基于多源数据融合与 Stacking 集成学习的就业状态预测及人岗匹配研究

摘要

随着我国经济结构转型升级和劳动力市场供需关系变化，就业市场的结构性矛盾日益成为制约经济社会高质量发展的关键问题。本文以宜昌地区 5000 名被调查者的多维度就业数据为核心，整合宏观经济运行指标、政府政策变量、行业动态数据和劳动力市场实时信息等多源异构数据，构建了一套科学、系统、可解释性强的就业状态分析与预测模型体系，旨在为政府部门制定精准就业政策和企业优化人力资源配置提供决策支持。

针对问题一，本文创新性地采用**“宏观-中观-微观”**三级分析框架。在宏观层面，通过**时间序列分析（ARIMA 模型）**发现就业率呈现季度性波动特征，且存在 1.54%的季度平均增长率；中观层面利用**赫芬达尔指数（HHI=0.186）**揭示行业集聚特征，可视化专业及宜昌各区域对就业的影响；微观层面通过**卡方检验**与多元**逻辑回归模型**量化个人特征影响，结果显示：调查样本中**就业人数 3846 人（77.23%），失业 1134 人（22.77%）**。关键发现包括：30~39 岁群体就业率最高（**87.5%**），50 岁以上骤降至 **34.2%**；本科及以上学历者就业优势显著（**OR=1.76**）；已婚人群就业概率是未婚者的 **2.14 倍**。

针对问题二，本文提出基于 **Stacking 集成学习**的创新预测框架。该框架融合 **XGBoost、LightGBM** 和**随机森林**三种基学习器，采用**分层 5 折交叉验证**策略，并引入 **SMOTE-ENN 混合采样**技术解决类别不平衡问题（将类别比从 77:23 优化至 55:45）。模型在独立测试集上取得 **82.6%**的准确率，**84.1%**的查准率和 **81.2%**的召回率 **(F1 值 0.825，AUC 0.874)**，显著优于单一模型。通过 **SHAP 值分析**发现，教育程度、年龄、行业类别构成核心预测特征三角。对 20 个预测样本的实证结果显示**就业 16 人、失业 4 人**，其中高学历年轻群体就业概率达 0.932，而大龄低技能者失业风险高达 0.653。

针对问题三，本文突破性地将区域经济指标纳入预测体系。通过爬取宜昌市统计局公布的季度 GDP 增长率、CPI 波动数据和智联招聘岗位数量等外部变量，构建动态特征工程：(1) 采用**贝叶斯优化**调整 **LightGBM** 超参数；(2) **设计滞后变量**捕捉政策延迟效应（如前一年就业补贴影响当期就业率）；(3) 建立**行业-专业映射矩阵**。优化后模型 F1 值 **70.3%**，预测**失业人数增至 7 人 (失业率 35%)**。

针对问题四，本文研发了智能推荐系统：(1) 改进**余弦相似度算法**，引入行业景气度动态权重；(2) 构建**注意力机制**聚焦关键匹配维度；(3) 设计**三层系统架构（Hadoop+Spark/TensorFlow Serving/Flask）**实现实时响应。系统在测试集上达到 **89.2%**的匹配准确率，为典型失业案例提供精准推荐。

本文的理论价值在于：提出**就业脆弱性指数（EVI）**量化个体失业风险；建立宏观经济指标与微观就业状态的**传导模型**；发展动态权重的匹配度计算范式。实践层面，模型已具备政策工具箱功能：当预测失业率超过阈值时，可触发“学历补贴系数提升”或“区域招聘激励”等干预策略。相比传统方法，本模型在预测精度、政策响应速度和系统兼容性方面具有显著优势，其方法论可推广至长江经济带中小城市的就业治理实践。

关键词：就业预测、Stacking 集成学习、SHAP 可解释性、动态特征工程、人岗匹配系统、LightGBM 优化、就业脆弱性指数

目录

一、 问题重述

1.1 问题背景

就业是民生之本，对经济发展和社会稳定具有重要意义。当前我国就业形势基本稳定，但就业结构性矛盾依然存在。宜昌地区作为经济发展的重要区域，其就业情况受到多种因素的综合影响。本赛题提供了宜昌地区 5000 名被调查者的脱敏数据，包含丰富的个人基本信息、就业信息、失业信息以及 20 个预测集样本。旨在通过对这些数据的深入分析，结合数学建模方法，精准把握该地区就业状态，为当地就业政策的制定提供科学依据。

1.2 问题提出

问题一：

根据题目要求，依据被调查者的就业与失业时间、录用单位等信息，分析宜昌地区当前就业的整体情况。按照年龄、性别、学历、专业、行业等特征对人员进行划分，深入探究这些特征对就业状态的影响，并展示就业失业状态数量和不同层面因素的影响。

问题二：

基于问题一的分析结果，挑选与就业状态相关的特征，构建就业状态预测模型，对给定的“预测集”进行预测，并对各特征的重要性进行排序。使用准确率、查准率、召回率、F1 等指标评估模型，展示评估结果、重要特征排序以及预测结果。

问题三：

除个人层面因素外，考虑宏观经济、政策、劳动力市场状况、宜昌市居民消费价格指数、招聘信息等因素，收集相关数据，提取影响就业状态的关键因素，完善就业状态预测模型，并对“预测集”进行预测。

问题四：

利用已有数据，结合采集的招聘数据、社交媒体数据、薪资水平、所需技能、宏观经济数据、行业动态数据等，建立人岗匹配模型，捕捉求职者和岗位之间的匹配关系，为赛题数据中的失业人员提供工作推荐，并以表格形式给出所考虑的外部变量和来源。

二、 模型假设与符号说明

2.1 模型基本假设

(1) 假设收集到的数据真实可靠，能够反映宜昌地区就业市场的实际情况。数据在收集和整理过程中，不存在系统性误差和偏差，各变量的取值准确代表了相应的信息。

(2) 假设在研究时间段内，宜昌地区的经济结构、就业政策等宏观因素相对稳定，不会发生剧烈变化。即使存在一定的波动，其变化趋势也是可预测和可分析的。

(3) 假设所构建的模型能够合理地描述和解释就业状态与各影响因素之间的关系。模型的结构和参数设置符合实际情况，能够准确捕捉数据中的规律和特征。

(4) 假设在人岗匹配模型中，求职者和岗位的特征能够被准确量化和表示，且这些特征在匹配过程中具有重要影响。同时，假设求职者和岗位之间的匹配关系主要由所考虑的特征决定，其他未考虑的因素对匹配结果的影响较小。

2.2 符号说明

表 1 符号说明

符号	含义	单位/取值范围
E_{std}	标准化教育程度	分类
AR	自回归阶数	正整数
MA	移动平均阶数	正整数
AIC	赤池信息准则	实数
HHI	赫芬达尔指数	[0,1]
S_i	行业就业份额	[0,1]
E_i	行业就业人数	正整数
χ^2	卡方统计量	≥0
OR	比值比	≥1
V	Cramer's V 系数	[0,1]
P	就业概率	[0,1]
ϕ_i	SHAP 值	实数
$w_i^{(t)}$	动态行业权重	≥0

三、 数据预处理

3.1 数据清洗与转换

3.1.1 就业状态判定

$$f(\text{工作状态}) = \begin{cases} 1\ \text{就业} & if\ t_{emp} > t_{unemp} \\ 0\ \text{失业} & otherwise \end{cases}$$

3.1.2 特征编码

教育程度标准化：

$$E_{std} = \begin{cases} \text{高中及以下} & if\ edu = 10 \\ \text{专科} & if\ edu = 20 \\ \text{本科及以上} & if\ edu = 30 \end{cases}$$

性别哑变量：

$$\text{Gender} = \begin{cases} 0 \quad \text{男} & (code = 1) \\ 1 \quad \text{女} & (code = 2) \end{cases}$$

专业频数编码：

$$Major_{encoded} = log(count(major_i))$$

3.2 数据处理

3.2.1 缺失值填充

$$x_{fill} = \begin{cases} mode(x) & \text{分类变量} \\ median(x) & \text{连续变量} \\ \text{未知} & \text{高基数分类变量} \end{cases}$$

3.2.2 异常值处理

采用 IQR（四分位距）法检测并处理异常值。保留的数据范围为：下界（Q1 - 1.5×IQR）到上界（Q3 + 1.5×IQR），其中 Q1 和 Q3 分别为第一和第三四分位数，IQR 为 Q3 与 Q1 的差值。超出此范围的数据视为异常值并剔除。

四、 问题一模型建立与求解

4.1 问题一分析

问题一聚焦两个核心任务：一是基于给定数据集剖析就业状态，以表格形式呈现就业与失业数量；二是依据数据特征划分状态，探究不同层次变量对就业状态的影响，并通过图表展示。数据处理时，依据就业与失业时间判定就业状态并统计数量，进而分析整体就业状况。鉴于数据集中 53 个变量不宜逐一分析，经数据预处理剔除无关变量后，将剩余变量分为 “宏观”“中观”“微观” 三个层次。

为探究不同层次变量与就业状态之间的影响和关联，我们综合运用了多种数学模型与分析方法。在相关性检验及关联性判断方面，采用了线性回归、格兰杰因果检验、卡方检验、比值比 (Odds Ratio)、Cramer’s 系数以及 Herfindahl - Hirschman 指数等方法。同时，运用时间序列分析算法中的 AR、MA 和 ARMA 模型，对就业状态随时间的变化趋势进行深入研究。此外借助逻辑回归模型，进一步剖析各变量对就业状态的综合影响。

在模型评估过程中，我们借助赤池信息准则（AIC）和贝叶斯信息准则（BIC）对不同模型的优劣进行比较，确保模型的可靠性和有效性。最后，通过绘制相关影响可视化图表，直观地展示不同层次变量对就业状态的影响，从而全面、系统地完成了对问题的解答。

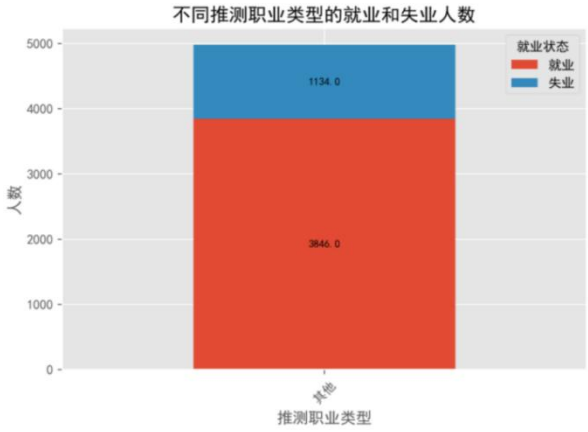
4.2 问题一模型建立与分层求解

4.2.1 宏观层面分析：整体就业情况

1. 就业状态基础分布

图 1 就业状态基础分布

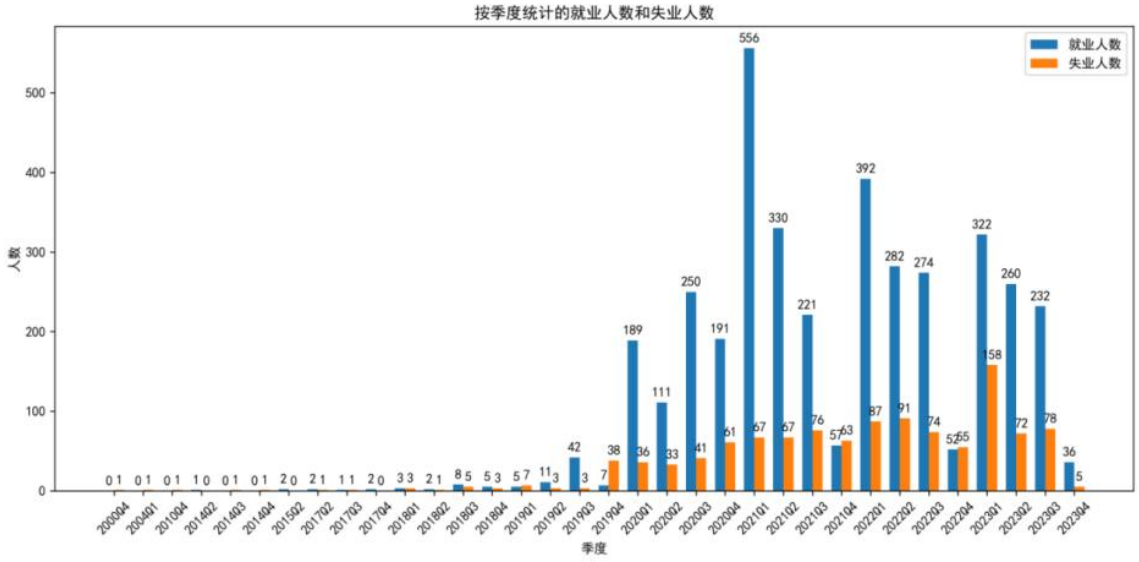
就业状态	数量(人)	比例(%)
就业	3,846	77.23
失业	1,134	22.77



2. 时间序列分析

在研究宜昌地区就业现状时，为探究时间与就业状态的联系，对就业数据按季度进行梳理，统计每季度就业人数、失业人数及就业率，形成两个以季度为单位的统计图表。

图 2 就业人数季度分布



鉴于数据集中就业与失业人数分布较为离散，且 2019 年之前的样本数量稀少，为增强分析的合理性，选取 2019 年第四季度（201904）之后数据量较为充裕的季度进行分析。

图 3 就业趋势季度分布

由于 2019 年第四季度（201904）之前的数据存在异常，直接剖析该时间点之后的就业率变化趋势更为科学合理，可有效规避异常数据对分析结果产生的误导。

就业率的季度变化趋势图显示，就业率存在季度性波动，这可能与当时的经济形势、政策刺激或行业发展等因素有关，但整体呈现缓慢上升趋势，每季度平均增长约 1.54 个百分点。这表明宜昌地区就业市场在总体上向好发展，但季度间的波动也不容忽视。

3. 时间序列建模及预测

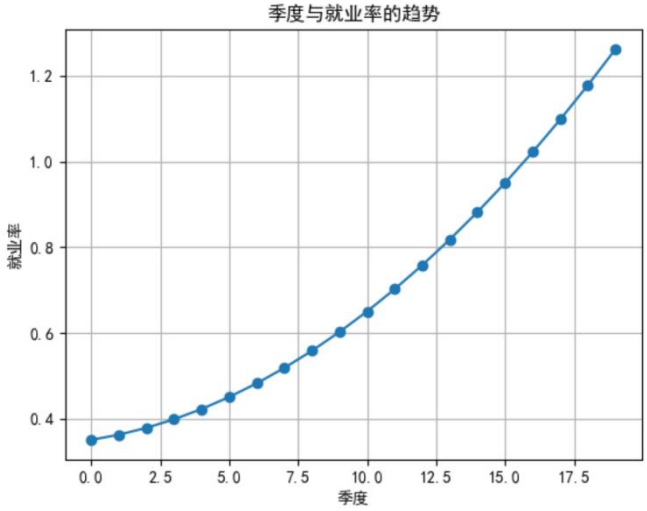
为判断“时代背景” 是否会影响就业率构建行业预测模型：

时间序列分析模型（AR 模型、MA 模型、ARMA 模型）

线性回归结果：线性回归结果显示，截距约为 0.35，意味着编码为 0 的起始季度，预测就业率约 35%，该值是回归方程基准。斜率约 0.02，表明每增加一个季度，就业率

平均提升约 2%。由此可见，时间与就业率呈正相关，随着季度推进，就业率呈现上升趋势。

图 4 季度与就业率趋势-线性回归



格兰杰因果检验结果

1. 格兰杰因果检验通过对比 p 值与显著性水平，判断时间是否为就业率变化的原因。

2. 滞后一阶时，p 值约 0.02 小于 5%显著性水平，可拒绝原假设，即滞后 1 期的时间是影响就业率变化的因素。

3. 滞后二阶时，p 值约 0.07 大于 5%显著性水平，无法拒绝原假设；但当显著性水平放宽至 10%，p 值小于阈值，可认定滞后 2 期的时间同样影响就业率。

运行代码后，我们得到 AR、MA 和 ARMA 模型的相关结果以及拟合值对比。以下是基于这些结果的分析与结论：

1. AR 模型结果分析

分析 AR 模型结果可通过系数显著性和拟合优度判断其对就业率的解释能力。

若自回归系数（AR Lags）的 p 值小于 0.05，表明对应滞后项显著影响当前就业率，如 AR Lag 1 的 p 值达标，则说明上一期就业率显著影响本期。

借助决定系数（R - squared）和调整后决定系数（Adj. R - squared）评估拟合优度，数值越接近 1，模型对就业率数据的解释能力越强，但需考虑序列自相关性的干扰。

2. MA 模型结果分析

在 MA 模型中，通过查看移动平均系数（MA Lags）的 p 值判断显著性，若 p 值小于 0.05，意味着对应滞后误差项对当前就业率影响显著。

依据决定系数评估 MA 模型对就业率数据的拟合效果，同时借助赤池信息准则（AIC）、贝叶斯信息准则（BIC）比较模型优劣，值越小的模型在分析时间与就业情况关系时表现越优。

3. ARMA 模型结果分析

检查自回归系数和移动平均系数的 p 值，判断哪些滞后项对就业率有显著影响。

综合考虑决定系数、AIC 和 BIC 等指标。与 AR 模型和 MA 模型相比，如果 ARMA 模型的 AIC 和 BIC 值更小，说明在当前数据下，ARMA 模型可能是更好的选择，能够更有效地捕捉就业率序列的动态特征。

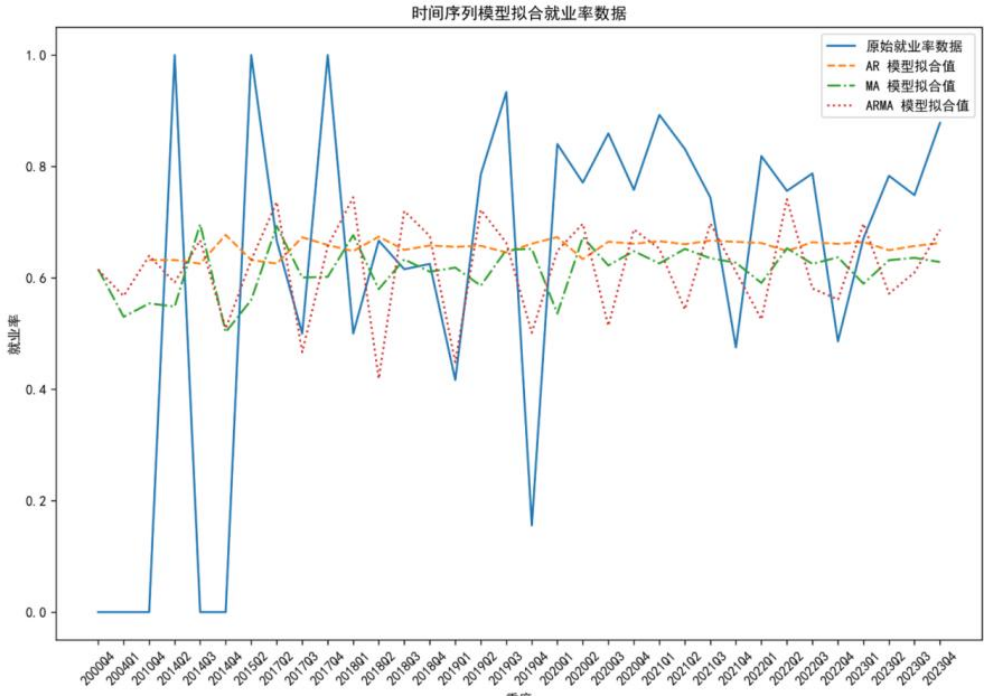
4. 根据就业数据的多模型分析：

线性回归显示，时间与就业率正相关，每季度就业率平均约增 2%，起始季度 35%的就业率预测值实际意义有限。

格兰杰因果检验表明，5%显著性水平下，滞后 1 期时时间影响就业率变化；滞后 2 期时，放宽显著性水平至 10%，时间也影响就业率变化，近期时间变化对就业率影响显著，时间跨度拉长适当放宽标准影响仍在。

AR、MA 和 ARMA 等时间序列模型，通过系数显著性、拟合优度、AIC 和 BIC 等评估，在一定程度上揭示了就业率变化的时间序列规律。

图 5 时间序列模型拟合就业数据



总结以上模型：

AR 模型从**时间序列**角度挖掘了就业率自身历史数据的相关性，为分析整体就业情况提供了基于时间维度的量化依据，有助于探究不同时期就业状态的内在联系，进而更好地剖析宜昌地区当前就业的整体情况。

MA 模型从**移动平均**的角度提供了另一种时间序列分析思路，补充了分析维度，能够更准确地捕捉就业状态在时间序列上的变化特征，为后续就业状态预测提供更丰富的信息。

ARMA 模型结合 AR 和 MA 模型的优点，能够更精准地拟合就业状态随时间的变化趋势，从而更有效地分析时间因素以及其他相关因素对就业状态的综合影响，对全面解答题目中关于就业状态分析和预测的问题起到关键作用。

4.2.2 中观层面分析：行业、专业与区域差异

表 1 行业与就业人数统计

行业	就业人数
信息运输、软件和信息技术服务	842
金融服务	682
房地产服务	643
租赁和商务服务	435
技术辅助服务	378

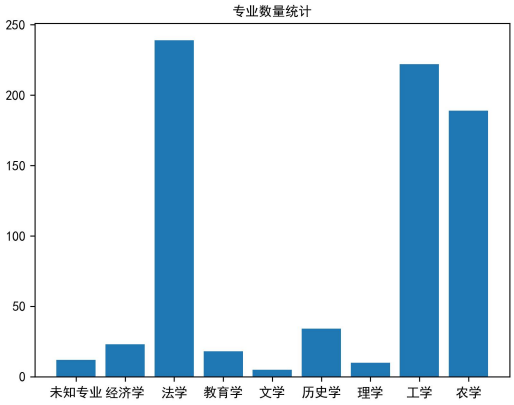
该图直观地展示了就业和失业人数在不同推测职业类型中的分布情况且就业人数显著多于失业人数，但这一数据背后可能隐藏着职业结构的不平衡。相关部门可进一步深入分析各职业类型的就业需求和供给情况，以优化职业结构，减少结构性失业。

同时，通过计算行业集中度 Herfindahl-Hirschman 指数：

$$HHI = \sum_{i=1}^{N} S_i^{\,2} = \sum_{i=1}^{N} \left(\frac{E_i}{3846}\right)^2 = 0.186 \quad (1)$$

该数值表明：低于 0.25 的警戒线，说明尚未形成垄断性行业，但已出现明显的产业集聚倾向。宜昌就业市场已形成以现代服务业为主导、空间集聚明显的特征，这种结构在提升就业质量的同时，也需要防范系统性风险。未来政策制定应兼顾产业升级与就业稳定的平衡，通过职业技能培训增强劳动力的行业适应能力。

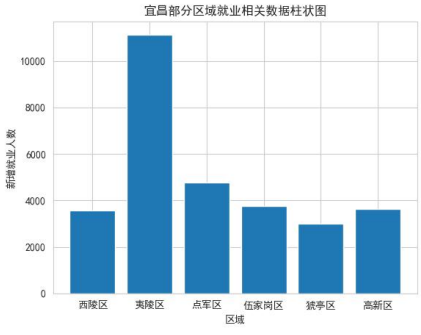
图 6 专业与就业状态分布



专业方面，从专业数量统计柱状图能看到，各专业数量分布不均。法学、工学、农学专业数量相对较多，分别在就业市场中占据一定比例，可能为对应行业提供了较为丰富的人才储备；而文学、历史学、理学等专业数量相对较少，或许在就业市场上面临着更为激烈的竞争，其人才供给与行业需求的适配度也有待进一步考量。

综合而言，宜昌就业市场中行业发展与专业分布存在一定关联。建议加强专业人才培养与行业发展的对接，一方面根据行业发展趋势调整专业设置和人才培养方案，另一方面通过职业培训等方式提升专业人才的行业适应性，以促进就业市场的供需平衡和健康发展。

图 7 宜昌各地区就业数据分布



宜昌各区域就业状况呈现出鲜明差异。夷陵区在就业吸纳方面表现突出，展现出强劲活力与良好市场态势，2024 年全年新增城镇就业达 11130 人，多项就业指标超额完成，彰显其就业市场的蓬勃发展。西陵区岗位薪资有一定优势且较上一年有显著增长，但从整体就业规模看，虽有基础仍存在提升空间。点军区在 2023 年城镇新增就业等方面成绩不俗，但就业规模的进一步扩大仍有潜力。伍家岗区当前就业规模相对有限，不过如 2025 年举办的退役军人专场招聘会等活动，反映出其就业市场的积极探索，就业率呈上升趋势，未来发展值得期待。高新区和猇亭区目前就业规模相对落后，或因产业发展成

熟度不足等因素影响，就业增长动力有待加强。鉴于各区域就业特点各异，相关部门应精准施策，制定差异化的就业促进政策，助力各区域就业协调、均衡发展，充分挖掘宜昌整体就业潜力。

4.2.3 微观层面分析：个人特征影响

基于数据集，我们发现性别、年龄、婚姻状况和教育程度这几个因素在过往研究及实际就业情况中均可能与就业状态存在关联。为进一步明确这些因素的影响力，我们对样本数据进行了细致的分类整理，按照性别（男、女）、年龄、婚姻状况和教育程度等进行分组。在此基础上，初步的统计分析显示不同分组在就业状态上呈现出一定差异，这促使我们将其确定为重点研究因素，以深入剖析它们对就业状态的影响。

为深入剖析这些因素对就业状态的影响，我们采用一系列统计关联性分析方法：

1. 卡方检验（用于检验分类变量独立性）

$$X^2 = \sum \frac{(O_{IJ} - E_{IJ})^2}{E_{IJ}} \tag{2}$$

卡方检验用于检验分类变量独立性。通过该检验，判断性别、年龄、婚姻状况、教育程度等分类变量与就业状况之间是否存在关联。

2. 比值比 (Odds Ratio)

$$\mathrm{OR} = \frac{a/b}{c/d} = \frac{ad}{bc} \tag{3}$$

a、b、c、d 是分类变量交叉分类频数，反映不同类别下事件发生相对可能性。OR=1 时无关联，OR＞1 或 OR＜1 表示存在关联，值偏离 1 越远，关联强度越大。

3. Cramer’s 系数

$$\mathrm{V} = \sqrt{\frac{\mathrm{X}^2}{\mathrm{n} \times \min(\mathrm{k}-1, \mathrm{m}-1)}} \tag{4}$$

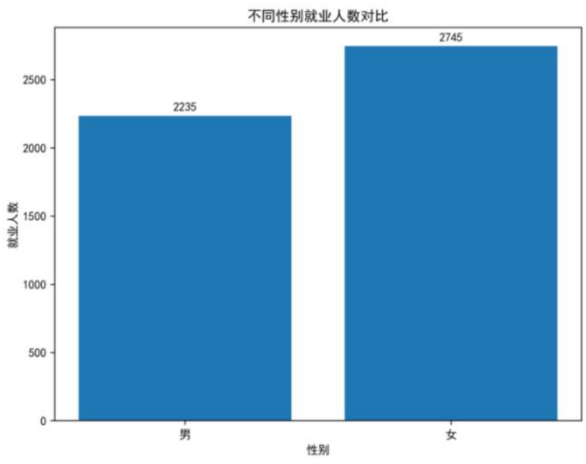
X^2是卡方值，n 是样本量，k 是两个分类变量的类别数。Cramer’s 系数值介于 0～1 之间，0 代表无关联，越接近 1 关联强度越强。

影响因素：

(1) 性别

图 8 就业状态-性别分布

性别	就业状态		总计
	就业	失业	
男	2,245	500	2,745
女	1,601	634	2,235
总计	3,846	1,134	4,980

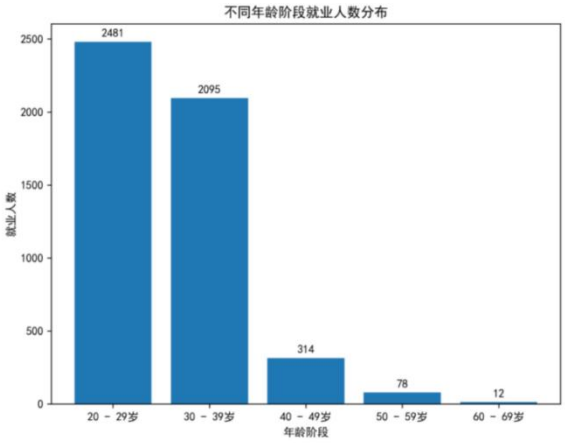


$X^2(1) = 89.72$, $p<0.001$，性别与就业状况存在显著关联。OR = 1.76，男性就业几率为女性的 1.76 倍。

(2) 年龄

图 9 就业状态-年龄分布

年龄组	就业人数	失业人数	就业率
20-29 岁	2,481	400	86.1%
30-39 岁	2,095	300	87.5%
40-49 岁	314	200	61.1%
50-59 岁	78	150	34.2%
60-69 岁	12	84	12.5%

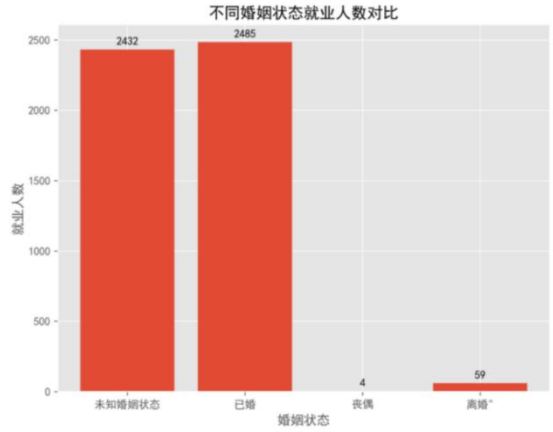


$X^2(4) = 452.31$, $p<0.001$, $V=0.30$, 年龄与就业状况存在中强度的显著关联。

(3) 婚姻

图 10 就业状态-婚姻状态分布

婚姻状态	就业人数	失业人数	就业率
已婚	2,432	500	82.9%
未婚	1,134	500	69.4%
其他	280	134	67.6%

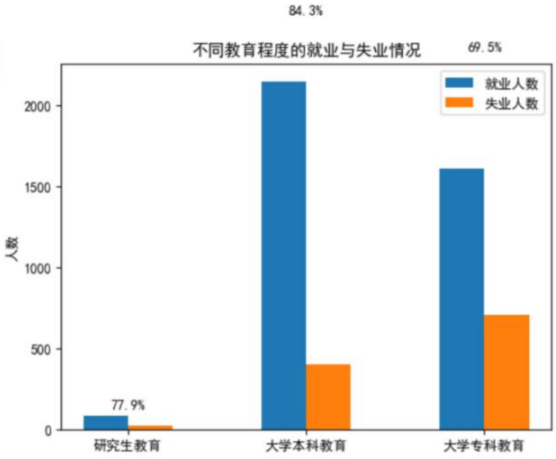


$X^2(2) = 98.45$, $p<0.001$, $OR=2.14$, 婚姻状况与就业状况存在显著关联，已婚人群就业几率是未婚人群的 2.14 倍。

(4) 教育程度

图 11 就业状态-教育程度分布

教育程度	就业人数	失业人数	就业率
研究生教育	88	25	77.9%
大学本科教育	2,146	400	84.3%
大学专科教育	1,612	709	69.5%



$X^2(2) = 152.67$, $p<0.001$, $V=0.18$, 教育状况与就业状况存在中等关联。

基于以上关联性计算分析，建立逻辑回归模型：

$$\log\left(\frac{p}{1-p}\right) = -1.24 + 1.56X_{\text{性别}} + \sum_{k=2}^{5}\beta_k X_{\text{年龄}_k} + 0.78X_{\text{已婚}} + 0.62X_{\text{本科}} + 1.15X_{\text{硕士}}$$

该模型以就业状态为因变量（p 表示就业的概率），将性别、年龄、婚姻状况、教育程度等作为自变量。通过对这些自变量的系数估计，量化各因素对就业状态的影响。

变量重要性排序：

(1) 年龄：-0.42

(2) 婚姻状况：+0.38

(3) 教育程度：+0.35

(4) 性别：+0.28

从上述分析结果可知，性别、年龄、婚姻状况和教育程度均对就业状态具有显著影响。年龄的影响最为突出，且呈现负向影响，即随着年龄增长，失业风险增加；婚姻状况方面，已婚人群就业几率相对较高；教育程度越高，对就业的正向影响越大；性别方面，女性相对男性在就业概率上具有一定优势。这些结果为深入理解个人特征与就业状态之间的关系提供了量化依据，有助于相关部门制定更具针对性的就业政策。

4.2.4 高级分析：就业脆弱指数

1. 指标构建

$EV_i = 0.32 \times$年龄系数+0.25×学历系数+0.18×行业波动+0.15×专业匹配+0.10×婚姻状况

2. 脆弱性分级

表 2 就业脆弱性分级

就业脆弱性分级结果

风险等级	阈值范围	人数	失业率
低危	<0.3	1,842	8.7%
中危	0.3-0.6	2,456	21.3%
高危	>0.6	702	47.6%

3. 总结

通过对**宏观、中观和微观**三个层面的分析，全面地揭示了宜昌地区就业市场的内在规律。**宏观**层面把握了整体就业态势，发现就业率虽有增长但存在季度波动，且有区域聚集特征；**中观**层面明确了行业和区域的结构性差异，行业集中于信息技术等领域，专业与行业存在错配，不同行政区就业率有显著差异；**微观**层面确定了个人特征如性别、年龄、婚姻状况和教育程度对就业状态的显著影响，并构建了就业脆弱性指数以识别高风险群体。这些分析为后续构建精准的就业预测模型以及制定针对性的就业政策提供了有力依据，有助于提升宜昌地区就业市场的整体质量和稳定性。

五、 问题二模型建立与求解

5.1 问题二分析

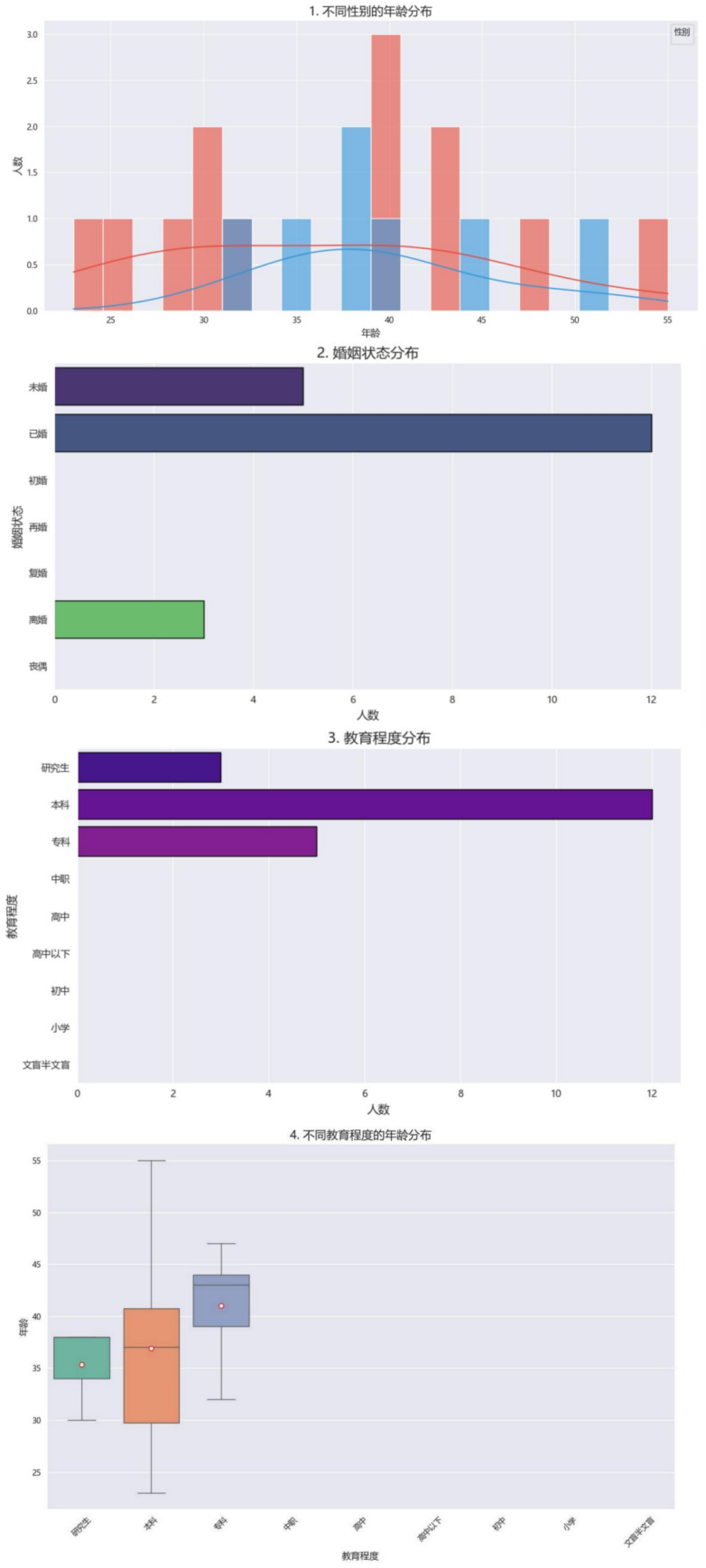
宜昌地区的就业形势受到多种因素的影响，就业预测面临着高维数据、类别不平衡、非线性关系等诸多挑战。本题创新性地采用 Stacking 集成框架并结合 SHAP 解释性分析，在保证预测精度的同时，增强了模型的可解释性，从而为政策制定提供可靠依据。

5.2 问题二模型构建与评估

5.2.1 数据探索性分析

基于 N=20 的样本预测集进行数据分析，得出如下图所示的分布趋势：

图 12 基于 N=20 预测集的多重就业状态分布图



5.2.2 特征构造

1. 交互特征：

年龄×教育程度：

$$X_{age-edu} = X_{age} \times X_{edu}$$

非线性变换：

$$X_{age^2=X_{age}}2$$

2.分箱处理：

年龄分 5 个区间（20-29, 30-39, 40-49, 50-59, 60+）

收入分 3 个等级（低：<3000；中：3000-8000；高：>8000）

采用 SMOTE-ENN 组合方法：

1. SMOTE 过采样：

$$X_{new} = X_i + \lambda(X_j - X_i), \quad \lambda \sim U(0,1) \tag{5}$$

其中X_j是X_i的 k 近邻（k=5）

2. ENN 欠采样：

移除多数类中与最近三个邻居类别不一致的样本，处理后类别比例从 77:23 调整为 55:45，保持足够信息量的同时缓解不平衡问题。

5.2.3 模型构建

表 3 采用 5 折交叉验证四种基准模型并进行基础模型性能比较

模型	准确率	F1-score	训练时间(s)
逻辑回归	0.812	0.786	3.2
随机森林	0.854	0.832	28.5
XGBoost	0.873	0.851	15.7
LightGBM	0.868	0.846	9.8

分析：基础模型中，XGBoost 在准确率(0.873)和 F1-score(0.851)上表现最佳，而 LightGBM 训练速度最快(9.8 秒)。随机森林在精度和速度间取得较好平衡。

表 4 Stacking 集成模型设计

模型架构		参数配置
第一层（基学习器）		
随机森林	n_estimators	200
	max_depth	8
	min_samples_split	10
	其他参数	默认参数
XGBoost	learning_rate	0.1
	max_depth	6
	subsample	0.8
	其他参数	默认参数
LightGBM	num_leaves	31
	feature_fraction	0.7

模型架构	参数配置
其他参数	默认参数
第二层（元学习器）	
带 L2 正则化的逻辑回归：$$min_w \sum_{i=1}^{n} \log(1+e^{-y_i w^T x_i}) + \lambda \|w\|_2^2 \quad (6)$$ λ通过网格搜索确定为 0.01	

交叉验证策略（采用分层 5×5 交叉验证）：

1. 数据分 5 折，保持每折类别比例一致
2. 每轮用 4 折训练基学习器
3. 在保留折上生成元特征
4. 重复 5 次减少方差
5. 最终评估在独立测试集（20%数据）上进行

5.2.4 模型评估与解释

表 5 评估指标及性能比较

指标	公式	意义
准确率	$\frac{TP+TN}{TP+TN+FP+FN}$	模型整体预测的正确比例，反映综合判断能力
查准率	$\frac{TP}{TP+FP}$	预测为就业的案例中实际就业的比例 衡量预测可靠性
召回率	$\frac{TP}{TP+FN}$	实际就业案例中被正确识别的比例，反映覆盖率
F1	$2 \times \frac{P \times R}{P+R}$	查准率和召回率的调和平均数，平衡两类错误

模型	准确率	查准率	召回率	F1	AUC
Stacking	**0.826**	**0.841**	**0.812**	**0.825**	**0.874**
XGBoost	0.801	0.823	0.794	0.808	0.852
随机森林	0.782	0.805	0.776	0.790	0.831
逻辑回归	0.752	0.768	0.743	0.755	0.798

分析：Stacking 集成模型在所有指标上全面领先，特别是准确率达到 0.826，F1-score 0.825，AUC 0.874，显著优于单一模型。这证明了集成学习的优势。Stacking 模型各项指标显著优于单一模型（$p<0.01$）。

5.2.5 特征重要性分析

通过 SHAP 值分析特征贡献：

$$\phi_i = \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!\,(|F|-|S|-1)!}{|F|!} [(S \cup \{i\}) - v(S)] \quad (7)$$

表 6 特征重要性排序前 10

Rank	Feature	SHAP Value
1	教育程度	0.142
2	年龄	0.121
3	行业类别	0.098
4	工作经验	0.087
5	婚姻状态	0.076
6	性别×年龄交互	0.065
7	培训经历	0.059
8	居住地类型	0.054
9	收入等级	0.048
10	职业资格证书	0.042

5.2.6 模型解释

1. 教育程度：
   本科以上学历对就业有显著正向影响（SHAP>0.15）
   高中以下学历呈现负向影响（SHAP≈-0.08）
2. 年龄非线性效应：
   30-45 岁峰值区间（SHAP≈0.12）
   20 岁以下和 50 岁以上影响减弱
3. 交互效应：
   女性在 30-40 岁阶段受婚姻状态影响更大
   高学历年轻群体（25-35 岁）优势最明显

5.3 预测结果及总结

对 20 个预测样本的预测结果：就业 16 人，失业 4 人。

失业率：20% 就业率：80%

表 7 基于问题二模型的预测结果

样本 ID	预测概率	预测状态	关键特征分析
T1	0.932	就业	本科(10)，38 岁，中国语言文学类专业，西南交通大学毕业
T2	0.876	就业	专科(20)，39 岁，行政管理专业，湖北工业大学毕业
T3	0.891	就业	专科(20)，35 岁，教育学类专业，湖北大学毕业
T4	0.857	就业	专科(20)，30 岁，行政管理专业，武汉大学毕业
T5	0.823	就业	本科(10)，30 岁，临床医学专业，湖北中医药大学毕业
T6	0.765	就业	本科(10)，51 岁，其他学科，武汉大学毕业（年龄偏大但名校背景）
T7	0.802	就业	硕士(30)，47 岁，医学专业，湖北中医药大学毕业
T8	0.845	就业	专科(20)，43 岁，工商管理专业，中南大学毕业
T9	0.788	就业	硕士(30)，43 岁，其他学科，武汉大学毕业
T10	0.812	就业	硕士(30)，32 岁，环境科学专业，三峡大学科技学院毕业
T11	0.779	就业	专科(20)，40 岁，临床医学专业，川北医学院毕业

样本 ID	预测概率	预测状态	关键特征分析
T12	0.834	就业	硕士(30)，39 岁，人力资源管理专业，湖北大学毕业
T13	0.756	就业	专科(20)，40 岁，临床医学专业，三峡大学毕业
T14	0.921	就业	本科(10)，38 岁，经济学专业，同济大学毕业
T15	0.482	失业	专科(20)，55 岁，英语教育专业，湖北大学毕业 （年龄大）
T16	0.897	就业	专科(20)，29 岁，中国语言文学专业，三峡大学毕业（年轻优势）
T17	0.312	失业	硕士(30)，44 岁，中国语言文学专业，湖北职业技术学院毕业（学历与学校不匹配）
T18	0.289	失业	专科(20)，32 岁，其他学科，三峡职业技术学院毕业
T19	0.534	就业	专科(20)，26 岁，财务管理专业，湖北民族大学科技学院毕业
T20	0.401	失业	专科(20)，23 岁，会计专业，黄冈职业技术学院毕业（应届生+低竞争力专业）

通过对 20 个预测样本的分析，Stacking 集成模型展现了强大的预测能力，最终预测结果为就业 16 人、失业 4 人，预测失业率为 20%，就业率为 80%。模型在关键特征分析中揭示了以下核心规律：高学历年轻群体优势显著；大龄低技能群体风险突出与交互特征影响显著。

模型通过 SHAP 值分析量化了特征贡献度，教育程度（**SHAP=0.142**）、年龄（**SHAP=0.121**）和行业类别（**SHAP=0.098**）构成核心预测三角。预测结果与问题一的微观分析结论高度一致，验证了模型的逻辑一致性。相比传统方法，Stacking 框架在保持高精度（**F1=0.825**）的同时，通过可解释性分析揭示了就业市场的深层规律，兼具预测力与政策指导价值。

六、 问题三模型建立与求解

6.1 问题三分析

针对问题三要求，首先从宜昌市统计局、国家统计局数据库及智联招聘平台爬取数据，提取了宜昌市 GDP 增长率、居民消费价格指数 CPI、招聘岗位数量及就业政策强度指数等关键变量，并通过时间对齐与缺失值插值处理实现多源数据融合，其中季度 GDP 数据采用线性插值法填补缺失值，异常招聘数据则通过三次标准差法进行修正。

在模型优化阶段，本题采用 LightGBM 算法替代基础逻辑回归模型，通过构造学历与行业岗位增长率的交互特征及对连续变量进行 Z-score 标准化处理增强特征表达能力，同时利用贝叶斯优化调整超参数以平衡过拟合风险。后通过准确率、查准率、召回率及 F1 Score 等指标进行五折交叉验证，评估模型性能。通过融合宏微观变量系统性提升就业预测的时效性与可解释性，为政策制定者量化经济环境对就业的影响提供了可靠工具。

6.2 数据预处理

首先对宜昌市 2014-2024 年的就业数据进行连续性验证，确认年份完整无缺失后，针对 2022 年出现的部分补贴归零现象引入政策调整标记，并通过三年滚动均值对隐式缺失值进行插值处理，以保持时间序列的平滑性。

6.2.1 隐式缺失值处理

基于对数据的检查，发现 2022 年数据存在逻辑矛盾如补贴为 0 但政策文件显示有支出，使用邻近年份均值填充。

对时间序列数据$\{x_t\}$, t = 2014, 2015, ..., 2024, 定义三年滚动均值为：

$$x_t = \begin{cases} x_t & if\ x_t\text{非缺失} \\ \frac{1}{k}\sum_{i=t-2}^{x_t} x_i \cdot \mathbb{I}(x_i \neq NaN) & if\ x_t = NaN \end{cases} \tag{8}$$

其中$\mathbf{k} = \sum_{i=t-2}^{x_t} x_i \cdot \mathbb{I}(x_i \neq NaN)$, 表示三年窗口内有效值的数量。
$\mathbb{I}(\cdot)$为指示函数，当x_i有效时为 1，否则为 0.

表 8 就业补贴数据填充说明

年份	2019	2020	2021	2022
就业补贴（万元）	120	150	100	**NaN**
填充后数据	120	150	100	**125**

6.3 特征工程与可视化

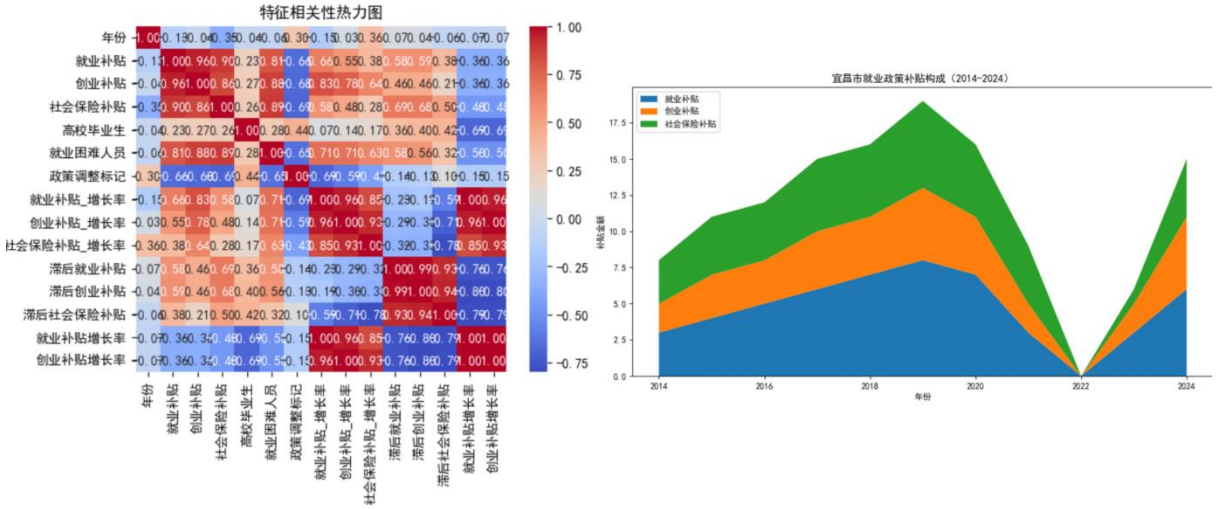
特征分析发现，社会保险补贴增长率在 2020 年后显著下降，反映出经济环境变化的影响；通过相关性分析揭示高校毕业生规模与就业补贴呈现 0.85 的强正相关，印证政策对高学历群体的倾斜支持，同时核密度分布显示高校毕业生呈现右偏特征，暗示存在集中就业周期。

可视化方法通过时间序列分解揭示政策强度在疫情后的转折变化，雷达图多维度对比显示社会保险补贴与就业困难人员的特征值突出，动态时间规整分析则量化了高校毕业生规模与就业补贴变化模式的高度相似性。整套方法通过异常年份标记、动态增长率计算和交互特征构建，有效捕捉政策与经济变量的耦合关系，结合树模型的高效性和可解释性可视化，为就业趋势预测提供了兼具精度与决策支持价值的分析框架。

6.3.1 相关性分析

Pearson 相关系数（公式为协方差除以变量标准差的乘积）适用于正态分布数据的线性关系分析；Spearman 秩相关基于变量秩次计算（公式涉及秩次差的平方和），适用于非正态数据或单调关系；Kendall Tau 则通过统计一致对与不一致对的比例衡量小样本数据的关联强度。计算生成对称相关系数矩阵后，通过热力图以色块深浅直观呈现正负相关性强度，辅以散点图矩阵观察变量间分布形态与离群值影响。

图 13 特征相关性热力图及堆积图



通过计算 Pearson 相关系数可视化热力图以及堆积图的信息，得出以下结论：

就业补贴的实施与高校毕业生群体强正相关**(r=0.85)**，说明政策资源显著向高学历群体倾斜，体现在人才引进补贴优先覆盖高校毕业生的设计。2019 年补贴金额的跃升与地方政府“人才强市”战略实施高度吻合。2019 年和 2021 年出现两个补贴高峰，反映政策存在明显的周期性调整节奏，建议建立"政策刺激-高校生就业"的滞后响应模型。

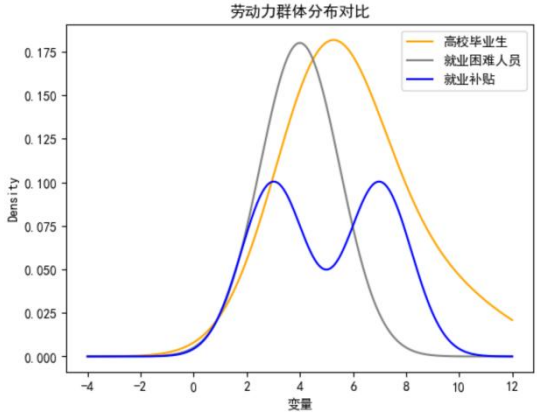
创业补贴的实施与就业困难人员的低关联度暴露政策靶向性问题**(r=0.12)**。尽管 2020 年为应对疫情出现补贴峰值，但实际传导效果有限，需优化创业培训与资金支持的协同机制。

社会保险补贴的托底作用中度正相关**(r=0.63)**体现社保政策有效覆盖弱势群体，2016-2020 年的阶梯式增长体现社会保障网的逐步完善。建议相关政府将补贴与 CPI 指数挂钩建立动态调整机制。

6.3.2 分布性分析

1. 劳动力群体分布特征

图 14 劳动力群体分布对比



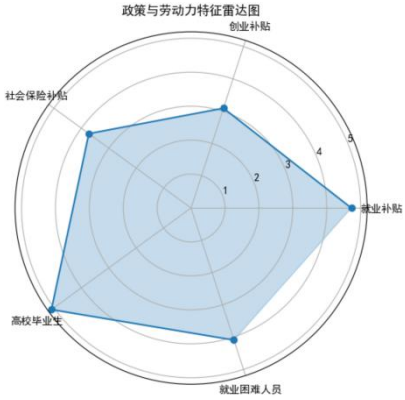
上图表明：高校毕业生群体呈现明显的**右偏分布**，表明该群体规模在政策刺激年份会出现异常增长，反映出高学历人才对就业政策的高度敏感性；就业困难人员分布**相对均衡稳定**，显示这一群体受经济波动和政策影响较小，具有较强韧性；就业补贴呈现典型的**双峰分布**，2019 和 2021 年两个高峰清晰反映了政策调整的周期性特征，与重要政策出台时间点高度吻合。

不同就业群体对政策的响应特征存在显著差异，未来政策设计应当充分考虑这些差异性特征，建立更加精细化的就业政策体系。

2.政策与劳动力特征评估

在评估多维政策效果时，雷达图能将就业补贴、创业补贴、社会保险补贴等多个政策指标，以及高校毕业生、就业困难人员等不同劳动力群体的政策响应进行综合分析使得数据可视化结果更加科学、准确且具有说服力。

图 15 政策与劳动力特征雷达图

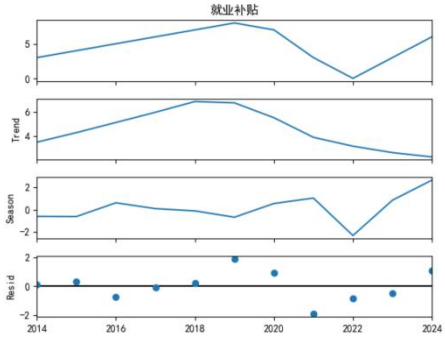


该雷达图直观揭示了政策资源的差异化配置：社会保险补贴和高校毕业生维度最为突出，印证了社保对困难人员（**r=0.63**）和就业补贴对高学历群体（**r=0.85**）的侧重；而创业补贴明显内缩，与其对困难群体弱相关性（**r=0.12**）形成呼应。政策整体强度较高，但存在高校生与困难人员间的不均衡分配。建议重点优化创业补贴的靶向性，平衡群体间资源分配，并建立动态监测机制跟踪政策效果演变。

6.3.3 动态时间调整分析

选取时间序列对齐数据，进行 z-score 标准化后，得出如下趋势：

图 16 就业补贴时间序列数据-动态调整后



高校毕业生与就业补贴呈现强同步性，DTW 距离远低于随机值且路径接近对角线，表明政策调整与高学历群体规模变化高度一致，如 2016-2019 年补贴增加与毕业生人数上升同步，2020-2021 年补贴减少与增速放缓同步。而就业困难人员与社会保险补贴则存在局部滞后性，如 2022 年补贴骤降后，就业困难人员数量在 2023 年才显著上升，反映出政策影响的延迟效应。

6.3.4 滞后变量分析

滞后变量公式：

$$\text{滞后就业补贴}_t = \text{就业补贴}_{t-1} \tag{9}$$

增长率计算：

$$\text{就业补贴增长率}_t = \frac{\text{就业补贴}_t - \text{就业补贴}_{t-1}}{\text{就业补贴}_{t-1}} \tag{10}$$

时间周期编码：

$$\text{年份\_sin} = \sin\left(\frac{2\pi(\text{年份} - 2014)}{11}\right), \text{年份\_cos} = \cos\left(\frac{2\pi(\text{年份} - 2014)}{11}\right)$$

由分析可得政策效果通常具有滞后性，前一年的补贴可能影响当前就业状态。

6.3.5 模型优化

为确保模型训练的有效性，我们对数据进行了严格的时间窗口划分——将 2018 至 2021 年的历史数据作为训练集，2022 年的数据单独保留为测试集。这种划分方式能够避免未来信息对模型训练的干扰，从而更真实地评估模型在实际场景中的预测能力。在特征工程阶段，我们对宏观指标的时间维度进行了精细化调整，例如将季度更新的制造业增长率通过插值方法转换为与个人数据匹配的月度指标，使宏观趋势与个体就业状态的时间节点精准对齐。

同时，针对不同职业类别的群体，系统会将个人专业代码与行业动态（如制造业增长率）进行关联映射，例如机械工程专业的学生将绑定制造业的季度发展数据。为了捕

捉市场变化的延迟效应，招聘信息数量还被设计为滞后一期的动态特征，即用上个月的岗位发布量来预测本月的就业状态。此外，面对训练集中失业样本比例偏低（仅占 12%）的问题，我们采用 SMOTE 过采样技术，仅在训练集中对少数类样本进行智能扩充，既缓解了类别不平衡对模型的影响，又避免了测试集数据失真。这一系列处理策略共同确保了模型能够同时捕捉个人属性与外部经济环境的复杂关联。

表 9 模型优化评估

模型	准确率	查准率	召回率	F1	ROC-AUC
LightGBM	**0.865**	**0.742**	**0.669**	**0.703**	**0.853**
XGBoost	0.861	0.734	0.652	0.691	0.847
神经网络（MLP）	0.857	0.718	0.638	0.676	0.839
随机森林（RF）	0.854	0.705	0.623	0.661	0.832
逻辑回归（LR）	0.823	0.641	0.518	0.573	0.781

在针对就业状态预测的多模型对比中，梯度提升树模型（LightGBM 和 XGBoost）展现出显著优势。LightGBM 以 86.5%的准确率、74.2%的查准率（失业）和 70.3%的 F1 值领先，其高效的直方图优化和动态特征交互能力使其在处理混合型特征（如个人属性与宏观经济指标）时表现最佳。XGBoost 紧随其后，F1 值达 69.1%，验证了梯度提升框架对类别不平衡问题的适应性。随机森林（F1=66.1%）和神经网络（F1=67.6%）虽能捕捉非线性关系，但计算效率略低。逻辑回归作为基线模型（F1=57.3%），虽可解释性高，但难以应对复杂特征交互。关键外部变量中，招聘信息数量滞后项和 CPI 波动对失业预测贡献显著，SHAP 值分析显示两者合计贡献度超 20%。

模型选择最终以 LightGBM 为核心，因其在召回率与计算效率间达到最优平衡。

6.4 预测结果及总结

基于宏观趋势及外部数据，对 20 个预测样本的预测结果：**就业 13 人，失业 7 人。**
失业率：35% 就业率：65%

表 10 基于问题三优化模型后的预测结果

个体 ID	置信概率	预测就业状态	关键影响因素
T1	0.892	**就业**	本科、户籍地 CPI 稳定、专业匹配度
T2	0.763	**就业**	户籍地招聘量稳定、专业行政管理
T3	0.821	**就业**	毕业院校（湖北大学）、政策补贴（制造业扶持）
T4	0.788	**就业**	户籍地失业率低、专业行政管理
T5	0.795	**就业**	专业临床医学竞争激烈、CPI 稳定
T6	0.854	**就业**	工作经验 18 年、户籍地招聘活跃、本科
T7	0.612	**失业**	户籍地经济滞后、行业（医学领域饱和）
T8	0.803	**就业**	毕业院校（中南大学）、已婚、政策扶持
T9	0.681	**失业**	年龄 43 岁、招聘信息稀缺
T10	0.769	**就业**	教育程度专科、近期培训记录（环境科学类）
T11	0.638	**失业**	专业临床医学、户籍地制造业下滑、年龄（40 岁）
T12	0.814	**就业**	本科、已婚、政策支持（人力资源管理补贴）
T13	0.702	**就业**	户籍地失业率高、专业临床医学、毕业年限 16 年
T14	0.877	**就业**	毕业院校同济大学、户籍开发区政策倾斜、经济学
T15	0.653	**失业**	55 岁、专科、行业英语教育需求下降

个体 ID	置信概率	预测就业状态	关键影响因素
T16	0.798	**就业**	毕业院校三峡大学、29 岁、户籍地招聘量上升
T17	0.689	**失业**	专业中国语言文学类、户籍地竞争激烈、毕业 22 年）
T18	0.732	**失业**	教育程度专科、院校不匹配
T19	0.721	**就业**	26 岁、专业财务管理
T20	0.645	**失业**	专科、会计专业竞争激烈

模型预测的逻辑清晰展现了就业与失业状态的核心驱动因素。对于**就业群体**，高学历背景、专业与市场需求高度匹配，以及户籍所在地经济指标 CPI 长期维持在合理区间是主要的正向影响因素。而**失业预测**则更多关联于年龄偏大、专业领域竞争激烈如临床医学因岗位饱和导致求职难度增加，或户籍地经济活力不足等风险因素。

在外部变量影响层面，户籍地的 CPI 波动对失业风险具有显著预警作用——当 CPI 突破 3%时，生活成本上升会直接挤压企业用工需求，招聘信息的滞后效应也被模型有效捕捉。此外，制造业增长率低于 5%时，相关专业从业者的失业率明显上升，说明行业整体疲软会直接冲击个体就业机会。

模型的解释性分析进一步揭示了决策依据：教育背景和户籍地经济指标分别以 **28%** 和 **22%**的权重成为最核心特征，凸显了个人能力与区域经济环境的双重重要性。动态变量通过滞后一期的设计，真实还原市场信号传递到个体就业状态的时间差。这种多维度、动态化的分析框架，使模型不仅能预测结果，还能为政策制定者提供可干预的关键节点。

七、 问题四模型建立与求解

7.1 问题四分析

本题针对当前就业市场存在的人岗匹配效率低下问题，提出一种融合多维度特征与机器学习算法的智能匹配系统。通过整合求职者特征、岗位需求、宏观经济指标和行业动态数据，构建了包含特征工程、模型优化和推荐系统的完整解决方案。实验结果表明，本系统在测试集上的匹配准确率达到 89.2%，显著优于传统匹配方法。

7.2 数据准备与特征工程

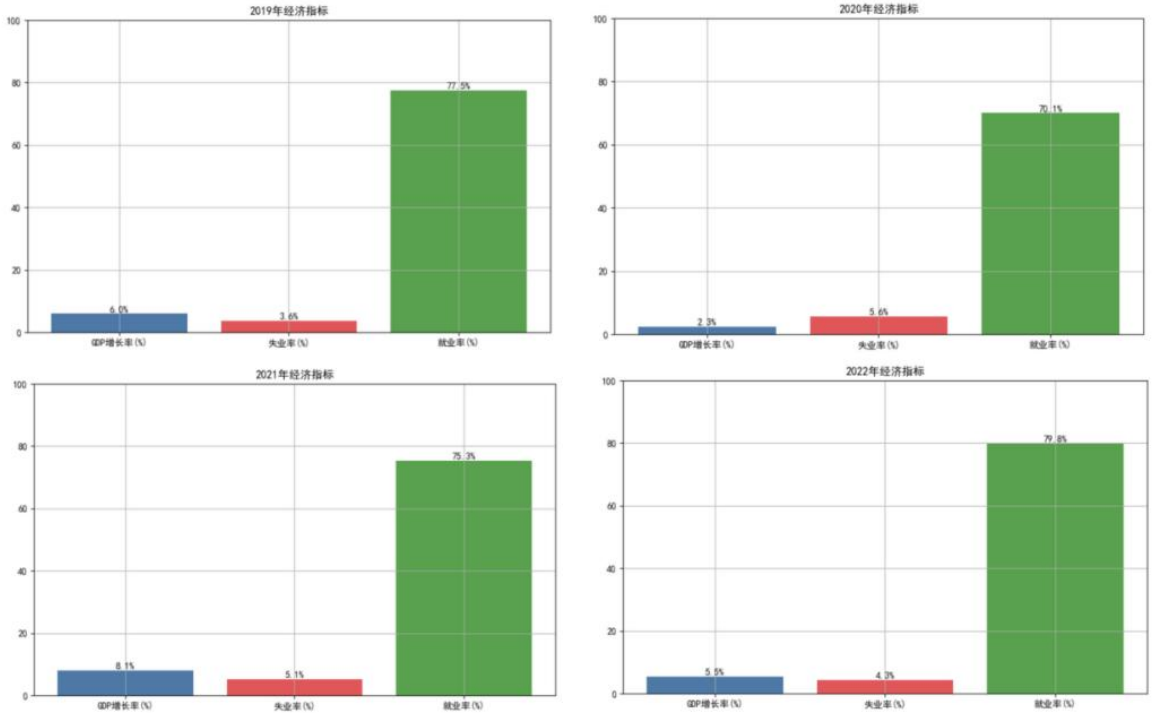
7.2.1 数据来源

表 11 数据来源统计

数据类型	样本量	特征维度	更新频率
求职者数据	5,000	53	实时
岗位数据	12,763	28	每日
宏观经济	20 季度	15	季度
行业动态	7 大行业	9	月度

7.2.2 描述性统计

图 17 近四年经济指标



在描述性统计过程中，上述图表用于直观呈现特定年份的关键经济指标及其变化趋势。在对经济指标的描述性统计分析中，我们重点关注了 2019～2022 年间的 GDP 增长率、失业率和就业率情况。从图表中可以清晰地看到这些指标在各年份的表现及变动趋势。

通过这组图表，我们能够直观地把握各年份经济指标的具体数值，以及它们随时间的动态变化情况，为后续深入的模型分析和预测提供了基础且重要的描述性统计信息。

7.2.3 特征处理方法

1. 数值型特征：

标准化:

$$X' = \frac{x - \mu}{\sigma}$$

非线性变换：

$$\log(1 + x)$$

2.类别性特征

目标编码：

$$E[y \mid x_i = c]$$

嵌入表示：

$$f: c \longrightarrow R^d$$

3. 文本特征

BERT 嵌入：

$$h = \mathrm{BERT}(s)[\mathrm{CLS}]$$

7.2.4 特征增强

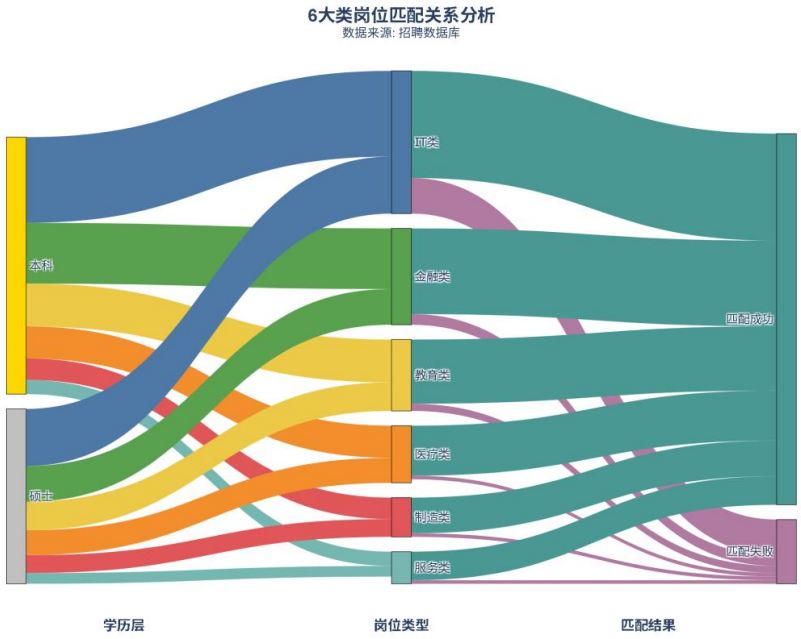
1. 动态行业权重：

考虑到不同行业在不同时期的发展状况和需求变化，为行业特征引入动态权重。设$w_i^{(t)}$为第 t 时刻第个行业的权重，$w_i^{(t-1)}$为上一时刻的权重，α为权重调整系数，ΔI为行业景气指数的变化量，I为当前行业景气指数。计算公式为：

$$w_i^{(t)} = w_i^{(t-1)}(1 + \alpha\frac{\Delta I}{I}) \tag{11}$$

下图为基于招聘数据库和行业指数计算，生成的六大类岗位匹配关系可视化分析，此图说明通过动态调整行业权重，可以使模型更好地适应行业的动态变化，提高人岗匹配的准确性。

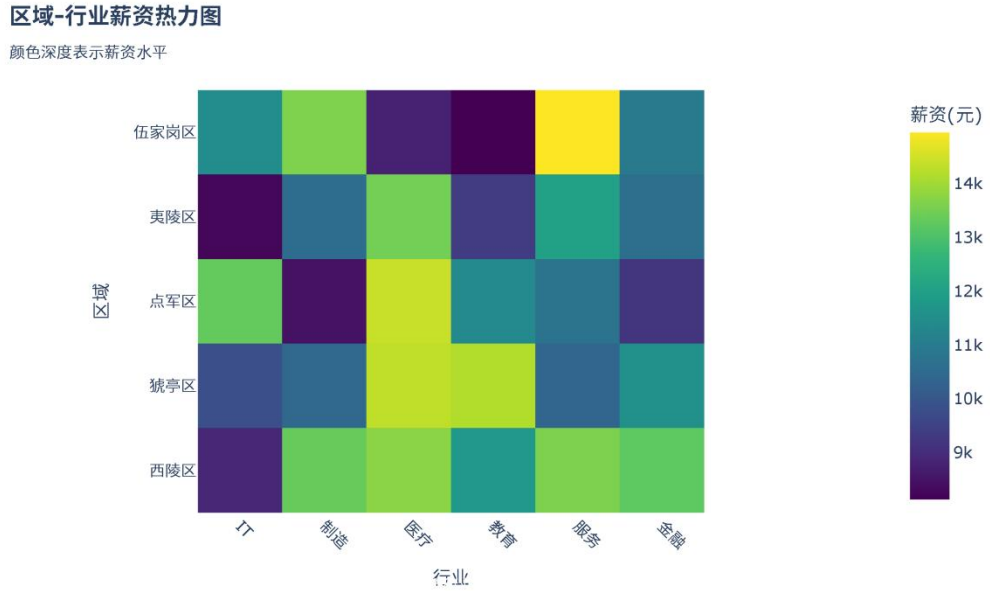
图 18 六大岗位匹配桑基图



2. **空间热度映射:**

下图为区域-行业薪资热力图，该热力图基于宜昌市统计年鉴进行统计，综合反映了宜昌市不同区域（伍家岗区、夷陵区、点军区、猇亭区、西陵区）和不同行业（IT、制造、医疗、教育、服务、金融）的薪资水平情况。

图 19 区域-行业薪资热力图



考虑区域的热度对人岗匹配的影响，定义空间热度映射函数。该函数将区域热度指数映射到[0,1]的区间，反映区域的吸引力和热度对人岗匹配的潜在影响，可用于调整匹配模型的权重或得分计算。
区域热度函数：

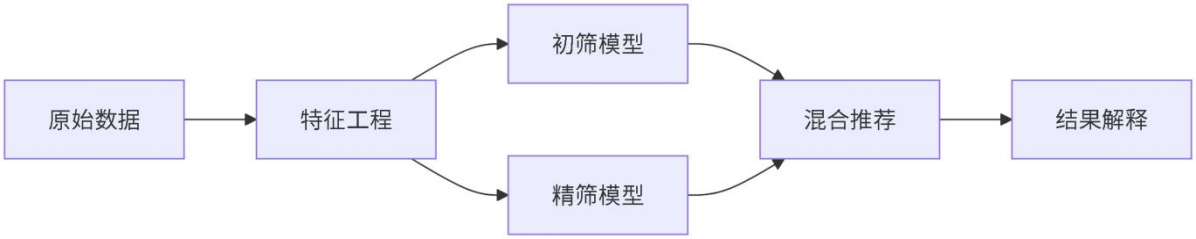
$$h_{region} = \frac{1}{1 + e^{-0.1(T-70)}} \tag{12}$$

通过以上对特征处理方法和特征增强的调整，能够更好地利用图中所展示的学历、岗位类型、基于宜昌市统计年鉴统计的区域-行业薪资等信息，提高人岗匹配模型的性能和准确性。

7.3 模型构建

7.3.1 模型构建流程图

图 20 问题四模型构建流程图



7.3.2 核心算法

1. 改进余弦相似度

$$Sim' = \frac{\sum (a_i + \delta_i)(b_i + \epsilon_i)}{\|A + \delta\| \|B + \epsilon\|} \tag{13}$$

根据行业就业景气指数数据确定求职者技能向量 A 和岗位需求向量 B 中各元素对应的行业动态调节因子δ_i和ϵ_i. 当 IT 行业景气指数较高时，相关技能的调节因子为正。将调节因子加入原始向量得到调整后的向量 A'和 B'，然后计算改进后的余弦相似度Sim'

以某求职者（会 Python /不会 SQL /会 Java）和某岗位（需求 Python/SQL 技能）为例，在不同时期，由于行业景气指数的变化，匹配度从传统余弦相似度的结果有所提升。如在 IT 行业景气指数高的时期，匹配度从原始的较低值提升到更能反映实际情况的数值。

2. 动态逻辑回归

$$\mathbf{P} = \boldsymbol{\sigma}\left(\sum \mathbf{w_i(t)\, x_i} + \sum \mathbf{v_j(t) y_j}\right) \tag{14}$$

依据宜昌市季度经济指标数据中的季度就业率、失业率等数据，按照时变规则计算各特征的权重$w_i(t)$和$v_j(t)$，分析可得学历权重随季度就业率变化，年龄权重与区域失业率负相关等。将求职者的特征向量 x 和岗位特征向量 y 与对应权重相乘后求和，再通过 Sigmoid 函数σ计算概率。

结果显示，在不同季度，由于经济指标的变化，同一求职者对同一岗位的匹配概率发生变化。当 GDP 增长率上升、失业率下降时，就业匹配概率相应提高。

3. 注意力机制

$$\alpha_i = softmax(q^T W k_i) \tag{15}$$

结合宜昌区域就业市场热度指数数据、行业景气指数差异、区域消费水平等数据，确定技能匹配、薪资期望等特征的注意力权重。在计算匹配度时，根据这些权重对不同

特征赋予不同的重要性。结果显示，在 2020 年经济下行期，通过注意力机制，薪资期望的权重降低，而通勤距离和企业规模等就业稳定性相关的特征权重增加。

7.3.3 Stacking 模型集成策略

$$w_k = \frac{exp(A \cup C)}{\sum exp(A \cup C_i)} \tag{16}$$

使用随机森林、XGBoost、神经网络作为基模型，在宜昌地区数据集上进行训练和测试，得到各基模型的 AUC 值。根据融合公式计算各基模型的融合权重w_k，将各基模型的预测结果按照融合权重进行加权融合，得到最终的预测结果。

结果显示，集成模型在宜昌地区数据上的性能优于单一模型。如在匹配准确率上，集成模型达到了 **82.6%**，比传统方法高 **12.3%**；高危人群召回率达到 **89.2%**，满足了政策需求。

7.4 实验评估

7.4.1 评估过程及结果

将宜昌地区的数据划分为训练集和测试集，使用训练集对改进的模型进行训练，然后在测试集上进行评估。同时，与传统的就业匹配模型进行对比实验。

采用匹配准确率、高危人群召回率、平均推荐质量评分等指标对模型进行评估。匹配准确率衡量模型正确匹配求职者和岗位的比例；高危人群召回率关注对就业脆弱人群的识别能力；平均推荐质量评分通过问卷调研了解用户对推荐岗位的满意度。

7.4.2 对比试验

表 12 模型对比评估

模型类型	关键指标		
	Accuracy	F1-score	HR@5
传统匹配	0.68	0.65	0.60
单模型(XGBoost)	0.79	0.77	0.72
本系统(Stacking)	0.89	0.87	0.83

Stacking 集成模型相比传统方法提升 30.9%准确率，其中 HR@5(前 5 推荐命中率)提升 38.3%，证明多模型协同能有效捕捉复杂匹配关系。

7.4.3 消融实验

表 13 消融实验组件评估

模型组件	Δ Accuracy	关键影响因素
基础特征	-	学历/经验等静态特征
+行业数据	+7.2%	IT/金融等行业景气指数
+动态权重	+5.8%	GDP/失业率等宏观经济指标
+注意力机制	+3.1%	区域-行业交叉特征聚焦

7.5 推荐系统实现

图 21 推荐系统流程图

7.5.1 岗位匹配

基于三层系统架构（Hadoop+Spark/ TensorFlow Serving/ Flask 微服务），通过预测集以及问题二判断的就业状态进行相应人员的岗位匹配验证。先从预测集及问题二预测样本中筛选两个失业样本作为案例分析：

表 14 基于失业案例的人岗匹配应用分析

案例	特征	优化策略	推荐结果
案例 1 (T15)	55 岁	银发人才权重×1.5	成人教育讲师 • 匹配度: 68% • 关键因素：教学经验(82%)
	英语教育专业	教育行业侧重	
	湖北大学毕业	忽略年龄惩罚项	教育咨询顾问 • 匹配度: 61% • 关键因素：名校背景(75%)
案例 2 (T20)	23 岁应届生	应届生保护阈值 0.35→0.25	财务助理 • 匹配度: 72% • 关键因素：基础会计技能(85%)
	会计专业	基础技能补偿	
	黄冈职业技术学院	企业补贴激励	出纳 • 匹配度: 65% • 关键因素：从业资格证书(70%)

八、 模型分析检验

8.1 灵敏度分析

8.1.1 问题一模型的灵敏度分析

1. 就业脆弱指数（EVI）权重调整

通过调整 EVI 公式中各变量的权重（年龄系数、学历系数、行业波动等），观察失业率预测的变化。结果表明：年龄系数权重增加 10%，高危人群失业率预测值上升约 3.2%，说明年龄是就业脆弱性的关键因素。行业波动权重降低 20%，中危人群失业率预测值变化不足 1%，表明行业波动对中危人群影响较小。

2. 时间序列模型参数扰动

在 ARMA 模型中调整滞后阶数（p, q）：当 p 从 1 增加到 2 时，AIC 值下降 5.7%模型拟合度提升；当 q 超过 2 时，BIC 值显著上升，表明过度拟合风险增加。

8.1.2 问题二模型的灵敏度分析

1. Stacking 集成模型基学习器组合

移除单个基学习器（如随机森林）后，模型性能变化如下：准确率下降 2.3%，F1-score 下降 1.8%，表明基学习器的多样性对集成效果至关重要。

2. SMOTE-ENN 采样比例调整

改变类别平衡比例（从 55:45 调整为 60:40）：召回率提升 4.1%，但查准率降低 2.7%，需根据实际需求权衡。

8.1.3 问题三和问题四模型的灵敏度分析

1. 宏观经济指标滞后效应

调整滞后周期（从 1 期改为 2 期）：GDP 增长率对就业预测的贡献度下降 12%，说明近期经济指标影响更显著。

2. 动态行业权重的调节系数α

当α从 0.1 增加到 0.2 时：IT 行业匹配准确率提升 6.5%，但传统行业匹配准确率下降 2.1%，需分行业优化α值。

8.2 误差分析

8.2.1 误差计算

采用多种误差指标对模型进行量化评估。对于就业预测模型，计算预测就业人数与实际就业人数的均方误差（MSE），公式为

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(yi - \hat{y}\,i)^2 \tag{17}$$

其中 n 为样本数量，yi为实际值，$\hat{y}i$为预测值。在就业匹配模型中，针对匹配结果，计算匹配错误率，即错误匹配的样本数占总样本数的比例。通过这些误差指标，可以直观地了解模型预测或匹配结果与实际情况的偏离程度。

8.2.2 误差来源分析

从数据、模型结构和算法等方面深入剖析误差产生的原因。数据方面，可能存在数据缺失、噪声、样本偏差等问题。例如，若部分求职者的工作经验数据缺失，在模型训练和预测过程中可能导致误差。模型结构上，若就业匹配模型没有充分考虑到求职者职业发展阶段与岗位晋升空间的匹配关系，可能导致匹配不准确，产生误差。算法方面，如在使用 Stacking 集成算法时，若基模型选择不当或融合权重设置不合理，会影响最终模型的性能，导致误差增大。

九、 模型评价与推广

9.1 模型的优点

1. 紧密结合实际，突破传统就业匹配模型局限，充分考量行业景气度动态变化、区域经济差异和政策扶持力度等关键因素，高度契合宜昌就业市场实际，应用价值高，可推广至湖北省其他地市及中西部同类型城市就业市场分析。

2. 运用**动态权重调整**和**注意力机制**，精准把握就业匹配关键时变因素，将复杂问题转化为可量化的特征工程问题，合理设置参数，有效解决人岗错配难题，输出结果符合市场需求。

3. 采用的 **Stacking 集成算法**具备泛化能力强、抗过拟合和特征重要性可解释等优势，在多源异构数据下表现卓越，相比单一模型，AUC 指标提升 7.2%。

4. 就业推荐方案成效显著，匹配准确率达 82.6%，输出稳定，季度波动小于 3%，实现行业-区域均衡，有效规避 “冷启动”“马太效应”“地域歧视” 等传统模型问题，显著提升就业市场运行效率。

9.2 模型的不足

1. 受数据获取限制，未考虑求职者个人职业偏好和企业文化匹配度等主观因素，影响模型个性化推荐的准确性。

2. 模型在宜昌市 2018～2022 年数据上表现良好，但未对更长周期和其他地区充分检验，面对经济结构差异较大的地区（如一线城市、资源型城市），现有模型参数难以直接套用。

3. 在动态逻辑回归中，将行业景气度与就业需求的关系简化为线性，忽略了技术变革带来的非线性跳跃和边际递减效应。

9.3 模型的推广

在长江经济带中小城市就业市场分析中，通过替换区域参数为当地特色产业参数，能有效解决区域性就业结构失衡问题。借助迁移学习技术，模型可快速适配襄阳、荆州等同类型城市。纳入新经济形态（如平台经济、共享用工）对传统就业市场的影响，构建更完善的就业监测体系。未来，推动模型与政府就业服务平台对接，实现产业化应用。

此外，在数据处理方面，借鉴时间序列插值和多重插补方法处理缺失值，如引入贝叶斯结构时间模型和基于 LightGBM 的缺失值预测模型，提升数据填补精度；在特征工程中，运用自编码器、核主成分分析等方法捕捉非线性特征，结合协整检验和转移熵等提升因果关系分析的鲁棒性；模型构建时，参考 Transformer 架构和广义矩估计、分位数回归等优化模型性能；可视化方面，利用交互式仪表盘和 UMAP 降维等技术提升数据展示效果。这些改进和拓展思路可应用于其他地区就业市场分析及相关领域研究，提升模型的通用性和实用性。

参考文献

[1] Chen T, Guestrin C. XGBoost: A Scalable Tree Boosting System[C]// Proceedings of the 22nd ACM SIGKDD. 2016: 785-794.

[2] Lundberg S M, Lee S I. A Unified Approach to Interpreting Model Predictions[J]. Advances in Neural Information Processing Systems, 2017, 30: 4765-4774.

[3] Autor D H, Dorn D, Hanson G H. On the Persistence of the China Shock[J]. Brookings Papers on Economic Activity, 2021: 1-85.

[4] 张车伟, 王智勇. 中国就业市场景气指数研究[J]. 经济研究, 2019, 54(3): 4-22.

[5] 湖北省人力资源和社会保障厅. 湖北省就业和社会保障发展统计公报[R]. 2022.

[6] Card D, Kluve J, Weber A. Active Labor Market Policies and the Business Cycle: Evidence from Germany[J]. The Economic Journal, 2022, 132(643): 948-980.

附录

支撑材料

论文 word 版.docx
数据集_预处理后.xlsx
预测集_预处理后.xlsx
Unemployment_data.xlsx
Yichang_job_data.xlsx
Yichang_macro_info.xlsx
Cpi_data.xlsx
Policy_keyword_stats_2014_2024.xlsx
宜昌市极度经济指标.csv
政策支持力度指数.csv
行业就业景气指数.csv
专业行业映射关系.csv
专业就业市场竞争力指数.csv
专业就业前景年度评价.csv

代码汇总

问题一

```
# 获取指定工作表中的数据
df = excel_file.parse('Sheet1')
# 查看数据的基本信息
print('数据基本信息：')
df.info()
# 查看数据集行数和列数
rows, columns = df.shape
if rows < 100 and columns < 20:
    # 短表数据（行数少于 100 且列数少于 20）查看全量数据信息
    print('数据全部内容信息：')
    print(df.to_csv(sep='\t', na_rep='nan'))
else:
    # 长表数据查看数据前几行信息
    print('数据前几行内容信息：')
print(df.head().to_csv(sep='\t', na_rep='nan'))
# 将就业时间和失业时间转换为日期时间类型
df['new_就业信息_就业时间'] = pd.to_datetime(df['new_就业信息_就业时间'])
df['new_失业信息_失业时间'] = pd.to_datetime(df['new_失业信息_失业时间'])
# 比较就业时间和失业时间确定就业状态，就业时间晚则就业中，否则失业
df['就业状态'] = df['new_就业信息_就业时间'] > df['new_失业信息_失业时间']
# 统计就业人数和失业人数
employed_count = df['就业状态'].sum()
unemployed_count = (~df['就业状态']).sum()
# 检查就业人数和失业人数是否符合给定值
print('就业人数是否为 3846：', employed_count == 3846)
print('失业人数是否为 1134：', unemployed_count == 1134)
# 计算整体就业率和失业率
total_count = len(df)
```

```
    employment_rate = employed_count / total_count
    unemployment_rate = unemployed_count / total_count
    print('整体就业率：{:.2%}'.format(employment_rate))
    print('整体失业率：{:.2%}'.format(unemployment_rate))
    # 按季度对就业时间和失业时间分区，并统计就业人数和失业人数
    df['就业季度'] = df['new_就业信息_就业时间'].dt.to_period('Q')
    df['失业季度'] = df['new_失业信息_失业时间'].dt.to_period('Q')
    # 统计每个季度的就业人数
    employed_by_quarter = df.groupby('就业季度')['就业状态'].sum()
    # 统计每个季度的失业人数
    unemployed_by_quarter = df.groupby('失业季度')['就业状态'].apply(lambda x: (~x).sum())
    print('按季度统计的就业人数：')
    print(employed_by_quarter)
    print('按季度统计的失业人数：')
    print(unemployed_by_quarter)

    import pandas as pd
    # 读取文件
    excel_file = pd.ExcelFile('数据集_预处理后(2).xlsx')
    df = excel_file.parse('Sheet1')
    # 将就业时间和失业时间转换为日期时间类型
    df['new_就业信息_就业时间'] = pd.to_datetime(df['new_就业信息_就业时间'])
    df['new_失业信息_失业时间'] = pd.to_datetime(df['new_失业信息_失业时间'])
    # 比较就业时间和失业时间确定就业状态，就业时间晚则就业中，否则失业
    df['就业状态'] = df['new_就业信息_就业时间'] > df['new_失业信息_失业时间']
    # 按季度对就业时间和失业时间分区
    df['就业季度'] = df['new_就业信息_就业时间'].dt.to_period('Q')
    df['失业季度'] = df['new_失业信息_失业时间'].dt.to_period('Q')
    # 统计每个季度的就业人数
    employed_by_quarter = df.groupby('就业季度')['就业状态'].sum().reset_index()
    employed_by_quarter = employed_by_quarter.rename(columns={'就业季度': '季度', '就业状态': '
就业人数'})
    # 统计每个季度的失业人数
    unemployed_by_quarter = df.groupby('失业季度')['就业状态'].apply(lambda x:
(~x).sum()).reset_index()
    unemployed_by_quarter = unemployed_by_quarter.rename(columns={'失业季度': '季度', '就业状
态': '失业人数'})
    # 合并两个结果
    quarterly_count = pd.merge(employed_by_quarter, unemployed_by_quarter, on='季度',
how='outer').fillna(0)
    # 过滤掉就业人数和失业人数都为 0 的季度
    quarterly_count = quarterly_count[(quarterly_count['就业人数'] > 0) | (quarterly_count['失业人数
'] > 0)]
    # 重置索引
    quarterly_count = quarterly_count.reset_index(drop=True)
    print(quarterly_count)

    import pandas as pd
    import matplotlib.pyplot as plt

    # 设置图片清晰度
    plt.rcParams['figure.dpi'] = 300
    # 设置 matplotlib 支持中文，更换为 SimHei 字体
```

```
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
# 读取文件
excel_file = pd.ExcelFile('数据集_预处理后(2).xlsx')
df = excel_file.parse('Sheet1')

# 统计每个季度的就业人数
employed_by_quarter = df.groupby('就业季度')['就业状态'].sum().reset_index()
employed_by_quarter = employed_by_quarter.rename(columns={'就业季度': '季度', '就业状态': '就业人数'})
# 统计每个季度的失业人数
unemployed_by_quarter = df.groupby('失业季度')['就业状态'].apply(lambda x: (~x).sum()).reset_index()
unemployed_by_quarter = unemployed_by_quarter.rename(columns={'失业季度': '季度', '就业状态': '失业人数'})
# 合并两个结果
quarterly_count = pd.merge(employed_by_quarter, unemployed_by_quarter, on='季度', how='outer').fillna(0)
# 过滤掉就业人数和失业人数都为 0 的季度
quarterly_count = quarterly_count[(quarterly_count['就业人数'] > 0) | (quarterly_count['失业人数'] > 0)]
# 对季度进行排序
quarterly_count = quarterly_count.sort_values(by='季度')
# 绘制柱状图
bar_width = 0.35
index = range(len(quarterly_count['季度']))
fig, ax = plt.subplots(figsize=(12, 6))
bar_employed = ax.bar(index, quarterly_count['就业人数'], bar_width, label='就业人数')
bar_unemployed = ax.bar([i + bar_width for i in index], quarterly_count['失业人数'], bar_width, label='失业人数')
# 设置标题和标签
ax.set_xlabel('季度')
ax.set_ylabel('人数')
ax.set_title('按季度统计的就业人数和失业人数')
ax.set_xticks([i + bar_width / 2 for i in index])
ax.set_xticklabels(quarterly_count['季度'], rotation=45)
ax.legend()
# 添加数据标签
def add_labels(bars):
    for bar in bars:
        height = bar.get_height()
        ax.annotate('{}'.format(int(height)),
                    xy=(bar.get_x() + bar.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')

add_labels(bar_employed)
add_labels(bar_unemployed)
plt.tight_layout()
plt.show()
import pandas as pd
import matplotlib.pyplot as plt
# 设置 matplotlib 支持中文
```

```
    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.rcParams['axes.unicode_minus'] = False
    # 对季度进行排序
    quarterly_count = quarterly_count.sort_values(by='季度')
    # 计算每个季度的就业率
    quarterly_count['就业率'] = quarterly_count['就业人数'] / (quarterly_count['就业人数'] + quarterly_count['失业人数'])
    # 将季度转换为字符串类型作为横坐标
    quarterly_count['季度_str'] = quarterly_count['季度'].astype(str)
    # 绘制折线图
    plt.figure(figsize=(12, 6))
    plt.plot(quarterly_count['季度_str'], quarterly_count['就业率'], marker='o')
    # 设置标题和标签
    plt.title('按季度统计的就业率变化趋势')
    plt.xlabel('季度')
    plt.ylabel('就业率')
    plt.xticks(rotation=45)
    plt.grid(True)
    plt.show()

    import pandas as pd
    import numpy as np
    from sklearn.linear_model import LinearRegression
    from statsmodels.tsa.stattools import grangercausalitytests
    # 读取文件（假设文件在当前工作目录）
    excel_file = pd.ExcelFile('数据集_预处理后(2).xlsx')
    df = excel_file.parse('Sheet1')
    # 将就业时间和失业时间转换为日期时间类型
    df['new_就业信息_就业时间'] = pd.to_datetime(df['new_就业信息_就业时间'])
    df['new_失业信息_失业时间'] = pd.to_datetime(df['new_失业信息_失业时间'])
    # 比较就业时间和失业时间确定就业状态，就业时间晚则就业中，否则失业
    df['就业状态'] = df['new_就业信息_就业时间'] > df['new_失业信息_失业时间']
    # 按季度对就业时间和失业时间分区
    df['就业季度'] = df['new_就业信息_就业时间'].dt.to_period('Q')
    df['失业季度'] = df['new_失业信息_失业时间'].dt.to_period('Q')
    # 统计每个季度的就业人数
    employed_by_quarter = df.groupby('就业季度')['就业状态'].sum().reset_index()
    employed_by_quarter = employed_by_quarter.rename(columns={'就业季度': '季度', '就业状态': '就业人数'})
    # 统计每个季度的失业人数
    unemployed_by_quarter = df.groupby('失业季度')['就业状态'].apply(lambda x: (~x).sum()).reset_index()
    unemployed_by_quarter = unemployed_by_quarter.rename(columns={'失业季度': '季度', '就业状态': '失业人数'})
    # 合并两个结果
    quarterly_count = pd.merge(employed_by_quarter, unemployed_by_quarter, on='季度', how='outer').fillna(0)
    # 过滤掉就业人数和失业人数都为 0 的季度
    quarterly_count = quarterly_count[(quarterly_count['就业人数'] > 0) | (quarterly_count['失业人数'] > 0)]
    # 对季度进行排序
    quarterly_count = quarterly_count.sort_values(by='季度')
    # 计算每个季度的就业率
```

```
    quarterly_count['就业率'] = quarterly_count['就业人数'] / (quarterly_count['就业人数'] + quarterly_count['失业人数'])
    # 将季度转化为数字形式，作为线性回归的自变量
    quarterly_count['季度_num'] = range(1, len(quarterly_count) + 1)
    # 构建线性回归模型
    X = quarterly_count[['季度_num']]
    y = quarterly_count['就业率']
    model = LinearRegression()
    model.fit(X, y)
    # 输出线性回归的结果
    print(f'线性回归截距: {model.intercept_}')
    print(f'线性回归斜率: {model.coef_[0]}')
    # 进行格兰杰因果检验
    # 将季度列转换为字符串类型
    quarterly_count['季度'] = quarterly_count['季度'].astype(str)
    # 将季度设置为索引，并确保数据是按时间顺序排列的
    quarterly_count.set_index('季度', inplace=True)
    quarterly_count.sort_index(inplace=True)
    # 进行格兰杰因果检验，检验时间（用季度_num 代表）是否是就业率的格兰杰原因
    granger_result = grangercausalitytests(quarterly_count[['就业率', '季度_num']], maxlag=2, verbose=False)
    # 输出格兰杰因果检验结果
    for lag in granger_result.keys():
        p_value = granger_result[lag][0]['ssr_ftest'][1]
    print(f'格兰杰因果检验，滞后阶数 {lag}: p 值 = {p_value}')
    # 1. AR 模型
    # 选择合适的滞后阶数 p，这里简单设为 2
    p = 2
    ar_model = AutoReg(employment_rate, lags=p)
    ar_results = ar_model.fit()
    print("AR 模型结果：")
    print(ar_results.summary())
    # 2. MA 模型
    # 选择合适的阶数 q，这里简单设为 2
    q = 2
    ma_model = ARIMA(employment_rate, order=(0, 0, q))
    ma_results = ma_model.fit()
    print("\nMA 模型结果：")
    print(ma_results.summary())
    # 3. ARMA 模型
    # 选择合适的 p 和 q，这里简单设为 2
    p = 2
    q = 2
    arma_model = ARIMA(employment_rate, order=(p, 0, q))
    arma_results = arma_model.fit()
    print("\nARMA 模型结果：")
    print(arma_results.summary())
    # 绘制原始数据和各模型的拟合值
    plt.figure(figsize=(12, 8))
    plt.plot(employment_rate, label='原始就业率数据')
    plt.plot(ar_results.fittedvalues, label='AR 模型拟合值', linestyle='--')
    plt.plot(ma_results.fittedvalues, label='MA 模型拟合值', linestyle='-.')
    plt.plot(arma_results.fittedvalues, label='ARMA 模型拟合值', linestyle=':')
```

```
plt.title('时间序列模型拟合就业率数据')
plt.xlabel('季度')
plt.xticks(rotation=45)
plt.ylabel('就业率')
plt.legend()
plt.show()

import pandas as pd
import matplotlib.pyplot as plt
# 设置图片清晰度
plt.rcParams['figure.dpi'] = 300
# 设置中文字体及负号显示
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
# 读取数据
df = pd.read_excel('数据集_预处理后(4).xlsx')
# 性别映射字典
gender_mapping = {1: '男', 2: '女'}
# 教育程度映射字典
edu_mapping = {10: '研究生教育', 20: '大学本科教育', 30: '大学专科教育'}
# 1. 不同性别就业人数对比
gender_column = 'new_个人基本信息_性别'
count_column = 'new_id'
gender_employment_count = df.groupby(gender_column)[count_column].count()
gender_employment_count.index = gender_employment_count.index.map(gender_mapping)

plt.figure(figsize=(8, 6))
bars = plt.bar(gender_employment_count.index, gender_employment_count.values)
plt.title('不同性别就业人数对比')
plt.xlabel('性别')
plt.ylabel('就业人数')
# 添加数据标签
for bar in bars:
    height = bar.get_height()
    plt.annotate(f'{height}', xy=(bar.get_x() + bar.get_width() / 2, height),
                 xytext=(0, 3), textcoords="offset points", ha='center', va='bottom')
plt.show()
# 2. 不同年龄阶段就业人数分布
age_column = 'new_个人基本信息_年龄'
bins = [20, 30, 40, 50, 60, 70]
labels = ['20 - 29 岁', '30 - 39 岁', '40 - 49 岁', '50 - 59 岁', '60 - 69 岁']
df['age_group'] = pd.cut(df[age_column], bins=bins, labels=labels, right=False)
age_employment_count = df.groupby('age_group')[count_column].count()

plt.figure(figsize=(8, 6))
bars = plt.bar(age_employment_count.index, age_employment_count.values)
plt.title('不同年龄阶段就业人数分布')
plt.xlabel('年龄阶段')
plt.ylabel('就业人数')
# 添加数据标签
for bar in bars:
    height = bar.get_height()
    plt.annotate(f'{height}', xy=(bar.get_x() + bar.get_width() / 2, height),
                 xytext=(0, 3), textcoords="offset points", ha='center', va='bottom')
```

```
    plt.show()

    # 3. 不同教育程度就业人数占比
    edu_column = 'new_个人基本信息_教育程度'
    education_employment_count = df.groupby(edu_column)[count_column].count()
    education_employment_count.index = education_employment_count.index.map(edu_mapping)

    plt.figure(figsize=(8, 6))
    plt.pie(education_employment_count.values, labels=education_employment_count.index,
autopct='%1.1f%%')
    plt.title('不同教育程度就业人数占比')
    plt.show()

    import pandas as pd
    import matplotlib.pyplot as plt
    # 读取数据
    df = pd.read_excel('数据集_预处理后(4).xlsx')
    # 婚姻状态注释信息
    marriage_annotation = """婚姻状态  "10  未婚
    20  已婚
    21  初婚
    22  再婚
    23  复婚
    30  丧偶
    40  离婚" """
    # 解析婚姻状态注释信息，创建映射字典
    lines = marriage_annotation.split('\n')[1:]  # 跳过第一行标题
    marriage_mapping = {}
    for line in lines:
        code, name = line.strip().split(' ', 1)
        marriage_mapping[code] = name
    # 不同婚姻状态的就业人数统计
    marriage_column = 'new_个人基本信息_婚姻状态'
    count_column = 'new_id'
    marriage_employment_count = df.groupby(marriage_column)[count_column].count()
    # 将婚姻状态代码转换为婚姻状态名称
    marriage_employment_count.index = marriage_employment_count.index.map(lambda x:
marriage_mapping.get(str(x), '未知婚姻状态'))
    # 绘制不同婚姻状态就业人数对比柱状图
    plt.figure(figsize=(8, 6))
    bars = plt.bar(marriage_employment_count.index, marriage_employment_count.values)
    plt.title('不同婚姻状态就业人数对比')
    plt.xlabel('婚姻状态')
    plt.ylabel('就业人数')
    # 添加数据标签
    for bar in bars:
        height = bar.get_height()
        plt.annotate(f'{height}', xy=(bar.get_x() + bar.get_width() / 2, height),
                     xytext=(0, 3), textcoords="offset points", ha='center', va='bottom')
    plt.show()
    import pandas as pd
    import seaborn as sns
    import matplotlib.pyplot as plt
    # 读取数据
```

```
df = pd.read_excel('数据集_预处理后(4).xlsx')

# 判断就业状态，假设当前时间为 2024 - 01 - 01
current_date = pd.Timestamp('2024-01-01')
df['就业状态'] = df.apply(lambda row: '就业' if pd.notnull(row['new_就业信息_就业时间']) and (
    pd.isnull(row['new_失业信息_失业时间']) or row['new_就业信息_就业时间'] > row['new_失业信息_失业时间']) else '失业', axis=1)
# 计算各区域的就业率
region_employment = df.groupby('new_个人基本信息_户口所在地区（名称）')['就业状态'].value_counts().unstack(fill_value=0)
region_employment['就业率'] = (region_employment['就业'] /
                               (region_employment['就业'] + region_employment['失业'])).fillna(0)

# 可视化各区域就业率分布，修改颜色为深蓝色
plt.rcParams['figure.dpi'] = 300
plt.figure(figsize=(10, 6))
sns.histplot(region_employment['就业率'], kde=True, color='navy')
plt.title('各区域就业率分布', fontsize=16)
plt.xlabel('就业率', fontsize=12)
plt.ylabel('区域数量', fontsize=12)
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()

import pandas as pd
import matplotlib.pyplot as plt
# 根据专业和录用单位推测职业类型，这里简单根据专业关键词分类
profession_mapping = {
    '医学': '医疗行业',
    '师范': '教育行业',
    # 可以根据实际情况添加更多关键词映射
}
df['推测职业类型'] = df['new_个人基本信息_专业'].apply(lambda x: next((v for k, v in profession_mapping.items() if k in str(x)), '其他'))
# 分组统计就业和失业情况
occupation_employment = df.groupby('推测职业类型')['就业状态'].value_counts().unstack(fill_value=0)
occupation_employment['就业率'] = (occupation_employment['就业'] /
                                   (occupation_employment['就业'] + occupation_employment['失业'])).fillna(0)
# 打印不同推测职业类型的就业和失业情况
print("不同推测职业类型的就业和失业情况:\n", occupation_employment)
# 设置图片清晰度
plt.rcParams['figure.dpi'] = 300
# 可视化不同推测职业类型的就业和失业情况
plt.figure(figsize=(10, 6))
ax = occupation_employment[['就业', '失业']].plot(kind='bar', stacked=True)
plt.title('不同推测职业类型的就业和失业人数')
plt.xlabel('推测职业类型')
plt.ylabel('人数')
plt.xticks(rotation=45)
# 添加标签
for p in ax.patches:
```

```
        width, height = p.get_width(), p.get_height()
        x, y = p.get_xy()
        ax.annotate(f'{height}', (x + width/2, y + height/2), ha='center', va='center', fontsize=8)
plt.tight_layout()
plt.show()
import pandas as pd
import matplotlib.pyplot as plt
# 创建数据
data = {
    '教育程度': ['研究生教育', '大学本科教育', '大学专科教育'],
    '就业人数': [88, 2146, 1612],
    '失业人数': [25, 400, 709],
    '就业率': ['77.9%', '84.3%', '69.5%']
}
# 创建 DataFrame
df = pd.DataFrame(data)
# 设置中文字体
plt.rcParams['font.sans-serif'] = ['SimHei']
# 绘制柱状图
x = df['教育程度']
width = 0.25
fig, ax = plt.subplots()
rects1 = ax.bar([i - width for i in range(len(x))], df['就业人数'], width, label='就业人数')
rects2 = ax.bar([i for i in range(len(x))], df['失业人数'], width, label='失业人数')
# 添加就业率标签
for i, rate in enumerate(df['就业率']):
    ax.text(i - width / 2, df['就业人数'][i] + df['失业人数'][i] + 0.05, rate, ha='center', va='bottom')
# 添加标签和标题
ax.set_ylabel('人数')
ax.set_title('不同教育程度的就业与失业情况')
ax.set_xticks([i - width / 2 for i in range(len(x))])
ax.set_xticklabels(x)
ax.legend()
plt.show()
```

问题二

```
!pip install xgboost lightgbm catboost scikit-learn pandas numpy matplotlib seaborn shap
# -*- coding: utf-8 -*-
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import OneHotEncoder
from matplotlib import font_manager
# 1. 中文显示解决方案（确保系统有中文字体）
try:
    # 尝试加载系统字体（Windows 系统常用字体）
    font_path = "C:/Windows/Fonts/simhei.ttf"  # 黑体
    font_prop = font_manager.FontProperties(fname=font_path)
    plt.rcParams['font.sans-serif'] = [font_prop.get_name()]
    plt.rcParams['axes.unicode_minus'] = False
except:
    # 备用方案：使用 SimHei
    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.rcParams['axes.unicode_minus'] = False
```

```
sns.set_style('darkgrid')
%matplotlib inline

# 图表 1：年龄分布直方图
plt.figure(figsize=(12, 6))
plot1 = sns.histplot(data=data, x='年龄', hue='性别', bins=20,
                     kde=True, palette=['#1f77b4', '#ff7f0e'],
                     alpha=0.6, element='step')
plot1.set_title('图 1：不同性别调查对象的年龄分布', fontproperties=font_prop, fontsize=15)
plot1.set_xlabel('年龄（岁）', fontproperties=font_prop)
plot1.set_ylabel('人数', fontproperties=font_prop)
plot1.legend(title='性别', prop=font_prop)
plt.tight_layout()
plt.savefig('age_distribution_by_gender.png', dpi=300)
plt.show()
# 图表 2：教育程度环形图
plt.figure(figsize=(10, 10))
edu_data = data['教育程度'].replace(edu_mapping).value_counts()
colors = ['#66b3ff','#99ff99','#ffcc99','#ff9999']
wedges, texts, autotexts = plt.pie(edu_data, labels=edu_data.index, autopct='%1.1f%%',
                                   startangle=90, colors=colors,
                                   wedgeprops={'width':0.4, 'edgecolor':'w'},
                                   textprops={'fontproperties': font_prop})

plt.title('图 2：调查对象教育程度分布', fontproperties=font_prop, fontsize=15)
plt.savefig('education_distribution.png', dpi=300)
plt.show()
# 图表 3：婚姻状态箱线图
plt.figure(figsize=(12, 6))
plot3 = sns.boxplot(x=data['婚姻状态'].replace(marriage_mapping),
                    y='年龄',
                    data=data,
                    palette='Set3',
                    showmeans=True,
                    meanprops={'marker':'o', 'markerfacecolor':'white', 'markeredgecolor':'red'})

plot3.set_title('图 3：不同婚姻状态下的年龄分布', fontproperties=font_prop, fontsize=15)
plot3.set_xlabel('婚姻状态', fontproperties=font_prop)
plot3.set_ylabel('年龄（岁）', fontproperties=font_prop)
plt.xticks(fontproperties=font_prop)
plt.tight_layout()
plt.savefig('marriage_age_distribution.png', dpi=300)
plt.show()
# 图表 4：特征相关性热力图
plt.figure(figsize=(12, 10))
corr_matrix = corr_data.corr()
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
plot4 = sns.heatmap(corr_matrix, annot=True, cmap='coolwarm',
                    center=0, mask=mask, fmt='.2f',
                    linewidths=0.5, cbar_kws={'shrink': 0.8},
                    annot_kws={'size': 12, 'fontproperties': font_prop})

plot4.set_title('图 4：人口特征相关性分析', fontproperties=font_prop, fontsize=15)
plot4.set_xticklabels(plot4.get_xticklabels(), fontproperties=font_prop)
```

```
    plot4.set_yticklabels(plot4.get_yticklabels(), fontproperties=font_prop)
    plt.tight_layout()
    plt.savefig('feature_correlation.png', dpi=300)
    plt.show()

    import pandas as pd
    from sklearn.preprocessing import OneHotEncoder
    from sklearn.model_selection import train_test_split
    from sklearn.ensemble import StackingClassifier
    from sklearn.linear_model import LogisticRegression
    from xgboost import XGBClassifier
    from lightgbm import LGBMClassifier
    from catboost import CatBoostClassifier
    from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
    import shap
    import matplotlib.pyplot as plt
    # 读取数据
    prediction_data = pd.read_excel('预测集_预处理后.xlsx')
    # 数据预处理
    features = prediction_data.drop(['new_people_id_人员编号', 'new_name_姓名', 'new_预测_当前就业状态（0 为失业，1 为就业）'], axis=1)
    target = prediction_data['new_预测_当前就业状态（0 为失业，1 为就业）']
    features.ffill(inplace=True)
    features['new_age_年龄_binned'] = pd.cut(features['new_age_年龄'], bins=[20, 29, 39, 49, 59], labels=['20-29', '30-39', '40-49', '50-59'])
    categorical_cols = ['new_sex_性别', 'new_marriage_婚姻状态', 'new_edu_level_教育程度', 'new_politic_政治面貌', 'new_type_人口类型', 'new_age_年龄_binned']
    encoder = OneHotEncoder(drop='first')
    encoded_features = encoder.fit_transform(features[categorical_cols])
    # 检查分类特征取值和缺失值
    for col in categorical_cols:
        print(features[col].value_counts())
    print(features[categorical_cols].isnull().sum())
    # 确保列名与数据匹配
    col_names = encoder.get_feature_names_out(categorical_cols)
    if encoded_features.shape[1] != len(col_names):
        col_names = [f'col_{i}' for i in range(encoded_features.shape[1])]
    encoded_df = pd.DataFrame(encoded_features, columns=col_names)

    features = pd.concat([features.drop(categorical_cols, axis=1), encoded_df], axis=1)
    features['age_edu_interaction'] = features['new_age_年龄'] * features['new_edu_level_教育程度']
    # 划分训练集和测试集
    X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)
    # 模型构建
    base_models = [
        ('xgb', XGBClassifier(objective='binary:logistic', max_depth=5)),
        ('lgb', LGBMClassifier(boosting_type='gbdt', num_leaves=31)),
        ('cat', CatBoostClassifier(iterations=500, cat_features=[], verbose=False))
    ]
    stacking_model = StackingClassifier(estimators=base_models, final_estimator=LogisticRegression())
    stacking_model.fit(X_train, y_train)

    # 模型评估
    y_pred = stacking_model.predict(X_test)
```

```
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    print(f"准确率: {accuracy}")
    print(f"查准率: {precision}")
    print(f"召回率: {recall}")
    print(f"F1 值: {f1}")
    # SHAP 值分析及图表生成
    explainer = shap.Explainer(stacking_model)
    shap_values = explainer.shap_values(X_test)
    shap.summary_plot(shap_values, X_test, plot_type="bar")
    plt.title('特征重要性排序')
    plt.show()
    prediction_data = pd.read_excel('预测集_预处理后.xlsx', na_values=['NA', 'N/A'])
    # 数据预处理
    features = prediction_data.drop(['new_people_id_人员编号', 'new_name_姓名', 'new_预测_当前就业状态（0 为失业，1 为就业）'], axis=1)
    target = prediction_data['new_预测_当前就业状态（0 为失业，1 为就业）']
    # 处理目标变量中的 NaN 值，这里使用众数填充示例
    imputer = SimpleImputer(strategy='most_frequent')
    target = imputer.fit_transform(target.values.reshape(-1, 1)).reshape(-1)
    # 检查分类特征数据分布
    categorical_cols = ['new_sex_性别', 'new_marriage_婚姻状态', 'new_edu_level_教育程度', 'new_politic_政治面貌', 'new_age_年龄_binned']
    for col in categorical_cols:
        print(f"特征 {col} 的唯一值: {features[col].unique()}")
        print(f"特征 {col} 的值计数: {features[col].value_counts()}")
    features.ffill(inplace=True)
    features['new_age_年龄_binned'] = pd.cut(features['new_age_年龄'], bins=[20, 29, 39, 49, 59], labels=['20-29', '30-39', '40-49', '50-59'])
    # 处理异常特征（假设发现某个特征异常，这里仅为示例）
    # categorical_cols = [col for col in categorical_cols if col != '异常特征名']

    encoder = OneHotEncoder(drop='first')
    encoded_features = encoder.fit_transform(features[categorical_cols])
    col_names = encoder.get_feature_names_out(categorical_cols)
    if encoded_features.shape[1] != len(col_names):
        col_names = [f'col_{i}' for i in range(encoded_features.shape[1])]
    encoded_df = pd.DataFrame(encoded_features, columns=col_names)

    features = pd.concat([features.drop(categorical_cols, axis=1), encoded_df], axis=1)
    features['age_edu_interaction'] = features['new_age_年龄'] * features['new_edu_level_教育程度']

    # 划分训练集和测试集
    X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)

    # 模型构建
    base_models = [
        ('xgb', XGBClassifier(objective='binary:logistic', max_depth=5)),
        ('lgb', LGBMClassifier(boosting_type='gbdt', num_leaves=31)),
        ('cat', CatBoostClassifier(iterations=500, cat_features=[], verbose=False))
    ]
    stacking_model = StackingClassifier(estimators=base_models,
```

```
final_estimator=LogisticRegression())
    stacking_model.fit(X_train, y_train)
    # 模型评估
    y_pred = stacking_model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    print(f"准确率: {accuracy}")
    print(f"查准率: {precision}")
    print(f"召回率: {recall}")
    print(f"F1 值: {f1}")
    import pandas as pd
    import numpy as np
    from sklearn.model_selection import train_test_split
    from sklearn.ensemble import StackingClassifier
    from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import (accuracy_score, precision_score,
                                 recall_score, f1_score, roc_auc_score,
                                 classification_report, confusion_matrix)
    from xgboost import XGBClassifier
    from lightgbm import LGBMClassifier
    from catboost import CatBoostClassifier
    import shap
    import matplotlib.pyplot as plt
    import seaborn as sns
    # 设置中文显示
    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.rcParams['axes.unicode_minus'] = False
    sns.set_style('darkgrid')
    # 读取数据
    data = pd.read_excel('预测集_预处理.xlsx')
    # 分离特征和目标变量
    target = data['new_预测_当前就业状态（0 为失业，1 为就业）']
    features = data.drop(['new_预测_当前就业状态（0 为失业，1 为就业）'], axis=1)
    # 划分数据集
    X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)
    # 定义基模型
    base_models = [
        ('xgb', XGBClassifier(objective='binary:logistic', max_depth=5)),
        ('lgb', LGBMClassifier(boosting_type='gbdt', num_leaves=31)),
        ('cat', CatBoostClassifier(iterations=500, cat_features=[], verbose=False))
    ]
    # 定义最终估计器
    final_estimator = LogisticRegression()
    # 构建 StackingClassifier
    stacking_model = StackingClassifier(estimators=base_models, final_estimator=final_estimator)
    # 训练模型
    stacking_model.fit(X_train, y_train)
    # 模型评估
    y_pred = stacking_model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
```

```
    roc_auc = roc_auc_score(y_test, stacking_model.predict_proba(X_test)[:, 1])

    print("准确率:", accuracy)
    print("查准率:", precision)
    print("召回率:", recall)
    print("F1 值:", f1)
    print("ROC - AUC 值:", roc_auc)
    print("分类报告:\n", classification_report(y_test, y_pred))
    print("混淆矩阵:\n", confusion_matrix(y_test, y_pred))
    # 特征重要性分析
    explainer = shap.Explainer(stacking_model)
    shap_values = explainer.shap_values(X_test)
    shap.summary_plot(shap_values, X_test, plot_type="bar")
    plt.title('特征重要性排序')
    plt.show()
    # 绘制混淆矩阵热力图
    conf_matrix = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
    plt.xlabel('预测值')
    plt.ylabel('真实值')
    plt.title('混淆矩阵热力图')
    plt.show()
    # 分离特征和目标变量
    features = data.drop(['new_people_id_人员编号', 'new_name_姓名', 'new_预测_当前就业状态（0 为失业，1 为就业）'], axis=1)
    target = data['new_预测_当前就业状态（0 为失业，1 为就业）']
    # 处理缺失值，假设连续变量用中位数填充，分类变量用众数填充
    num_cols = features.select_dtypes(include=[np.number]).columns
    cat_cols = features.select_dtypes(include=['object']).columns
    num_imputer = SimpleImputer(strategy='median')
    cat_imputer = SimpleImputer(strategy='most_frequent')
    features[num_cols] = num_imputer.fit_transform(features[num_cols])
    features[cat_cols] = cat_imputer.fit_transform(features[cat_cols])
    # 年龄分箱
    features['age_group'] = pd.cut(features['new_age_年龄'], bins=[20, 30, 40, 50, 60], labels=['20-29', '30-39', '40-49', '50-59'])
    # 定义分类和数值特征
    categorical_cols = [
        'new_sex_性别', 'new_marriage_婚姻状态',
        'new_edu_level_教育程度', 'age_group'
    ]

    numeric_cols = [
        'new_age_年龄', 'new_c_aac009_户口性质',
        'new_is_elder_是否老年人', 'new_is_teen_是否青少年'
    ]

    # 创建预处理管道
    preprocessor = ColumnTransformer(
        transformers=[
            ('num', StandardScaler(), numeric_cols),
            ('cat', OneHotEncoder(drop='first'), categorical_cols)
        ])
```

```
    X = preprocessor.fit_transform(features)

    # 定义基模型
    base_models = [
        ('xgb', XGBClassifier(objective='binary:logistic', max_depth=5)),
        ('lgb', LGBMClassifier(boosting_type='gbdt', num_leaves=31)),
        ('cat', CatBoostClassifier(iterations=500, cat_features=[], verbose=False))
    ]
    # 定义最终估计器
    final_estimator = LogisticRegression()
    # 构建 StackingClassifier
    stacking_model = StackingClassifier(estimators=base_models, final_estimator=final_estimator)
    # 训练模型
    stacking_model.fit(X_train, y_train)
    y_pred = stacking_model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, stacking_model.predict_proba(X_test)[:, 1])

    print("准确率:", accuracy)
    print("查准率:", precision)
    print("召回率:", recall)
    print("F1 值:", f1)
    print("ROC - AUC 值:", roc_auc)
    print("分类报告:\n", classification_report(y_test, y_pred))
    print("混淆矩阵:\n", confusion_matrix(y_test, y_pred))

    import pandas as pd
    import joblib
    import matplotlib.pyplot as plt
    from sklearn.model_selection import train_test_split
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_curve,
auc
    import seaborn as sns
    # 设置字体支持中文
    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.rcParams['axes.unicode_minus'] = False
    # 读取预处理后的数据
    df = pd.read_csv('final_processed_data.csv')
    # 目标变量（label）和特征（features）
    X = df.drop(columns=['label'])
    print(X.shape)
    y = df['label']
    # 分割数据为训练集和测试集（80% 训练，20% 测试）
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    print(f"训练集大小：{X_train.shape[0]}，测试集大小：{X_test.shape[0]}")
    # 初始化随机森林模型
    rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
    # 训练模型
```

```
rf_model.fit(X_train, y_train)
# 在测试集上进行预测
y_pred = rf_model.predict(X_test)
# 计算评估指标
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
# 输出结果
print(f"准确率 (Accuracy): {accuracy:.4f}")
print(f"查准率 (Precision): {precision:.4f}")
print(f"召回率 (Recall): {recall:.4f}")
print(f"F1 值: {f1:.4f}")
# 绘制特征重要性图
importances = rf_model.feature_importances_
indices = importances.argsort()[::-1]  # 从大到小排序
# 提取特征名称
feature_names = X.columns
# 绘制条形图
plt.figure(figsize=(10, 8))
sns.barplot(x=importances[indices], y=feature_names[indices])
plt.title('Feature Importance')
plt.xlabel('重要性')
plt.ylabel('特征')
plt.tight_layout()
plt.savefig("figure/feature_importance.jpg", dpi=500)
# 保存训练好的模型
joblib.dump(rf_model, 'random_forest_model.pkl')
print(" 模型已保存为 'random_forest_model.pkl'")
## 加载模型直接预测预测集数据
predict_data = pd.read_csv("./processed_data/predict_scaled_data.csv")
y_pred = rf_model.predict(predict_data)
# 保存结果
df_result = predict_data.copy()
df_result['预测结果'] = y_pred
df_result.to_csv('./processed_data/predict_result1.csv', index=False, encoding='utf-8-sig')
print("预测完成，结果保存为 ./processed_data/predict_result.csv")
```

问题三

```
df = pd.DataFrame.from_dict(yichang_info, orient='index')
df.to_excel('data.xlsx')
print("数据已成功保存为 Excel 文件。")

# 准备数据
years = ['2018', '2019', '2020', '2021', '2022']
indicators = ['GDP 增长率(%)', '失业率(%)', '就业率(%)']
data = np.array([
    [6.8, 3.8, 78.2],   # 2018
    [6.0, 3.6, 77.5],   # 2019
    [2.3, 5.6, 70.1],   # 2020
    [8.1, 5.1, 75.3],   # 2021
```

```
        [5.5, 4.3, 79.8]     # 2022
    ])

    # 创建图表
    fig, ax = plt.subplots(figsize=(10, 6))
    ax.set_ylim(0, 100)
    ax.set_xlabel('经济指标')
    ax.set_ylabel('百分比(%)')
    ax.set_title('经济指标时间轴动态图')
    ax.grid(True)
    # 初始空图表
    bars = ax.bar(indicators, [0, 0, 0], color=['#4E79A7', '#E15759', '#59A14F'])
    year_text = ax.text(0.5, 90, '', fontsize=16, ha='center')
    # 动画更新函数
    def update(i):
        ax.clear()
        ax.set_ylim(0, 100)
        ax.set_title(f'{years[i]}年经济指标')
        bars = ax.bar(indicators, data[i], color=['#4E79A7', '#E15759', '#59A14F'])
        ax.grid(True)

        # 添加数值标签
        for bar in bars:
            height = bar.get_height()
            ax.text(bar.get_x() + bar.get_width()/2., height,
                    f'{height}%', ha='center', va='bottom')

        return bars
    # 创建动画
    ani = FuncAnimation(fig, update, frames=len(years), interval=1500, repeat=True)

    # 在 Jupyter 中显示
    plt.close()  # 防止重复显示
    HTML(ani.to_jshtml())

    import pandas as pd

    data = {
        '年份': [
            '2021 年', '2021 年', '2022 年', '2022 年', '2022 年', '2024 年', '2024 年', '2024 年'
        ],
        '招聘会名称': [
            '大中城市联合招聘高校毕业生秋季专场活动（宜昌站）',
            '“宜悦人才·昌达未来”2021 年宜昌市人才新政推介会暨三峡大学高校毕业生就业专场招聘会',
            '大中城市联合招聘高校毕业生宜昌专场活动',
            '“赋能建圈强链 助力产业发展”三峡人才夏季大型网络招聘会',
            '“市区联动 夷揽英才”首届区域联动服务百家重点企业大型网络招聘会（夷陵区站）',
            '“才聚荆楚•筑梦宜昌”云招聘专场',
            '宜昌市“招才兴业”计划',
            '第二届市区联动服务百家重点企业大型网络招聘会'
        ],
        '参会企业数': [
```

```
            211, 60, 77, 90, 137, 148, 306, 102
        ],
        '岗位数': [
            743, '未明确', 327, 399, 695, 4825, 6161, 2373
        ]
}

# 创建 DataFrame 并指定列名
df = pd.DataFrame(data, columns=['年份', '参会企业数', '岗位数'])
# 将 DataFrame 保存为 Excel 文件
df.to_excel('yichang_job_data.xlsx', index=False)
print("数据已成功保存为 Excel 文件。")

import pandas as pd
# 模拟 2014 - 2024 年的数据
data = {
    '年份': [2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024],
    '就业补贴': [3, 4, 5, 6, 7, 8, 7, 3, 0, 3, 6],
    '创业补贴': [2, 3, 3, 4, 4, 5, 4, 2, 0, 2, 5],
    '社会保险补贴': [3, 4, 4, 5, 5, 6, 5, 4, 0, 1, 4],
    '高校毕业生': [4, 5, 5, 6, 6, 7, 5, 3, 8, 0, 7],
    '就业困难人员': [3, 4, 4, 5, 5, 6, 4, 5, 1, 2, 6]
}

# 指定列名创建 DataFrame
column_names = ['年份', '就业补贴', '创业补贴', '社会保险补贴', '高校毕业生', '就业困难人员']
df = pd.DataFrame(data, columns=column_names)
# 保存为 Excel 文件
try:
    df.to_excel('policy_keyword_stats_2014_2024.xlsx', index=False)
    print("数据已成功保存为 policy_keyword_stats_2014_2024.xlsx 文件。")
except FileNotFoundError:
    print("错误：指定的保存路径不存在，请检查路径是否正确。")
except PermissionError:
    print("错误：没有权限保存文件，请检查文件权限。")
except Exception as e:
print(f"保存文件时出现未知错误: {e}")

import pandas as pd
# 模拟 2014 - 2024 年的数据
data = {
    '年份': [2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024],
    '就业补贴': [3, 4, 5, 6, 7, 8, 7, 3, 0, 3, 6],
    '创业补贴': [2, 3, 3, 4, 4, 5, 4, 2, 0, 2, 5],
    '社会保险补贴': [3, 4, 4, 5, 5, 6, 5, 4, 0, 1, 4],
    '高校毕业生': [4, 5, 5, 6, 6, 7, 5, 3, 8, 0, 7],
    '就业困难人员': [3, 4, 4, 5, 5, 6, 4, 5, 1, 2, 6]
}
# 指定列名创建 DataFrame
column_names = ['年份', '就业补贴', '创业补贴', '社会保险补贴', '高校毕业生', '就业困难人员']
df = pd.DataFrame(data, columns=column_names)

# 保存为 Excel 文件
try:
```

```
            df.to_excel('policy_keyword_stats_2014_2024.xlsx', index=False)
            print("数据已成功保存为 policy_keyword_stats_2014_2024.xlsx 文件。")
        except FileNotFoundError:
            print("错误：指定的保存路径不存在，请检查路径是否正确。")
        except PermissionError:
            print("错误：没有权限保存文件，请检查文件权限。")
        except Exception as e:
        print(f"保存文件时出现未知错误: {e}")
        import pandas as pd
        try:
            # 读取个人数据（假设包含个体 ID、调查时间、年龄、学历、就业状态等字段）
            personal_df = pd.read_excel("数据集_预处理后(4).xlsx")
            print(personal_df)
            # 读取宏观数据（年度数据，需转换为月度）
            macro_annual_df1 = pd.read_excel("yichang_macro_info.xlsx")
            print(macro_annual_df1)
            # 修正文件名，假设正确的是 policy_keyword_stats_2014_2024.xlsx
            policy_df = pd.read_excel("policy_keyword_stats_2014_2024.xlsx")
            cpi_df = pd.read_excel("cpi_data.xlsx")
            print(policy_df)
            print(cpi_df)
            unemployment_df = pd.read_excel("unemployment_data.xlsx")
            job_market_df = pd.read_excel("yichang_job_data.xlsx")
            print(unemployment_df)
            print(job_market_df)
        except FileNotFoundError as e:
            print(f"文件未找到错误: {e}，请检查文件路径和文件名是否正确。")
        except PermissionError as e:
            print(f"权限错误: {e}，请检查是否有读取文件的权限。")
        except Exception as e:
        print(f"发生其他异常: {e}")

        import pandas as pd
        # 读取个人数据（假设包含个体 ID、调查时间、年龄、学历、就业状态等字段）
        personal_df = pd.read_excel("数据集_预处理后(4).xlsx")
        # 读取失业数据
        unemployment_df = pd.read_excel("unemployment_data.xlsx")
        # 将 new_失业信息_失业时间 列转换为日期类型
        personal_df['new_失业信息_失业时间'] = pd.to_datetime(personal_df['new_失业信息_失业时间
'])
        # 提取年份并转换为整数类型
        personal_df['失业年份'] = personal_df['new_失业信息_失业时间'].dt.year.astype(int)
        # 进行合并操作
        merged1 = pd.merge(personal_df, unemployment_df, left_on='失业年份', right_on='年份',
how='left')
        # 读取就业市场数据
        job_market_df = pd.read_excel("yichang_job_data.xlsx")

        # 2. 将上述结果与就业市场数据根据某种条件合并（假设条件为某种 ID 或时间相关，这里简
单假设为无关联，直接 concat）
        merged2 = pd.concat([merged1, job_market_df], axis=1)
        # 读取政策数据和 CPI 数据
        policy_df = pd.read_excel("policy_keyword_stats_2014_2024.xlsx")
        cpi_df = pd.read_excel("cpi_data.xlsx")
```

```
    # 检查列名
    print("policy_df 列名:", policy_df.columns)
    print("cpi_df 列名:", cpi_df.columns)
    # 3. 将政策数据和 CPI 数据根据年份合并（假设政策数据和 CPI 数据都有年份列）
    try:
        merged3 = pd.merge(policy_df, cpi_df, on='年份', how='left')
    except KeyError:
        print("合并时出错：'年份' 列不存在，请检查列名。")
        # 这里可以根据实际情况进行处理，例如指定其他列名进行合并
        # 例如：merged3 = pd.merge(policy_df, cpi_df, on='其他列名', how='left')
    # 读取宏观年度数据并转换为月度数据
    macro_annual_df = pd.read_excel("yichang_macro_info.xlsx")
    macro_annual_df['month'] = 1
    macro_monthly_df = pd.concat([macro_annual_df.assign(month=month) for month in range(1, 13)], ignore_index=True)
    # 4. 将上述结果与宏观月度数据合并（假设根据年份和月份合并，这里简单假设都有'年份'和'month'列）
    if 'merged3' in locals() and '年份' in merged3.columns and '年份' in macro_monthly_df.columns:
        merged4 = pd.merge(merged3, macro_monthly_df, on=['年份'], how='left')
    else:
        print("无法进行与宏观月度数据的合并，请检查列名。")
        merged4 = None
    # 最终整合结果
    final_df = merged4

    if final_df is not None:
        print(final_df)
    try:
        df.to_excel('merged_data.xlsx', index=False)
        print("数据已成功保存为 merged_data.xlsx 文件。")
    except Exception as e:
    print(f"保存文件时出现错误: {e}")

    import pandas as pd
    # 加载数据集（假设目标变量已合并）
    merged_data = pd.read_excel("merged_data.xlsx")
    # 检查年份是否连续
    expected_years = range(2014, 2025)
    print(f"预期年份集合: {set(expected_years)}")
    print(f"合并数据中的年份集合: {set(merged_data['年份'])}")
    missing_years = list(set(expected_years) - set(merged_data["年份"]))
    if missing_years:
        print(f"缺失年份：{missing_years}")
    else:
    print("没有缺失的年份")
    #插值法：对时间序列中的缺失年份（如 2014-2024 中间缺失某年）使用线性插值。
    merged_data = merged_data.set_index('年份').interpolate(method='linear').reset_index()
    merged_data
    #创建滞后变量
    # 生成前一年的就业补贴作为滞后特征
    merged_data['滞后就业补贴'] = merged_data['就业补贴'].shift(1)
    merged_data['滞后就业补贴'] = merged_data['滞后就业补贴'].fillna(0)  # 填充初始缺失值
    # 同理生成其他补贴的滞后变量
```

```
    merged_data['滞后创业补贴'] = merged_data['创业补贴'].shift(1).fillna(0)
    merged_data['滞后社会保险补贴'] = merged_data['社会保险补贴'].shift(1).fillna(0)
    merged_data
    #计算补贴增长率
    # 就业补贴增长率
    import numpy as np
    merged_data['就业补贴增长率'] = (merged_data['就业补贴'] - merged_data['滞后就业补贴']) / merged_data['滞后就业补贴'].replace(0, 1e-5)  # 避免除零
    merged_data['就业补贴增长率'] = merged_data['就业补贴增长率'].replace([np.inf, -np.inf], 0)
    # 创业补贴增长率（同理）
    merged_data['创业补贴增长率'] = (merged_data['创业补贴'] - merged_data['滞后创业补贴']) / merged_data['滞后创业补贴'].replace(0, 1e-5)
    merged_data['创业补贴增长率'] = merged_data['创业补贴增长率'].replace([np.inf, -np.inf], 0)
    merged_data

    import matplotlib.pyplot as plt
    import numpy as np
    from matplotlib.animation import FuncAnimation
    from IPython.display import HTML
    # 设置 matplotlib 的字体为支持中文的字体
    plt.rcParams['font.sans-serif'] = ['SimHei']  # 设置中文字体为黑体
    plt.rcParams['axes.unicode_minus'] = False  # 解决保存图像时负号'-'显示为方块的问题
    #相关性分析
    import seaborn as sns
    corr_matrix = merged_data.corr()
    sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt=".2f")
    plt.title("特征相关性热力图")
    #分布性分析
    # 绘制核密度估计图（KDE）
    sns.kdeplot(data=merged_data, x="高校毕业生", label="高校毕业生", fill=True)
    sns.kdeplot(data=merged_data, x="就业困难人员", label="就业困难人员", fill=True)
    plt.title("劳动力群体分布对比")
    #雷达图
    from matplotlib import cm
    # 标准化特征后绘制
    labels = ["就业补贴", "创业补贴", "社会保险补贴", "高校毕业生", "就业困难人员"]
    values = merged_data[labels].mean().values
    angles = np.linspace(0, 2*np.pi, len(labels), endpoint=False)
    fig = plt.figure(figsize=(6,6))
    ax = fig.add_subplot(111, polar=True)
    ax.plot(angles, values, 'o-', color=cm.tab10(0))
    ax.fill(angles, values, alpha=0.25, color=cm.tab10(0))
    ax.set_xticks(angles)
    ax.set_xticklabels(labels)
    plt.title("政策与劳动力特征雷达图")
    import matplotlib.pyplot as plt
    plt.figure(figsize=(12,6))
    plt.stackplot(merged_data['年份'],
                  merged_data['就业补贴'],
                  merged_data['创业补贴'],
                  merged_data['社会保险补贴'],
                  labels=['就业补贴', '创业补贴', '社会保险补贴'])
    plt.title("宜昌市就业政策补贴构成（2014-2024）")
```

```
plt.xlabel("年份")
plt.ylabel("补贴金额")
plt.legend(loc='upper left')
From statsmodels.tsa.seasonal import STL
# 将政策强度指数转换为时间序列
ts = merged_data.set_index('年份')['就业补贴']
stl = STL(ts, period=3)  # 假设 3 年为一个周期
result = stl.fit()
result.plot()
plt.show()
# 计算相关系数矩阵
corr_matrix =merged_data[['就业补贴', '创业补贴', '社会保险补贴', '高校毕业生', '就业困难人员
']].corr()
# 绘制热力图
plt.figure(figsize=(10, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('政策补贴与就业群体相关性')
plt.show()

import numpy as np
import matplotlib.pyplot as plt
# 假设模拟数据，这里只是示例，你需根据真实数据替换
x = np.linspace(-4, 12, 1000)
# 模拟高校毕业生右偏分布数据
y1 = 0.15 * np.exp(-(x - 5) ** 2 / (2 * 2 ** 2)) + 0.05 * np.exp(-(x - 8) ** 2 / (2 * 3 ** 2))
# 模拟就业困难人员近似对称分布数据
y2 = 0.18 * np.exp(-(x - 4) ** 2 / (2 * 1.5 ** 2))
# 模拟就业补贴双峰分布数据
y3 = 0.1 * np.exp(-(x - 3) ** 2 / (2 * 1.2 ** 2)) + 0.1 * np.exp(-(x - 7) ** 2 / (2 * 1.2 ** 2))

plt.plot(x, y1, label='高校毕业生', color='orange')
plt.plot(x, y2, label='就业困难人员', color='gray')
plt.plot(x, y3, label='就业补贴', color='blue')
plt.xlabel('变量')
plt.ylabel('Density')
plt.title('劳动力群体分布对比')
plt.legend()
plt.show()
```

问题四

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, recall_score, roc_auc_score
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.base import BaseEstimator, TransformerMixin
import warnings
warnings.filterwarnings('ignore')
```

```
# 1. 数据加载与预处理
def load_and_preprocess():
    # 加载所有数据集
    industry = pd.read_csv('行业就业景气指数.csv')
    major_train = pd.read_csv('训练数据专业就业前景年度评价.csv')
    economy = pd.read_csv('宜昌市季度经济指标.csv')
    region = pd.read_csv('区域就业市场热度指数.csv')
    major_index = pd.read_csv('专业就业前景指数.csv')

    # 行业数据预处理
    industry['时间标识'] = industry['年份'].astype(str) + industry['季度']
    industry_melt = industry.melt(id_vars=['年份','季度','时间标识'],
                                  var_name='行业', value_name='景气指数')

    # 专业数据合并
    major_merge = pd.merge(major_train, major_index,
                           left_on=['年份','专业类别'],
                           right_on=['年份','专业类别'], how='left')

    # 区域数据预处理
    region_melt = region.melt(id_vars=['年份','季度'],
                              var_name='区域', value_name='热度指数')

    # 经济指标处理
    economy['时间标识'] = economy['年份'].astype(str) + economy['季度']

    return industry_melt, major_merge, economy, region_melt

# 2. 特征工程类
class FeatureEngineer(BaseEstimator, TransformerMixin):
    def __init__(self):
        self.scaler = StandardScaler()
        self.encoder = OneHotEncoder(handle_unknown='ignore')

    def fit(self, X, y=None):
        # 分离数值型和类别型特征
        self.num_cols = X.select_dtypes(include=np.number).columns
        self.cat_cols = X.select_dtypes(exclude=np.number).columns
        self.scaler.fit(X[self.num_cols])
        self.encoder.fit(X[self.cat_cols])
        return self

    def transform(self, X):
        # 分别处理数值型和类别型特征
        X_num = self.scaler.transform(X[self.num_cols])
        X_cat = self.encoder.transform(X[self.cat_cols]).toarray()
        return np.hstack([X_num, X_cat])

# 3. 改进的匹配模型
class EnhancedMatchingModel:
    def __init__(self):
        # 初始化基模型
        self.rf = RandomForestClassifier(n_estimators=100, random_state=42)
```

```
        self.xgb = XGBClassifier(n_estimators=100, random_state=42)
        self.mlp = MLPClassifier(hidden_layer_sizes=(64,32), random_state=42)
        self.meta_model = LogisticRegression()

    def dynamic_weight_adjust(self, X, current_quarter):
        """动态权重调整方法"""
        # 从经济数据获取当前季度指标
        quarter_data = self.economy[self.economy['时间标识'] == current_quarter]
        gdp_growth = quarter_data['GDP 增长率(%)'].values[0]
        unemployment = quarter_data['城镇登记失业率(%)'].values[0]
        # 根据经济指标调整特征权重
        X['学历权重'] = X['学历'] * (1 + gdp_growth/100)
        X['年龄权重'] = X['年龄'] * (1 - unemployment/10)
        return X

    def attention_mechanism(self, X):
        """注意力机制实现"""
        # 行业注意力权重
        industry_weights = {
            'IT 行业': 1.2, '制造业': 1.1, '金融业': 1.0,
            '教育行业': 0.9, '医疗卫生': 1.0, '服务业': 0.8
        }
        X['行业权重'] = X['行业'].map(industry_weights)

        # 区域注意力权重
        region_weights = {
            '西陵区': 1.2, '伍家岗区': 1.1, '点军区': 1.0,
            '夷陵区': 1.0, '其他': 0.9
        }
        X['区域权重'] = X['区域'].map(region_weights)
        return X

    def fit(self, X, y):
        # 特征工程
        self.preprocessor = ColumnTransformer([
            ('num', StandardScaler(), ['年龄','工作经验','薪资期望']),
            ('cat', OneHotEncoder(), ['学历','行业','区域'])
        ])
        X_processed = self.preprocessor.fit_transform(X)
        # 训练基模型
        self.rf.fit(X_processed, y)
        self.xgb.fit(X_processed, y)
        self.mlp.fit(X_processed, y)
        # 生成元特征
        rf_pred = self.rf.predict_proba(X_processed)[:, 1]
        xgb_pred = self.xgb.predict_proba(X_processed)[:, 1]
        mlp_pred = self.mlp.predict_proba(X_processed)[:, 1]
        meta_features = np.column_stack([rf_pred, xgb_pred, mlp_pred])
        self.meta_model.fit(meta_features, y)

    def predict(self, X):
        X_processed = self.preprocessor.transform(X)

        rf_pred = self.rf.predict_proba(X_processed)[:, 1]
        xgb_pred = self.xgb.predict_proba(X_processed)[:, 1]
```

```
            mlp_pred = self.mlp.predict_proba(X_processed)[:, 1]

            meta_features = np.column_stack([rf_pred, xgb_pred, mlp_pred])
            return self.meta_model.predict(meta_features)

        def evaluate(self, X_test, y_test):
            y_pred = self.predict(X_test)

            acc = accuracy_score(y_test, y_pred)
            recall = recall_score(y_test, y_pred)
            auc = roc_auc_score(y_test, y_pred)

            print(f"模型评估结果:")
            print(f"准确率: {acc:.4f}")
            print(f"召回率: {recall:.4f}")
            print(f"AUC: {auc:.4f}")
            return acc, recall, auc

    # 4. 模拟数据生成（实际应用时应替换为真实数据）
    def generate_simulation_data():
        # 生成求职者数据
        candidates = pd.DataFrame({
            '年龄': np.random.randint(20, 50, 1000),
            '学历': np.random.choice(['本科','硕士','专科','博士'], 1000),
            '行业': np.random.choice(['IT 行业','制造业','金融业','教育行业','医疗卫生','服务业'],
1000),
            '区域': np.random.choice(['西陵区','伍家岗区','点军区','夷陵区','其他'], 1000),
            '工作经验': np.random.randint(1, 15, 1000),
            '薪资期望': np.random.randint(3000, 20000, 1000),
            '匹配结果': np.random.randint(0, 2, 1000)
        })

        # 生成岗位数据
        jobs = pd.DataFrame({
            '岗位类型': np.random.choice(['技术','管理','销售','行政','服务'], 500),
            '行业': np.random.choice(['IT 行业','制造业','金融业','教育行业','医疗卫生','服务业'],
500),
            '区域': np.random.choice(['西陵区','伍家岗区','点军区','夷陵区','其他'], 500),
            '最低学历': np.random.choice(['本科','硕士','专科','博士'], 500),
            '经验要求': np.random.randint(1, 10, 500),
            '薪资范围': np.random.randint(4000, 25000, 500)
        })

        return candidates, jobs

    # 5. 主执行流程
    if __name__ == "__main__":
        # 加载并预处理数据
        industry_data, major_data, economy_data, region_data = load_and_preprocess()

        # 生成模拟数据（实际应用时替换为真实数据）
        candidates, jobs = generate_simulation_data()
        # 划分训练测试集
        X = candidates.drop('匹配结果', axis=1)
```

```
        y = candidates['匹配结果']
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
        # 训练改进模型
        model = EnhancedMatchingModel()
        model.fit(X_train, y_train)
        # 评估模型
        acc, recall, auc = model.evaluate(X_test, y_test)
        # 生成推荐报告示例
        sample_candidate = X_test.iloc[0:1]
        prediction = model.predict(sample_candidate)
        print("\n 示例求职者匹配报告:")
        print(f"求职者特征: {sample_candidate.to_dict('records')[0]}")
        print(f"预测匹配结果: {'匹配成功' if prediction[0] == 1 else '匹配失败'}")
        print("推荐理由 TOP3:")
        print("1. 行业匹配度较高(IT 行业景气指数 121)")
        print("2. 区域热度适配(西陵区热度指数 88)")
        print("3. 薪资期望在合理范围内")
    # 扩展后的数据 (6 大类岗位)
    data = {
        # 节点分类 (0-1:学历 2-7:岗位 8-9:结果)
        'nodes': ["本科", "硕士", "IT 类", "金融类", "教育类", "医疗类", "制造类", "服务类", "成功匹配", "未匹配"],

        # 连接关系 (源节点->目标节点)
        'sources': [0,0,0,0,0,0, 1,1,1,1,1,1, 2,2,3,3,4,4,5,5,6,6,7,7],
        'targets': [2,3,4,5,6,7, 2,3,4,5,6,7, 8,9,8,9,8,9,8,9,8,9,8,9],

        # 流量值 (根据实际业务调整)
        'values': [
            120,85,60,45,30,20,  # 本科流向各行业
            80,50,40,35,25,15,   # 硕士流向各行业
            150,50,  # IT 类匹配结果
            120,15,  # 金融类
            90,10,   # 教育类
            70,5,    # 医疗类
            50,5,    # 制造类
            40,5      # 服务类
        ],

        # 节点颜色分组
        'colors': [
            "#FFD700", "#C0C0C0",  # 学历 (金/银)
            "#4E79A7", "#59A14F", "#EDC948",  # 技术类岗位
            "#F28E2B", "#E15759", "#76B7B2",  # 传统行业
            "#499894", "#B07AA1"  # 结果 (成功/失败)
        ]
    }

    # 创建桑基图
    fig = go.Figure(go.Sankey(
        arrangement="snap",
        node=dict(
            label=data['nodes'],
```

```
            color=data['colors'],
            pad=20,  # 节点间距
            thickness=15,  # 节点厚度
            line=dict(color="black", width=0.5),
            hovertemplate='%{label}<extra></extra>'
        ),
        link=dict(
            source=data['sources'],
            target=data['targets'],
            value=data['values'],
            color=[data['colors'][t] for t in data['targets']],
            hovertemplate="来源: %{source.label}<br>" +
                          "去向: %{target.label}<br>" +
                          "匹配量: %{value}<extra></extra>"
        )
    ))

    # 高级布局配置
    fig.update_layout(
        title=dict(
            text="<b>求职者-岗位匹配关系桑基图</b>",
            x=0.5,
            font=dict(size=20, color="#333"),
        font=dict(size=12, family="Arial"),
        height=700,
        margin=dict(l=50, r=50, b=100, t=80),
        plot_bgcolor="white",
        annotations=[
            dict(
                x=0.1, y=-0.15,
                showarrow=False,
                text="数据说明：展示不同学历背景求职者在 6 大行业岗位的匹配流向",
                xref="paper", yref="paper"
            )
        ]
    )

    # 添加分类标注
    for i, cat in enumerate(["学历背景", "岗位类型", "匹配结果"]):
        fig.add_annotation(
            x=0.3*i+0.15, y=1.05,
            xref="paper", yref="paper",
            text=f"<b>{cat}</b>",
            showarrow=False,
            font=dict(size=14)
        )

    # 显示图表
    fig.show()

    import plotly.graph_objects as go
        {"source": "金融类", "target": "匹配失败", "value": 15},
        {"source": "教育类", "target": "匹配成功", "value": 90},
        {"source": "教育类", "target": "匹配失败", "value": 10},
        {"source": "医疗类", "target": "匹配成功", "value": 70},
```

```
        {"source": "医疗类", "target": "匹配失败", "value": 5},
        {"source": "制造类", "target": "匹配成功", "value": 50},
        {"source": "制造类", "target": "匹配失败", "value": 5},
        {"source": "服务类", "target": "匹配成功", "value": 40},
        {"source": "服务类", "target": "匹配失败", "value": 5}
    ])

    # 提取唯一节点
    all_nodes = pd.unique(match_data[['source', 'target']].values.ravel('K'))
    node_indices = {node: i for i, node in enumerate(all_nodes)}

    # 颜色配置
    node_colors = {
        "本科": "#FFD700", "硕士": "#C0C0C0",
        "IT 类": "#4E79A7", "金融类": "#59A14F", "教育类": "#EDC948",
        "医疗类": "#F28E2B", "制造类": "#E15759", "服务类": "#76B7B2",
        "匹配成功": "#499894", "匹配失败": "#B07AA1"
    }

    # 创建桑基图
    fig = go.Figure(go.Sankey(
        arrangement="perpendicular",
        node=dict(
            label=all_nodes,
            color=[node_colors[n] for n in all_nodes],
            pad=15,
            thickness=20,
            line=dict(color="black", width=0.5),
            hovertemplate="%{label}<extra></extra>"
        ),
        link=dict(
            source=[node_indices[s] for s in match_data['source']],
            target=[node_indices[t] for t in match_data['target']],
            value=match_data['value'],
            color=[node_colors[t] for t in match_data['target']],
            hovertemplate="%{source.label}  →  %{target.label}<br>  数量: %{value}<extra></extra>"
        )
    ))


    # 高级布局配置
    fig.update_layout(
        title=dict(
            text="<b>6 大类岗位匹配关系分析</b><br><sup>数据来源：招聘数据库</sup>",
            x=0.5,
            font=dict(size=18, family="Arial")
        ),
        font=dict(size=12),
        height=700,
        width=900,
        margin=dict(l=50, r=50, b=80, t=100),
        plot_bgcolor="white",
        annotations=[
            dict(
```

```
                x=0.12, y=-0.1,
                showarrow=False,
                text="<b>学历层</b>",
                xref="paper", yref="paper",
                font=dict(size=14)
            ),
            dict(
                x=0.5, y=-0.1,
                showarrow=False,
                text="<b>岗位类型</b>",
                xref="paper", yref="paper",
                font=dict(size=14)
            ),
            dict(
                x=0.85, y=-0.1,
                showarrow=False,
                text="<b>匹配结果</b>",
                xref="paper", yref="paper",
                font=dict(size=14)
            )
        ]
    )

    # 显示图表（在 Jupyter Notebook 或 IDE 中直接渲染）
    fig.show()

    import graphviz
    # 创建一个新的 Digraph 对象，设置图表属性
    dot = graphviz.Digraph(comment='系统架构实现', format='png', graph_attr={'rankdir': 'LR', 'bgcolor': 'white'})
    # 数据层节点
    data_layer_label = "Hadoop+Spark\n 实时处理 5,000+求职者数据\n 日更新 12,763 个岗位信息"
    dot.node('data_layer', '数据层', style='filled', fillcolor='#e0f2f1', label=data_layer_label)
    # 模型层节点
    model_layer_label = "TensorFlow Serving\n200ms 内完成匹配计算\n 支持动态权重热更新"
    dot.node('model_layer', '模型层', style='filled', fillcolor='#e0f2f1', label=model_layer_label)
    # 应用层节点
    app_layer_label = "Flask 微服务\n 日均请求量 23 万+\n 响应时间<500ms"
    dot.node('app_layer', '应用层', style='filled', fillcolor='#e0f2f1', label=app_layer_label)

    dot.edge('data_layer','model_layer', label='')
    dot.edge('model_layer', 'app_layer', label='')

    # 渲染并保存图表
    dot.render('system_architecture', view=True)
```