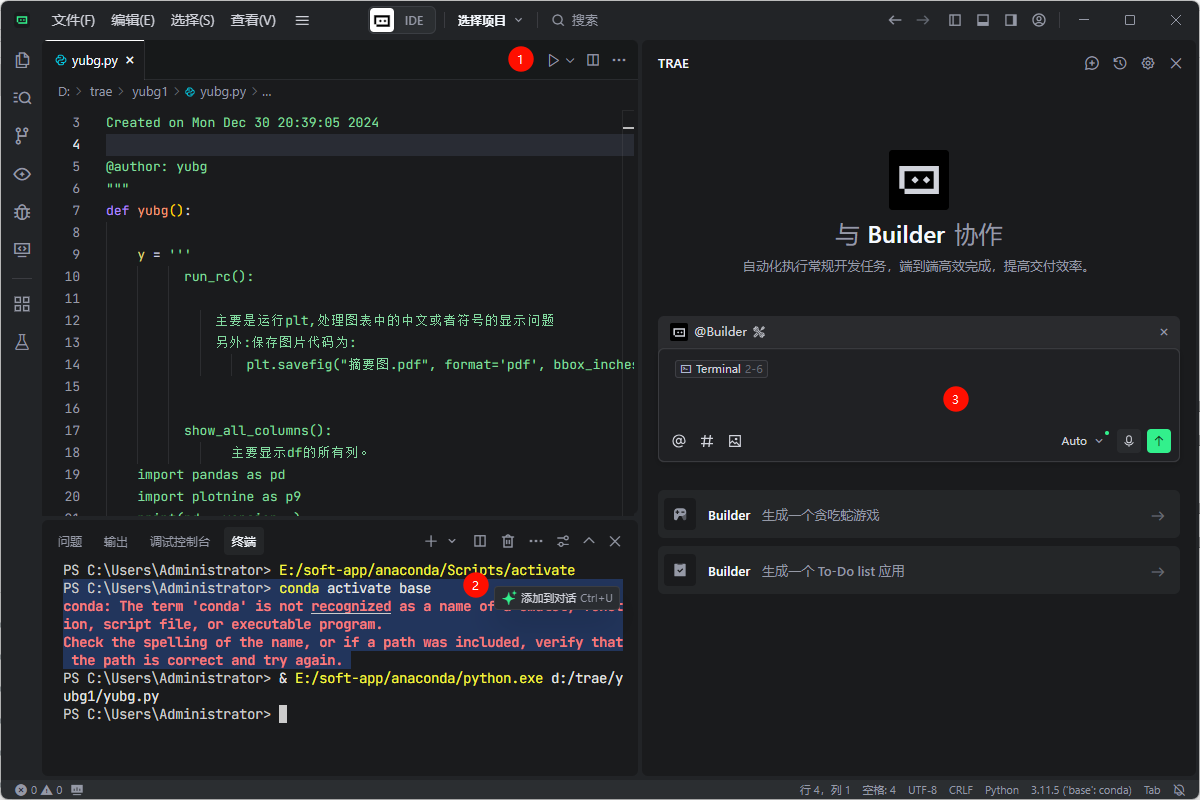


图15-5 执行代码并处理错误

图15-5中的Auto mode为默认启用模式。TRAE Auto 模式，通过自动化模型调度，有效降低使用成本、显著提升响应效率、简化操作流程，助力开发者将精力专注于核心开发任务本身。针对 TRAE 中国版 及 TRAE 国际版 Free 用户，Auto 会基于效果和速度自动选型，优先不排队的模型。如果没有特定模型偏好，追求省心、高效、省钱的综合体验时，Auto 模式是理想选择，它可以根据任务复杂度、响应速度与模型可用性智能选型。

如图15-5所示界面中，右侧为AI对话区，中间为代码显示运行区，左侧为应用功能菜单栏和文件管理区。在右侧AI对话区内可以输入工作任务，这跟DeepSeek或者豆包等AI工具使用一致。如果在代码区想修改代码可以有两种方式，第一直接在代码区自己手动修改代码，如果不会修改代码，也可以让AI替我们修改，只需要说出目标即可，可按下“Ctrl+i”或“Ctrl+u”模式。“Ctrl+u”是直接定位到右侧下面的对话框内输入任务命令；“Ctrl+i”则是在代码区弹出对话框输入任务命令，效果一致。

## 15.2 AI编写代码

本章将以第11章为例，应用AI进行零代码编写实现机器学习模型训练，并给出模型评估效果。

打开Trae，如图15-6所示，单击“选择项目”，选择相应的文件夹作为保存代码及其他文件的位置，如图中所示。并在位置输入以下提示词命令：

桌面上现有csv格式的数据文件Airlines.csv，根据所给数据，利用python进行以下分析：

1.找出最繁忙的机场；

2.找出JAX机场和DFW机场间的最短路径；

3.找出适合作为中转的机场（适合作为中转的机场不仅需要有较大的度，还要具有紧密中心性和介数中心性）；

4.给出完整的python代码。

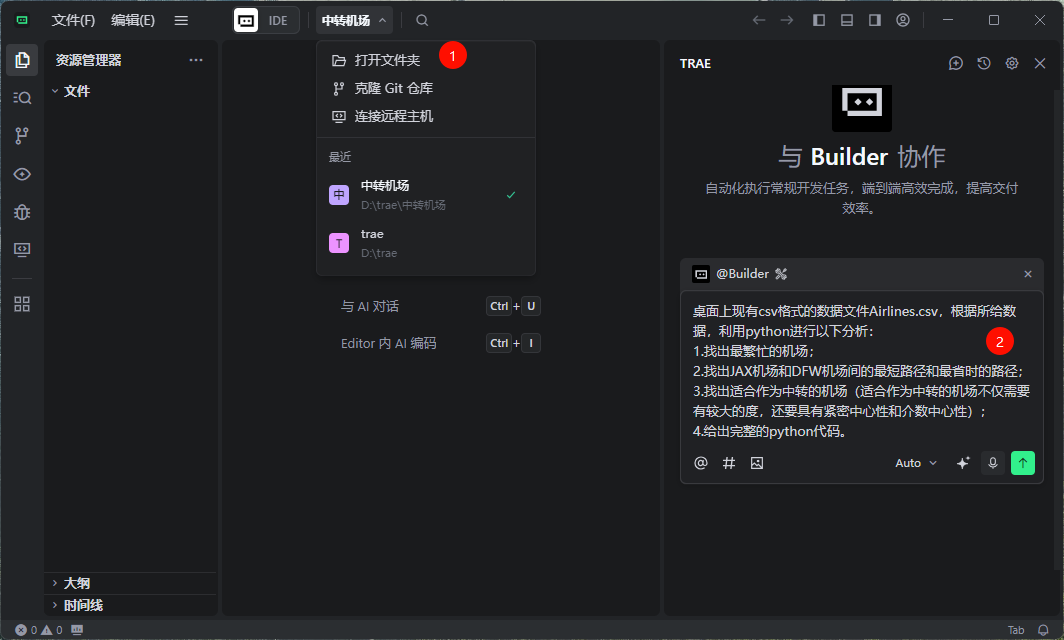


图15-6 设置文件夹并输入提示词命令

执行命令后，在界面的中央代码显示区域会自动输出代码，如图15-7所示。在执行命令和输出代码的过程中会出现图15-7中和处所示的“接受”/“全部接受”按钮字样，大部分情况下只需要点击“接受”/“全部接受”即可。若执行结束后出现图15-5中处所示处理错误，可按Ctrl+U处理。一般情况下会输出代码运行结果，如图15-8所示。

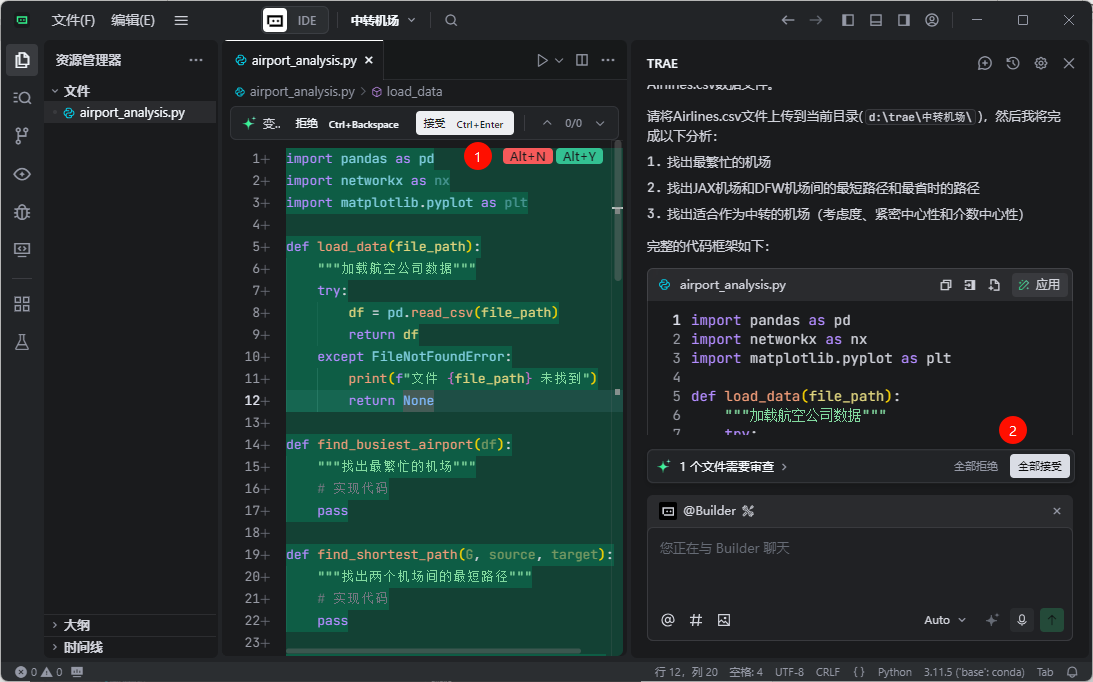


图15-7 执行命令输出代码并执行

如果没有给出结果，可点击图15-8中的处运行按钮（），即可执行代码，结果输出在中央区域下方处。

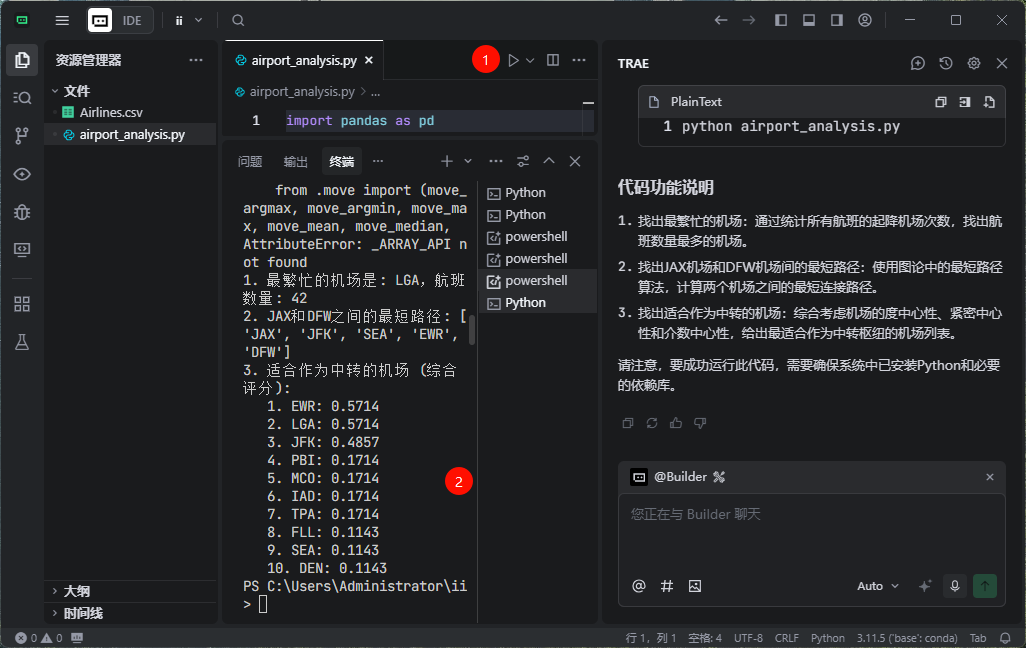


图15-8 执行命令输出代码并执行

输出结果如下：

1. 最繁忙的机场是: LGA，航班数量: 42

2. JAX和DFW之间的最短路径: ['JAX', 'JFK', 'SEA', 'EWR', 'DFW']

3. 适合作为中转的机场 (综合评分):

1. EWR: 0.5714

2. LGA: 0.5714

3. JFK: 0.4857

4. PBI: 0.1714

5. MCO: 0.1714

6. IAD: 0.1714

7. TPA: 0.1714

8. FLL: 0.1143

9. SEA: 0.1143

10. DEN: 0.1143

以上结论与第11章结论相似。

## 14.3 Trae生成完整代码

Trae生成的完整代码如下：

import pandas as pd

import networkx as nx

from collections import Counter

def load\_data(file\_path):

    """加载航班数据"""

    df = pd.read\_csv(file\_path)

    return df

def find\_busiest\_airport(df):

    """找出最繁忙的机场"""

    # 合并出发和到达机场

    all\_airports = list(df['origin']) + list(df['dest'])

    # 统计每个机场的航班数量

    airport\_count = Counter(all\_airports)

    # 找出最繁忙的机场

    busiest\_airport = airport\_count.most\_common(1)[0]

    return busiest\_airport

def find\_shortest\_path(df, origin, destination):

    """找出两个机场间的最短路径"""

    # 创建图

    G = nx.Graph()

    # 添加边

    for \_, row in df.iterrows():

        G.add\_edge(row['origin'], row['dest'])

    # 使用Dijkstra算法计算最短路径

    try:

        shortest\_path = nx.shortest\_path(G, origin, destination)

        return shortest\_path

    except nx.NetworkXNoPath:

        return f"无法找到从 {origin} 到 {destination} 的路径"

    except nx.NodeNotFound as e:

        return f"节点未找到: {e}"

def find\_hub\_airports(df, top\_n=10):

    """找出适合作为中转的机场"""

    # 创建有向图

    G = nx.DiGraph()

    # 添加边

    for \_, row in df.iterrows():

        G.add\_edge(row['origin'], row['dest'])

    # 计算度中心性

    degree\_centrality = nx.degree\_centrality(G)

    # 计算紧密中心性

    try:

        closeness\_centrality = nx.closeness\_centrality(G)

    except:

        closeness\_centrality = {}

    # 计算介数中心性

    betweenness\_centrality = nx.betweenness\_centrality(G)

    # 综合评分

    hub\_scores = {}

    for node in G.nodes():

        degree = degree\_centrality.get(node, 0)

        closeness = closeness\_centrality.get(node, 0)

        betweenness = betweenness\_centrality.get(node, 0)

        # 综合评分 = 度中心性 + 紧密中心性 + 介数中心性

        hub\_scores[node] = degree + closeness + betweenness

    # 排序并返回前top\_n个

    sorted\_hubs = sorted(hub\_scores.items(), key=lambda x: x[1], reverse=True)

    return sorted\_hubs[:top\_n]

def main():

    # 加载数据

    file\_path = 'Airlines.csv'

    df = load\_data(file\_path)

    # 1. 找出最繁忙的机场

    busiest\_airport = find\_busiest\_airport(df)

    print(f"1. 最繁忙的机场是: {busiest\_airport[0]}，航班数量: {busiest\_airport[1]}")

    # 2. 找出JAX机场和DFW机场间的最短路径

    shortest\_path = find\_shortest\_path(df, 'JAX', 'DFW')

    print(f"2. JAX和DFW之间的最短路径: {shortest\_path}")

    # 3. 找出适合作为中转的机场

    hub\_airports = find\_hub\_airports(df)

    print("3. 适合作为中转的机场 (综合评分):")

    for i, (airport, score) in enumerate(hub\_airports, 1):

        print(f"   {i}. {airport}: {score:.4f}")

if \_\_name\_\_ == "\_\_main\_\_":

    main()