

2024 年第三届“钉钉杯”大学生 大数据挑战赛论文

题 目 “数据如烟，千帆竞发”：烟草销量与金额预测中的应用

摘 要

本研究旨在通过分析某地区多种烟草品牌的历史销售数据，构建并评估多种机器学习模型，以预测未来的销量和销售金额。在数据处理阶段，附件中的销量和销售金额数据经过数据清洗、特征工程以及标准化处理，从而确保数据的质量和模型的输入准确性。

基于预处理后和精细的特征工程的数据，针对问题中的烟草销量和金额预测，先实施了四种不同的时间序列预测方法，包括 SARIMA、LSTM、XGBoost、决策树以及集成学习模型来进行预测。特别是 SARIMA-LSTM 混合模型，利用 SARIMA 模型处理时间序列的长期趋势，以及 LSTM 模型捕捉短期波动的能力，在预测未来销量方面表现优异。

基于前期的四个不同时间序列方法，为了进一步提高预测的精度和稳健性，我们还构建了 Stacking 集成学习模型，将多种基模型的预测结果进行整合，对 A5 烟草品牌的销量和销售金额进行了联合预测。本研究尝试了多种机器学习和深度学习算法，包括逻辑回归、KNN、随机森林回归、GBDT、XGBoost、AdaBoost、BP 神经网络、CNN、LSTM 等。其中 XGBoost 模型表现最为突出，达到了 R^2 为 0.8528，RMSE 为 1.59016，MAE 为 1.005。集成模型在应对高维和非线性问题时具有显著优势，预测结果在均方根误差（RMSE）、平均绝对误差（MAE）等评估指标上均表现优异。

本研究深入分析了烟草营销案例的数据特征，揭示了其关键属性。然而，研究过程中仍存在一定的局限性，特别是在数据处理和预测分析方面，仍有优化的空间。未来的研究将引入更为先进的技术和方法，以提高分析和预测的准确性。本研究的分析和预测方法，能够为烟草行业的库存管理、供应链优化及市场策略制定提供科学的决策支持，从而提升企业的市场竞争力和运营效率。

关键词：烟草销量预测；机器学习模型；SARIMA-LSTM；集成学习；数据预处理

目录

摘 要 1

第一章 绪论 1

 1.1 问题背景 1

 1.2 文献综述 1

第二章 问题重述与分析 3

 2.1 问题重述 3

 2.2 问题分析 3

第三章 数据预处理与分析 5

 3.1 数据集概述 5

 3.2 数据预处理策略 5

 3.3 特征筛选及优化 7

 3.4 缺失值处理 8

第四章 方法与技术 9

 4.1 对未来销量进行预测 9

 4.2 SARIMA + LSTM 混合模型 10

 4.3 Prophet 模型 11

 4.4 对销售金额进行预测 11

 4.5 XGBoost 12

 4.6 决策树模型 12

 4.7 构建 Stacking 集成模型 13

第五章 结果展示 15

 5.1 未来销量预测结果展示 15

 5.2 集成学习模型结果展示 19

第六章 讨论与深度分析 22

 6.1 深度洞察与模型解读 22

 6.2 研究限制与潜在改进 22

第七章 结论与未来研究方向 23

 7.1 研究成果与贡献 23

 7.2 未来研究方向展望 23

参考文献 24

附录 26

第一章 绪论

1.1 问题背景

中国的烟草行业是一个庞大的经济体，在国家财政收入中占据重要地位。中国的烟草产业链由烟叶种植与购销、烟标制造、烟叶加工与卷烟制造、卷烟批发和卷烟零售五个主要环节构成。在烟草行业中，准确的销售预测对优化库存管理和供应链计划至关重要。

由于烟草产品的市场需求具有季节性和周期性波动的特点，通过使用历史销售数据构建时间序列预测模型，可以准确预测未来的销售趋势，为企业制定销售策略提供科学依据^[1]。

准确的销售预测有助于优化库存管理，减少库存积压和断货风险，提高供应链的整体效率^[2]。通过对烟草销售数据的分析，可以为政府部门提供决策支持，帮助制定更有效的市场监管政策和税收政策。企业通过数据分析可以更好地了解市场需求和消费者偏好，及时调整产品结构和市场策略，从而提升市场竞争力^{[3][4]}。通过分析各环节的销售数据，可以识别产业链中的薄弱环节，推动各环节的协同发展，提高整个烟草产业链的运作效率^[5]。

本研究重点解决的是预测分析问题，对 A1 和 A2 香烟品牌的未来销量进行预测，对 A3 和 A4 销售金额进行预测，构建集成学习模型，对 A5 香烟品牌的销量和销售金额进行联合预测。解决完这些问题将有助于更深入地理解，能够提升对烟草市场的理解和预测能力，为企业的战略决策提供强有力的数据支持。

1.2 文献综述

烟草是中国重要的财政收入来源，对其销售情况的准确预测有助于更好地制定生产和销售策略，提升市场竞争力。研究中国烟草销售情况，利用先进的时间序列预测方法，可以提高预测准确性，减少库存积压和短缺，优化资源配置。

多元逻辑回归分析在处理复杂关系和高维数据方面具有优势。决策树回归模型通过分割数据集，建立预测模型。这种方法易于解释且处理非线性数据效果良好^[16]；杨振宇在 2016 年通过决策树回归方法，对烟草销售进行了预测，取得了较好的效果^[17]；方大山等人通过多元逻辑回归模型，表明可以更准确地预测烟草销售量和销售金额^[18]；王敏菲等在 2023 年指出，随机森林回归是多个决策树模型的集成，通过投票机制提高预测精度。研究表明，随机森林在处理复杂数据集方面表现优异^[19]；丁福林等提出了 SVR 适用于小样本和高维数据的预测，能够在高维空间中找到最优超平面并在 2015 年提出，SVR 在烟草销售预测中表现出色^[20]；梯度提升回归通过迭代地构建多个弱模型，不断提升预测性能。张勇等在 2017 年发现，梯度提升回归在复杂数据集上的预测性能优异^[21]；Helmini 等（2019）指出，LSTM 在捕捉时间序列中的长短期依赖关系方面表现优异^[22]；Arunraj & Ahrens（2015）研究表明，SARIMA 结合量化回归能够有效处理具有季节性波动的销售数据^[23]。通过结合多个模型的预测来提高整体预测的准确性。这种方法可以包括随机森林、梯度提升机等技术，它们通过整合多个简单模型的学习结果来减少预测误差，特别适用于处理复杂和高维度的数据集。

针对问题中的销量和金额的预测，钟天乐等人指出，ARIMA-LSTM 混合模型能够有效地整合 ARIMA 模型的季节性分解能力和 LSTM 的长短期记忆能力，提高时间序列数据的预测精度。他们通过实证研究展示了该模型在烟草销售数据上的应用效果，强调了其在处理具有强季节性和趋势性的销售数据中的优势^[25]。李沛昂在其研究中探讨了

N-BEATS 模型在时间序列预测中的应用，并指出该模型通过基于神经网络的时间序列分解能力，能够有效地捕捉和预测复杂的非线性时间序列数据。他的研究强调了 N-BEATS 模型在烟草销售预测中的实用性和精确性^[26]。程浩与赵新星比较了基于 CatBoost 与梯度提升回归（GBR）的销售预测模型。他们发现，在处理大规模和多维度的烟草销售数据时，CatBoost 模型因其高效的分类特征处理能力而显示出更高的预测准确率。此外，他们还探讨了梯度提升回归模型在减少预测偏差中的优势^[27]。王晓东在其研究中分析了 Transformer 模型在时间序列预测中的应用。他指出，Transformer 模型凭借其注意力机制，能够有效地处理长距离时间依赖问题，这在烟草销售预测中尤为关键，尤其是在分析季节性和市场趋势时^[28]。

通过应用先进的时间序列预测模型，可以提高对烟草销售数据的预测准确性，帮助企业更好地制定生产和销售计划，减少库存积压和短缺现象。准确的销售预测有助于企业优化资源配置，合理安排生产、运输和销售，降低运营成本，提高经营效率。

烟草销售预测模型将为企业和政府提供重要的决策支持，推动行业的可持续发展。未来，可以结合大数据和人工智能技术，烟草销售预测模型将在精度、实时性、适用性和解释性等方面实现全面提升，为企业和市场带来更大的价值。

第二章 问题重述与分析

2.1 问题重述

烟草产业是我国重要的经济收入来源，国家对烟草实施专卖政策，严格控制生产和销售环节。为了研究烟草的销售情况，本研究利用每种烟草的数据（excel 文件，共 5 个文件）对未来销量和销售金额进行预测。本研究关注三个主要问题：

1. 未来销量预测：利用历史销售数据，为 A1 和 A2 品牌构建两种不同类型的时间序列预测模型，预测未来销量。

2. 销售金额预测：利用历史销售数据，为 A3 和 A4 品牌构建两种不同类型的时间序列预测模型，预测未来的销售金额。

3. 集成学习模型：在上述预测模型的基础上，构建一个集成学习模型，用于预测 A5 品牌的销量和销售金额。

2.2 问题分析

2.2.1 预测香烟品牌的未来销量

针对 A1、A2 香烟品牌未来销量的预测问题，首先需要对数据进行精细的预处理。这包括填补缺失值、鉴别和处理异常值、以及数据标准化等关键步骤。这些措施确保了数据的质量，为后续分析打下坚实基础。接下来，我们会创建销量与时间的相关特征，例如平均销量、滞后销量等，从原始数据中提取有助于模型预测的信息。考虑到香烟销量数据的复杂性，我们进行了详细的特征工程，包括季节性特征的提取和滞后特征的构建等。

本研究最终使用了混合模型 **SARIMA-LSTM** 混合模型和 **N-BEATS (Neural Basis Expansion Analysis)** 模型对 A1,A2 未来销量进行了数据预测。将使用交叉验证等方法进一步优化 LSTM 的超参数引入更多的特征，如假日、季节性特征等对销量的影响，进一步提高模型的预测能力，以及调整学习率、增加更多的 LSTM 层。

在预测未来销量时，本研究还将采用多种经典算法，如多元逻辑回归分析、决策树回归、随机森林回归、支持向量回归、线性回归、梯度提升回归、K 近邻算法（KNN）和神经网络回归。其中，CatBoost 和梯度提升回归因其稳定性和高准确性，被选为预测未来销量的最佳模型。同时，为了便于后续构建集成学习模型及模型精确性的对比，我们还纳入了 XGBoost 和一些能够捕捉时间依赖性的深度学习结构，特别是 LSTM 和 GRU 模型。这样的方法确保了我们能够全面且准确地预测 A1 和 A2 香烟品牌的未来销量，也为后续集成模型的构建做好基础模型。

2.2.2 预测香烟品牌的销售金额

针对 A3、A4 香烟品牌未来金额的预测问题，第一步是全面的数据预处理。这包括处理缺失值、数据规范化以及提取可能影响销售收入的有意义特征，本研究考虑到了季节趋势、历史销售数据和价格变化等特征，以提高模型的预测准确性。例如，历史销售数据被分解为趋势和周期组成部分，以更好地理解可能不立即显现的潜在模式。此外，还分析了定价数据以确定其对销售收入的影响，承认价格调整可能显著影响消费者行为，

从而影响销售收入。

本研究对于这个问题最终使用了 **XGBoost** 和 **Transformer** 模型对 A3、A4 未来金额进行了数据预测。将使用交叉验证等方法进一步优化，并进行超参数搜索，引入更多的特征，如假日、季节性特征等对销量的影响，进一步提高模型的预测能力，以及调整学习率等。

2.2.3 构建集成学习模型

在上述预测模型的基础上，构建一个集成学习模型，用于预测 A5 品牌的销量和销售金额。本节将详细说明如何通过结合不同的基本预测模型来形成一个集成学习模型以提高预测的准确性和可靠性。首先，针对 A5 品牌，我们使用已经训练好的多个基模型（包括多元线性回归、岭回归、LASSO 回归、ElasticNet、KNN、随机森林回归、GBDT、XGBoost 回归、AdaBoost 回归、BP 神经网络回归，以及长短期记忆神经网络（LSTM）和门控循环单元（GRU））。每个模型都独立地对销量和销售金额进行预测，然后使用一个元模型来整合这些基模型的预测结果。

集成学习的关键在于元模型的训练，我们采用的方法是交叉验证，即将数据集分为多个子集，每个基模型在除了一部分数据作为测试集的情况下训练，然后在测试集上进行预测。这些预测结果作为新的特征（称为元特征）输入到元模型中。元模型再次训练，以学习如何最有效地结合各个基模型的预测。为了验证集成模型的有效性，计算模型的多种评估指标，均方根误差（RMSE）、平均绝对误差（MAE）和决定系数（R-squared）。通过这些指标，我们可以比较不同模型的性能，并选择出表现最佳的集成模型。最后，基于所选择的最优集成模型，我们对 A5 品牌的未来销量和销售金额进行预测，为企业的决策提供数据支持。

通过这种集成学习的方法，可以有效地提高预测的准确性和稳定性，尤其是在面对高维度和非线性问题时，集成模型能够显著优于单一模型。

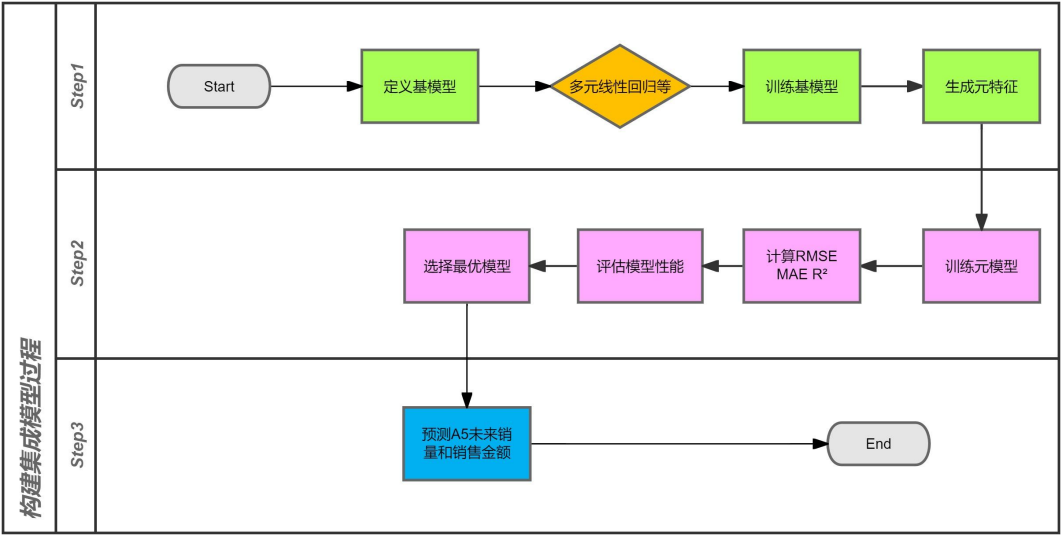


图 0 集成模型的构建过程

第三章 数据预处理与分析

3.1 数据集概述

本研究的数据集来源于某地区多种品牌的烟草销售数据，涵盖多个时间段的销售情况。数据已经过脱敏处理和数据变换，以确保数据的安全和隐私。不同品牌香烟共分为 5 个 Excel 文件。

每种品牌的烟草数据被保存在一个 excel 文件中，数据集分为 5 个 Excel 文件，每个文件包含 5 列，详细记录了各品牌烟草在不同月份的销售情况。数据的具体列的定义如表 1 所示：

表 1 烟草的数据表

变量编号	变量名	释义
1	月份	销售的自然月
2	样品代码	产品身份标识
3	名称	产品名称
4	销量（箱）	产品当月销售箱数
5	金额（元）	产品当月销售总金额

数据的主要用途是进行时间序列分析和预测模型的构建，以估算未来的销售量和销售金额。此外，该数据集还可以用于深入理解各个品牌在不同时间段的市场表现，评估市场策略的有效性，优化库存管理和销售策略。

这些数据将用于分析和预测不同品牌烟草的销量和销售金额，以帮助制定更有效的营销策略。通过对历史数据的分析和模型构建，可以为未来的销售趋势提供参考。

3.2 数据预处理策略

在数据分析中，数据预处理是构建准确且高效预测模型的关键步骤。预处理的目的是将原始数据转换为适合模型处理的格式。这通常包括执行探索性数据分析以了解数据分布、处理缺失值和异常值等问题。此外，还涉及数据类型的转换，如将分类数据转为数值型数据或标准化日期信息。通过这些可以显著提升模型的预测性能和处理效率。

对于日期的规范化处理，首先从 Excel 文件中读取数据，将“月份”字段分解为独立的“年”和“月”列，并删除其中的小数点以标准化“样品代码”字段为字符串类型。同时，将“销量（箱）”和“金额（元）”字段转换为整数类型。完成转换后，移除了原有的“月份”列，并将新生成的“年”和“月”列放置在数据框的前两列。最终，处理过的数据被保存为新的 Excel 文件。

年	月	样品代码	名称	销量 (箱)	金额 (元)
0	2011	01	43010103	A(绿和)	266 4785048
1	2011	02	43010103	A(绿和)	72 1300824
2	2011	03	43010103	A(绿和)	96 1735128
3	2011	04	43010103	A(绿和)	76 1375200
4	2011	05	43010103	A(绿和)	101 1824624
...	...	...	...	...	...
132	2022	03	nan	nan	<NA> <NA>
133	2022	04	nan	nan	<NA> <NA>
134	2022	05	nan	nan	<NA> <NA>
135	2022	06	nan	nan	<NA> <NA>
136	2022	07	nan	nan	<NA> <NA>

137 rows x 6 columns

年	月	样品代码	名称	销量 (箱)	金额 (元)
0	2011	01	43010104	A(硬)	235 2646630
1	2011	02	43010104	A(硬)	0 2115
2	2011	03	43010104	A(硬)	109 1223100
3	2011	04	43010104	A(硬)	195 2195640
4	2011	05	43010104	A(硬)	298 3349260
...	...	...	...	...	...
103	2019	12	nan	nan	<NA> <NA>
104	2020	02	nan	nan	<NA> <NA>
105	2020	03	nan	nan	<NA> <NA>
106	2020	04	nan	nan	<NA> <NA>
107	2020	05	nan	nan	<NA> <NA>

108 rows x 6 columns

图 1 格式转换后的数据展示

接下来就进行数据清洗。在这个阶段，对不同品牌的销量数据进行了异常值处理。具体来说，销量（箱）小于 2 的记录被视为异常值，并进行了删除，因为这些数据可能不具代表性（0.6 被视为小于 1 箱的记录）。通过清理这些异常值，可以确保数据的准确性，提高模型预测的稳定性。例如 A1,A2 被删除的异常值如下表：

表 2 如 A1,A2 的异常值数据

年	月	样品代码	名称	销量（箱）	金额（元）
2013	12	43010103	A(绿和)	0.0	720.0
2011	2	43010104	A(硬)	0.0	2115.0
2012	1	43010104	A(硬)	1.0	7065.0
2013	12	43010104	A(硬)	0.0	630.0

以上这部分异常值可能是由于禁烟令、税率调整、广告限制等相关政策导致。在数据清洗过程中，本研究还对原始数据集进行了可视化分析。通过计算各品牌每箱的平均价格，分析了 2015 年 4 月至 6 月期间的价格变动情况。数据可视化结果如下，揭示了不同品牌在该期间的价格波动情况。通过查阅相关文献，我们得出了一些重要结论，这些结论帮助解释了数据中观察到的趋势，进一步增强了对市场行为和价格变化的理解。

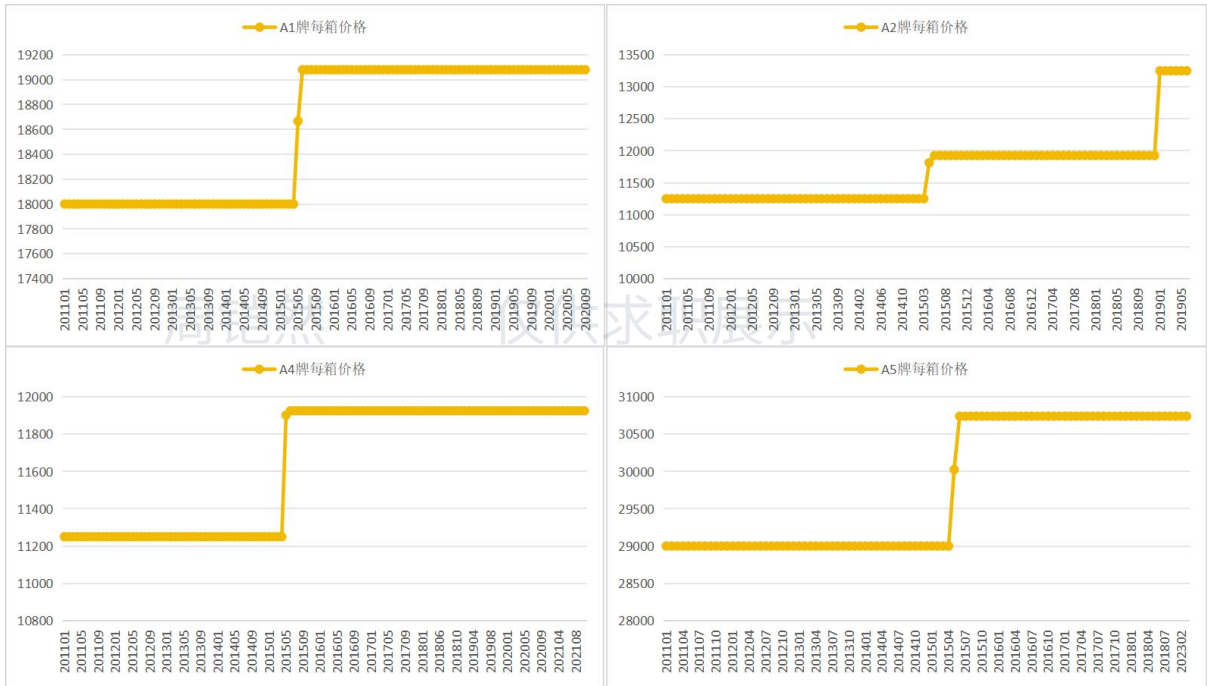


图 2 各个数据集的每箱价格

可以看出，2015 年 5 月的烟草价格在几个品牌中都出现了显著的上涨。2015 年 4 月到 6 月，中国的烟草市场经历了显著的价格变化。这主要是由于政府在 2015 年 5 月实施的烟草税调整^[3]。数据显示，平均批发价格上涨了 8.9%，零售价上涨了约 10.29%。研究指出，税收增加不仅影响了价格，还改变了消费者的品牌选择行为^[4]。

这种政策变化不仅减少了卷烟的消费量，特别是在低收入群体中显著减少，还增加了政府的税收收入。我们查阅的文献普遍建议，为了持续改善公共健康状况，未来需要进一步提高卷烟税率，以使价格维持在更高水平，减少烟草产品的可负担性^{[5][6]}。

最后，为了优化模型的学习和泛化能力，本研究使用了 StandardScaler 进行数据标准化处理。StandardScaler 将数据的均值调整为 0，标准差调整为 1，有效消除了不同特征的量纲差异，使得不同尺度的数据可在同一水平上比较和分析。以下是部分标准化后

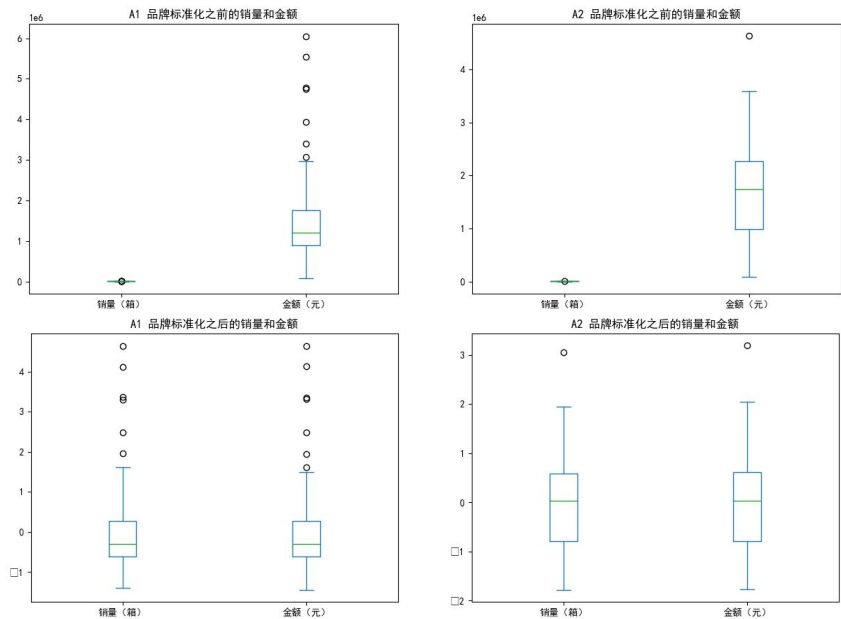


图 3 销量、金额标准化前后对比

总之，数据预处理阶段将原始数据有效地转换为适合深度分析和模型构建的数据集，为后续的研究工作奠定了坚实的基础。

3.3 特征筛选及优化

特征选择与优化在预测模型的构建过程中起到关键作用，决定模型性能与预测精度。在本研究中，进行了多个特征工程步骤以优化特征集。

首先，题目中包含的 5 个数据文件分别代表了不同品牌的烟草销售数据，这些数据已经按品牌分好类。我们对不同品牌的销售数据进行了可视化分析，并计算了每个品牌的总销量、平均销量、总金额和平均金额，以及把每个品牌的每箱烟草价格都计算出来。这些特征分别揭示了各品牌的烟草销售状况以及消费者对不同品牌的偏好程度。

此外，通过计算每个品牌每年的总销售量以及总体购买量，我们可以得出不同品牌烟草的受欢迎程度和消费者的购买意愿。这些分析有助于了解各品牌在市场上的表现和消费者的偏好。

同时，由于烟草销售可能受到季节性因素的影响，创建季度特征可以帮助模型捕捉到这些季节性波动^[6]。比如某些季度的促销活动、节假日消费习惯或天气变化都可能影响烟草的销售量。通过计算每个月所属的季度这个新特征，为模型提供了一个用于识别季度性销售模式的基础。

烟草的销售数据具有自然的年度周期性，例如春节或其他节假日可能会影响销售峰值或谷值^[7]。为了让模型更好地捕捉这一周期性，本研究使用正弦和余弦函数将月份转化为周期性特征。正弦和余弦特征能够表达数据中的周期变化，使得模型能够更容易识别和预测每年不同月份的销售变化趋势。

在特征工程的最后一步，我们对不同品牌的烟草数据进行了分箱处理，将连续的总销量和总金额转换为分类变量。这一处理方式揭示了不同时期品牌烟草的分布情况，同时也有助于后续基于不同数据集上的预测准确性均值进行综合评分。

通过这些工作，我们从原始数据中提取出了一系列有意义的特征。这些特征不仅提升了模型的预测能力，还为后续的数据分析和模型构建提供了有力支持。

经过精细的数据预处理和特征工程后，通过数据可视化，检查数据的连续性，确保

数据没有断点，每个月都有数据，以避免在预测时出现问题。比如对于 A1 文件，原始数据有 117 个月，但是经过数据预处理和特征工程筛选之后，时间数据占比如下图。

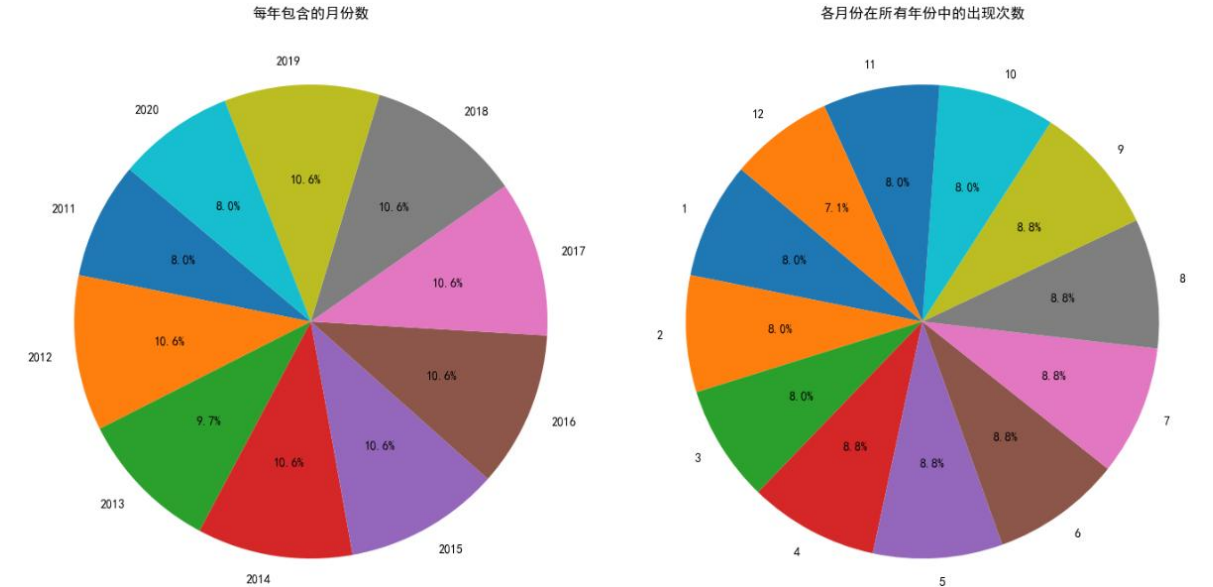


图 4 检查数据的连续性

从可以看到 A1_4 文件，除了部分年份（2011、2020、2013）显示的比例小于其他年份，这表明这些年份的数据不完整，只包含部分月份的数据。通过检查发现，这三个年份缺少的月份为：2011[1, 2, 3]，2013[12]，2020[10, 11, 12]。而原始数据中的 2021 年 1 月份和 2022 年 1 月份是缺失的。

3.4 缺失值处理

对于烟草数据更复杂的情况，尤其是当缺失值的缺失可能影响到关键决策时。在烟草销售数据中，“销量（箱）”和“金额（元）”在一年中的有些月份缺失，本研究使用了样条时间插值方法进行插值。考样条时间插值是处理时间序列数据中缺失值的一种有效方法，尤其适用于烟草销售这类可能受到季节性影响的数据。样条插值使用分段多项式来逼近数据，每个分段之间在端点处连续且光滑。这种方法在金融、气象和市场分析等领域常见，因为它能够很好地捕捉数据的自然波动和季节性模式。

首先，确保数据以时间序列的格式组织，且时间标签正确。我们将数据中的“月份”列转换为日期格式，并设为数据框的索引。接着提取已知的销量（箱）和金额（元）数据点作为样条插值的基础。

本研究将以 A1_4 中 2011[1, 2, 3]数据为例进行解释。首先选择已知数据点作为插值节点，如 2010 年 12 月至 2011 年 4 月的数据，这些数据点将作为构建样条的基础。

根据插值节点，构建三次样条插值模型。三次样条通常具有以下形式：

对于每个区间 $[X_i,X_{i+1}]$,样条函数 $S(x)$ 可以表示为：

$$S_i(x)=a_i+b_i(x-x_i)+c_i(x-x_i)^2+d_i(x-x_i)^3$$

其中 a_i,b_i,c_i,d_i 是区间 $[X_i,X_{i+1}]$ 上的系数，它们需要通过以下条件求解：

- 插值条件： $S_i(x_i)=y_i$ 和 $S_i(x_{i+1})=y_{i+1}$
- 光滑性条件： $S_i'(x_{i+1})=S_{i+1}'(x_{i+1})$, $S_i''(x_{i+1})=S_{i+1}''(x_{i+1})$
- 边界条件（自然边界）： $S''(x_1)=0$, $S''(x_n)=0$

接着使用 python-SciPy 库中的 InterpolatedUnivariateSpline 函数自动处理对三个系数进行求解，即可预测缺失的月份数据。

第四章 方法与技术

4.1 对未来销量进行预测

针对未来销量进行预测任务，本研究本先采用了 Prophet 模型，但由于附件数据为月度数据，故导致预测效果不理想，本研究没采纳。接着，采用了 SARIMA- LSTM-Transformer 混合模型以及 N-BEATS (Neural Basis Expansion Analysis)模型对模型进行详细的建立求解，并且进行了详细的超参数调优、设置早停机制、调整学习率以及增加更多的 LSTM 层以确保模型的性能最佳化。

为对未来销量进行预测，本研究基于前期数据处理和特征工程，深入分析并提取了在连续的时间段内的平均月销售量，平均年销售量，年总销售量，每季度销售量等相关特征。

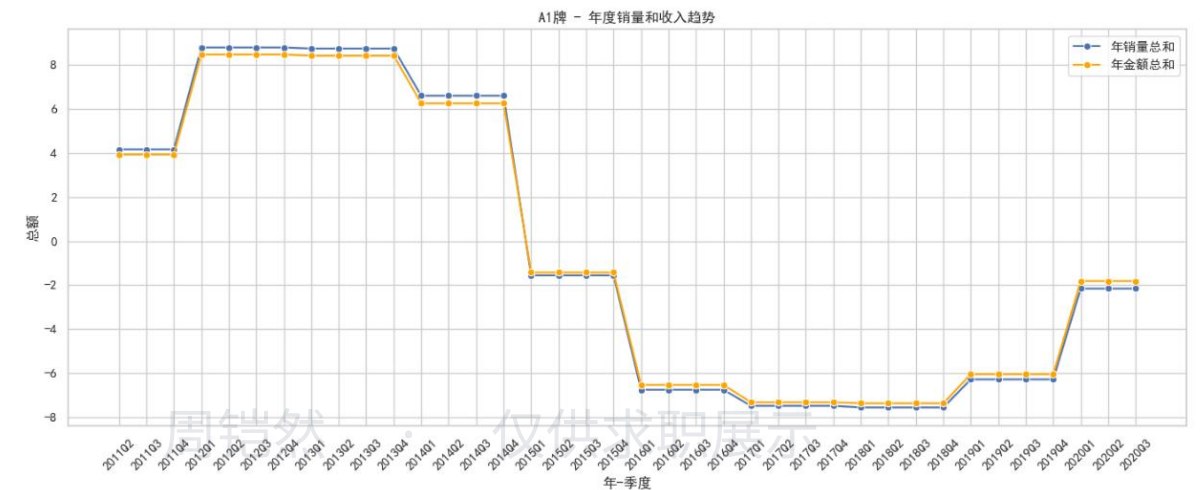


图 5 A1 牌特征工程及标准化

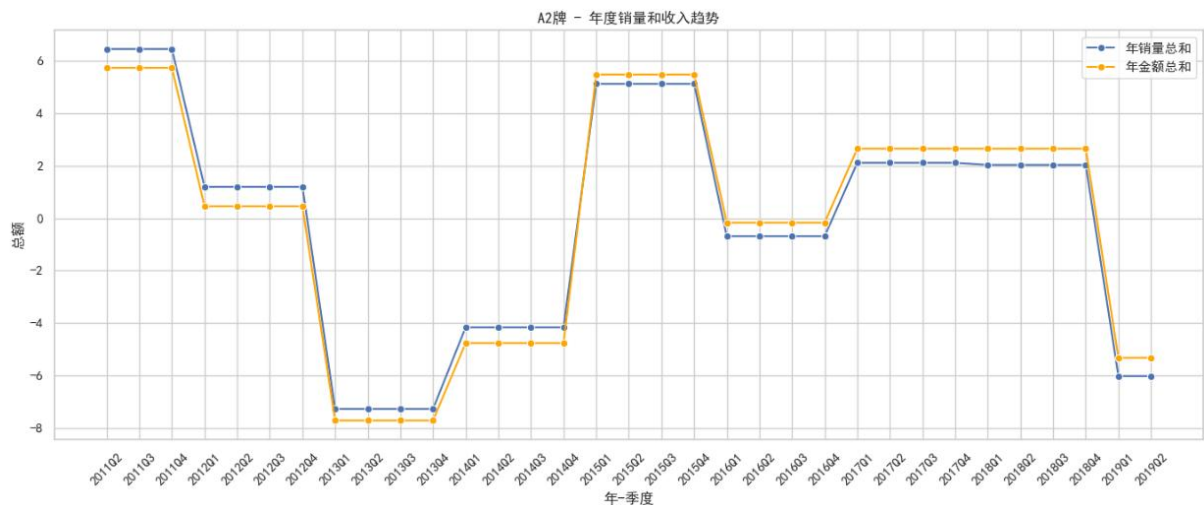


图 6 A2 牌特征工程及标准化

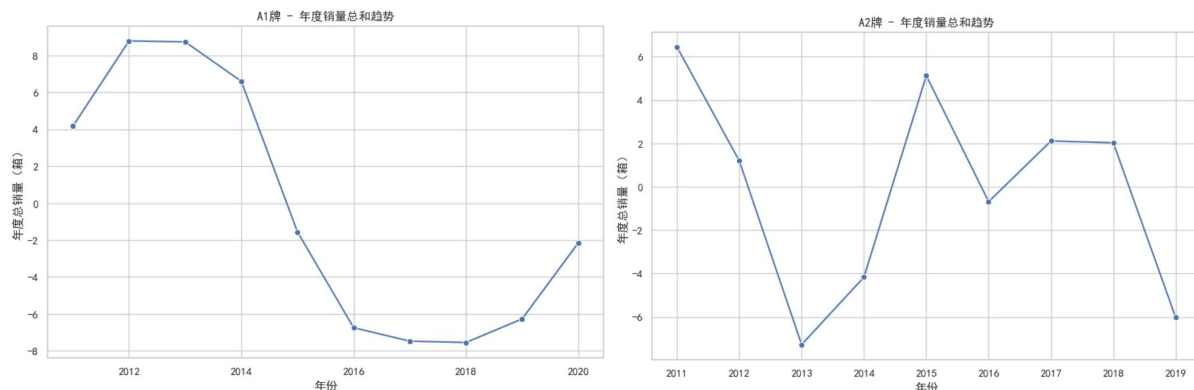


图7 A1、A2 牌数据趋势初步分析

根据图表，A1 牌烟草在 2011 年到 2020 年的销售额和收入趋势显示出显著的变化。2011 年至 2015 年期间，A1 牌烟草的销售额和收入保持在较高水平，2015 年底出现显著下降，由相关文献得知，这可能是由于市场调整和政策变化。2016 年至 2019 年，销售额和收入处于较低水平，这段时间中国烟草行业推进供给侧改革，减少产量以清理库存，导致销售额和收入下降^[12]。

国家通过烟草专卖制度对烟草生产和流通进行严格管理，确保市场秩序和税收来源。假烟和走私活动对市场秩序造成了严重冲击，国家加大了打击力度，通过信息化建设加强监管^{[14][15]}。此外，近年来控烟宣传和政策执行的加强使公众对吸烟危害的认知增强，吸烟率有所下降，导致烟草需求减少^[16]。这些因素共同作用，影响了 A1 牌烟草的销售额和收入趋势。

对于 A1、A2 牌销售量的预测，本研究还采用了多元逻辑回归分析、决策树回归、随机森林回归、支持向量回归 (SVR)、线性回归、梯度提升回归 (Gradient Boosting Regression)、K 最近邻回归 (KNN)、神经网络回归 (Neural Network Regression) 作为预测基础。本研究对这些模型进行了对比和评估，最终选择一个准确率最高的一个模型来预测 A1、A2 牌烟草的销售量。由于 A1 文件中空白项并没有 202101 和 202201，A2 文件空白项没有 202002，所以 A1 的 test size 设定为 22，A2 设为 11。在模型的训练和验证环节，本研究还利用交叉检验技术来确保模型的泛化表现，通过比较各个模型在验证集上的准确率，选出了最佳预测模型。（此部分的详细训练过程在附录 0）

4.2 SARIMA + LSTM 混合模型

本节详细探讨了 SARIMA-LSTM 混合模型在烟草销售预测中的应用，结合代码实现和数据处理过程。本模型结合了 SARIMA 模型的长期趋势分析能力与 LSTM 网络的短期行为学习能力，以提高预测精度。

首先读取销售数据，将月份列转换为日期格式，以便进行时间序列分析。接下来，删除了不相关的样品代码和名称列，因为这些数据对预测未来销量没有直接帮助。数据清洗是任何预测模型中的重要步骤，确保了模型训练的质量和后续预测的准确性。

使用 `auto_arima` 函数自动寻找最佳的 SARIMA 模型参数，该步骤考虑季节性因素，确保模型能够识别数据中的周期性变化。自动选择的过程中，`auto_arima` 通过多种参数组合来评估模型，选择 AIC 最小的模型，以期达到最佳的拟合效果。此外，计算模型残差，为后续的 LSTM 模型提供可以学习的误差信息，以进一步优化预测。

对于 LSTM 网络的构建和调优，本研究使用 `MinMaxScaler` 对残差数据进行归一化处理，这是神经网络训练的常见步骤，有助于加速模型的收敛。归一化处理后，通过自定义的 `create_dataset` 函数生成用于 LSTM 训练的数据集，保证数据的序列性和结构正

确。

利用 `keras_tuner` 库进行 LSTM 网络的超参数优化，这包括选择不同数量的神经元、优化器类型等。通过 `EarlyStopping` 回调来避免过拟合，确保模型在训练过程中若性能不再提升则提前停止。最后将预测和结果融合 LSTM 模型训练完成后，用于预测未来的残差，结合 ARIMA 模型的预测结果，得到最终的销售预测。这种融合方法利用了两种模型的优点，SARIMA 在长期趋势预测上的稳定性和 LSTM 在捕捉数据短期波动上的灵敏性。最后，使用 `Matplotlib` 库对实际销量、SARIMA 预测和混合模型预测进行可视化，直观展示模型预测的效果。

通过这一复杂的模型构建和优化过程，SARIMA-LSTM 混合模型能够有效地提升烟草销售数据的预测精度，为企业的销售策略和库存管理提供科学依据。

4.3 Prophet 模型

Facebook 开发的 Prophet 预测模型专为处理表现出强烈季节性模式的时间序列数据而设计，适用于假日影响显著的数据。Prophet 在处理缺失数据、趋势变化和大的异常值方面表现出色，非常适合预测任务，如本研究中的预测销售数据。

Prophet 通过将时间序列分解为三个主要组成部分来工作：

1. **Trend**: 模拟时间序列的非周期性变化。
2. **Seasonality**: 代表周期性变化，即每周、每月、每年。
3. **Holidays**: 包括可能不规律的时间表上的假期影响。

Prophet 模型可以用以下方程表示：

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t$$

其中：

- $y(t)$ 是预测值。
 - $g(t)$ 是趋势函数，用于模拟非周期性变化。
 - $s(t)$ 是季节性组件，用于模拟周期性变化。
 - $h(t)$ 表示假日的影响，这些假日可能在不规则的时间表上发生。
- ϵ_t 是误差项，用于解释模型无法适应的异常变化。

Prophet 中的趋势 $g(t)$ ，采用分段线性或逻辑增长曲线来捕捉数据系列的非周期性变化。这种灵活性允许趋势随着时间的推移适应系列行为的变化。季节性 $s(t)$ 通过使用傅里叶级数来拟合日常、周和年度周期性。这允许捕获在固定标准间隔重复的模式。假日组件 $h(t)$ 通过包含假日和其他特殊事件的指示来建模。这些可以显著影响预测，例如销售中的促销或事件。

预测销售时，Prophet 可以有效捕捉诸如在特定月份或特殊事件日销售增加等季节性趋势。它可以优雅地处理数据边缘，适应突然跳跃和下降，而不会过度拟合噪声。本研究查阅相关文献得知烟草销售数据的特点。销售数据会由于节假日，比如春节等销售而不可预测地飙升，将这些作为假日进行建模，Prophet 将调整预测以适应这些事件。

4.4 对销售金额进行预测

为了对销售金额进行预测，本研究基于前期数据处理和特征工程，深入分析并提取了相关特征后构建了以下 2 个不同类型的时间序列预测模型。

4.5 XGBoost

XGBoost (eXtreme Gradient Boosting) 是一种高效的机器学习算法, 尤其适合在结构化数据上进行分类和回归任务。它基于梯度提升机 (GBM) 框架, 但优化了速度和效率, 并增加了正则化项以控制模型的复杂度, 从而防止过拟合。XGBoost 在各种数据科学竞赛和实际应用中都表现出卓越的性能。

在销售金额预测的时间序列分析中, XGBoost (极端梯度提升) 因其处理各种数据类型的效率和稳健性而显著, 尤其是在处理数据中的非线性关系和交互时表现出色。XGBoost 是梯度提升的高级实现, 在各种预测建模挑战中已证明其有效性。

使用 XGBoost 预测 A3 和 A4 香烟品牌的销售金额包括几个关键步骤:

1. 数据预处理: 首先需要通过选择可能影响销售金额的相关特征来准备数据。这包括历史销售数据、经济指标及其他品牌特定变量。数据转换和标准化可能也是必需的, 以对齐不同特征的规模并处理潜在的异常值或数据分布的偏斜。

2. 特征工程: 时间序列预测中使用 XGBoost 从创建时间特征 (如滞后、移动平均或与时间相关的虚拟变量) 中受益, 以捕捉数据中的季节性趋势和周期。

3. 模型规范: 设置 XGBoost 模型以预测销售金额。这涉及到配置多个参数, 如树的数量 (`n_estimators`)、树的深度 (`max_depth`) 和学习率, 这决定了模型学习的速度。

模型训练涉及使用历史数据来拟合 XGBoost 模型, 而验证通常使用时间序列交叉验证技术完成。这确保了模型在未见数据上的测试, 模拟了它在现实世界情景中的表现。

交叉验证: 实施滚动预测场景, 其中模型在不断扩展的数据窗口上进行训练, 可以验证模型效果和稳健性。

超参数调优: 使用网格搜索或随机搜索来优化模型的超参数, 可以显著提高模型性能。关键参数包括:

`n_estimators`: 执行的提升阶段总数。更多的阶段可能提高模型的准确度, 但也可能导致过拟合。

`max_depth`: 每次提升中树的最大深度。增加此值将使模型更复杂, 更可能过拟合。

`learning_rate`: 在每次提升后缩小特征权重, 使得提升过程更为保守。

性能评估: RMSE (均方根误差)、MAE (平均绝对误差) 和 R-squared (决定系数) 指标来评估模型, 以评估模型预测销售金额的准确性。

误差指标: 这些指标有助于确定预测值与实际值有多接近, 较低的 RMSE 或 MAE 表示更好的拟合。

4.6 决策树模型

决策树模型是一种常见的数据挖掘方法, 用于分类和回归任务。它通过一系列规则对数据进行分割, 每一步分割都基于最优化某个特定标准的决策, 这种方式类似于树状图的分枝过程。决策树的主要优点是模型易于理解, 不需要任何统计知识, 用户可以很容易地理解模型决策路径。

决策树模型特点如下:

1. 是直观性强。模型结果易于理解, 非技术背景的用户也能 grasp 模型的决策逻辑。

2. 不需要归一化: 决策树不要求特征标准化或归一化。

3. 处理非线性关系有效: 能够有效地处理数据间的非线性关系, 适用于复杂的分类边界。

4. 自动特征选择: 决策树能够在训练过程中进行特征选择, 选最优特征进行分裂。

对于时间序列数据，尽管决策树不是传统意义上用于时间序列预测的工具，通过适当的特征工程，决策树仍然可以有效地应用于预测任务。在预测 A3 和 A4 香烟品牌的销售金额时，可以通过以下步骤使用决策树模型：

1. 特征工程：转换时间序列数据为监督学习问题，通过滞后 (Lags)、移动平均 (Moving Averages)、时间窗口统计 (Rolling Statistics) 等方法创建特征。
2. 模型训练：使用决策树模型来拟合训练数据。在这个过程中，可能需要调整决策树的深度和叶节点数，以控制模型复杂度和防止过拟合。
3. 模型验证：使用时间序列的特定交叉验证方法，如时间序列分割 (TimeSeriesSplit)，来评估模型的预测性能。
4. 性能评估：使用 RMSE、MAE 等指标来衡量预测准确性。
5. 结果解释：决策树的结果易于解释，可以清晰地看到哪些特征对预测销售金额有重要影响。

4.7 构建 Stacking 集成模型

Stacking 集成方法是一种高级的机器学习技术，通过结合多个不同基模型的预测能力，并使用一个元模型来整合这些基模型的输出，从而提高整体预测精度。这种方法在处理复杂的预测问题时特别有效，尤其是在数据具有高维特征和非线性关系时。

本研究针对 A5 香烟品牌的销量和销售金额进行联合预测，首先进行了数据预处理和特征工程。随后，训练了多种模型，包括多元线性回归、岭回归、LASSO 回归、ElasticNet、KNN、随机森林回归、GBDT、XGBoost 回归、AdaBoost 回归、BP 神经网络回归，以及长短期记忆神经网络 (LSTM) 和门控循环单元 (GRU)。

在模型训练过程中，我们使用学习曲线来跟踪模型的学习进度。在最优模型的选择阶段，采用交叉验证方法，并使用均方根误差 (RMSE)、平均绝对误差 (MAE) 和 R 平方 (R-squared) 作为评估指标。数据集被分为训练集和测试集，保持时间顺序，将最后 11 个时间步作为测试集，其余作为训练集。每个基模型都使用完整的训练集进行训练，以尽可能多地从训练数据中学习信息。

通过交叉验证，每个基模型对验证集的预测结果将用于训练元模型，从而确保元模型能够从各基模型的预测中学习泛化特征。这一过程能够有效提升预测的准确性和稳定性。

在销售预测方面，集成学习方法展现出了显著的优势。例如，Pavlyshenko (2018) 在研究中表明，使用堆叠技术构建的回归集成模型能够提高销售时间序列预测的性能。Rajpoot et al. (2023) 的研究也指出，通过线性回归和 XGBoost 回归器构建的集成模型，可以有效提升销售预测的准确性。此外，Nti et al. (2020) 在其关于股票市场预测的研究中，发现堆叠和混合方法在预测精度上明显优于袋装和提升方法，这一发现同样适用于销售预测领域。这些文献的共同结论是，集成学习方法在处理复杂数据集和提高预测模型的泛化能力方面具有显著的优势。

本研究通过对 A5 香烟品牌销量和销售金额的预测验证了这一点，采用了包括多元线性回归、随机森林回归和长短期记忆神经网络在内的多种模型，最终通过堆叠方法整合这些基模型的输出，从而大幅提升了预测的准确性和稳定性。通过系统性的模型评估和优化，本研究结果进一步支持了集成学习方法在实际应用中的有效性和可靠性。

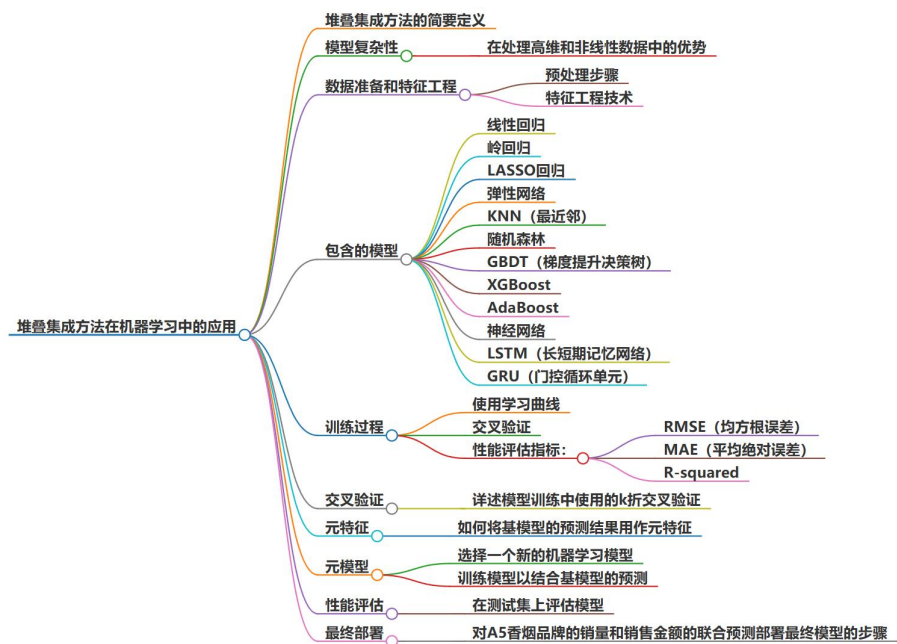


图 8 堆叠集成方法在机器学习中的应用

在研究过程中，基模型在训练集上的预测结果被用作“元特征”。元特征是指基模型的输出构成的新特征，这些新特征将被用作元模型的输入。为了生成这些元特征，本研究采用了交叉验证的方式来确保模型不会过拟合。具体而言，使用了 K 折交叉验证，其中每个基模型在 K-1 折(k=5)上训练，并在剩余的一折上进行预测，重复此过程直到每个数据点都被用作测试集。选择一个新的机器学习模型作为元模型。元模型的任务是使用前一步骤中生成的元特征来进行训练，学习如何最有效地结合基模型的预测，从而提高整体的预测精度。通常，元模型会选择与基模型不同类型的模型，以充分利用各模型的优势。在测试集上评估元模型的性能。测试集上的表现可以视为模型的实际表现。根据性能结果，可能需要调整模型的选择或其超参数。本研究还对模型进行了超参数搜索，此步骤可能需要多次迭代，以找到最佳的基模型和元模型组合。

一旦确定了最终模型，就可以在实际的应用场景中部署。此时，所有的预测工作都由元模型完成，它综合了所有基模型的预测智慧，从而提供更高精度的预测结果。

总的来说，堆叠集成方法通过充分利用基模型的优势，显著提升了销售预测的精度，为 A5 香烟品牌的销量和销售金额预测提供了强有力的技术支持。

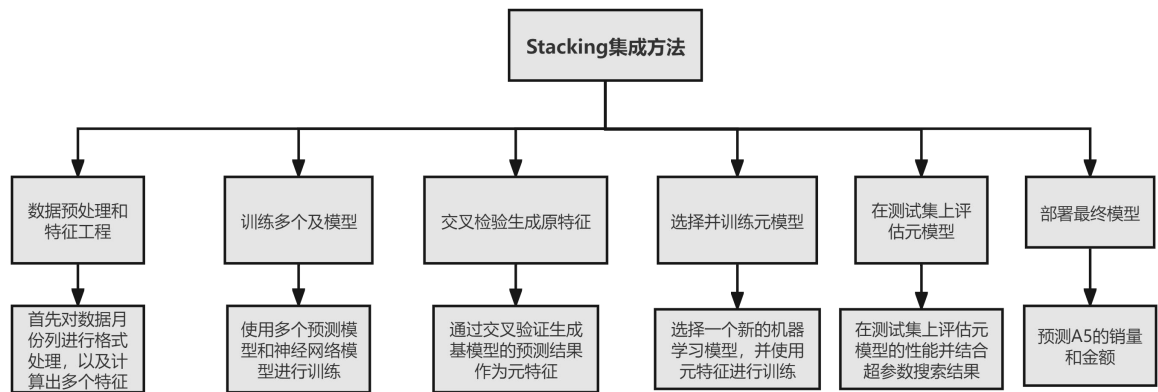


图 9 本研究中 Stacking 集成方法

第五章 结果展示

5.1 未来销量预测结果展示

关于 A1 和 A2 香烟品牌未来销量的数据预测问题，本研究进行了全面的数据预处理和特征工程，随后采用多种模型进行训练。这些模型包括 Prophet、多元逻辑回归分析、决策树回归、随机森林回归、支持向量回归 (SVR)、线性回归、梯度提升回归 (Gradient Boosting Regression)、K 最近邻回归 (KNN)、神经网络回归 (Neural Network Regression)。

在训练过程中，学习曲线被用于跟踪模型的学习进度。在最优模型的选择阶段，采用交叉验证方法，并将 RMSE（均方根误差）、MAE（平均绝对误差）和 R-squared（决定系数）设为评估指标。通过评估和比较，最终选择了在这些评估指标上表现最好的两种不同类型的模型。CatBoost 模型对于 A1 表现最佳，决策树回归对于 A2 表现最佳，在测试集上的 RMSE 和 MAE 最小且 R-squared 接近于 1。

在接下来的图表中，将详细展示各类时间序列预测模型的训练过程和最终结果，包括训练过程，展示模型在训练集和验证集上的学习曲线，帮助理解各模型的学习进度和过拟合情况，以及评估指标，展示模型在预测任务上的评估指标，如 RMSE、MAE 和 R-squared，以便直观比较各模型的性能。

附录 1 将呈现部分模型的训练过程和超参数搜索过程，提供深入和全面的视角，帮助理解和比较各模型的表现。

对于 A1 的未来销量与预测，本研究使用 python 中的 auto_arima 函数得到 Best model 为 SARIMA(2,1,1)(2,0,1)[12] intercept

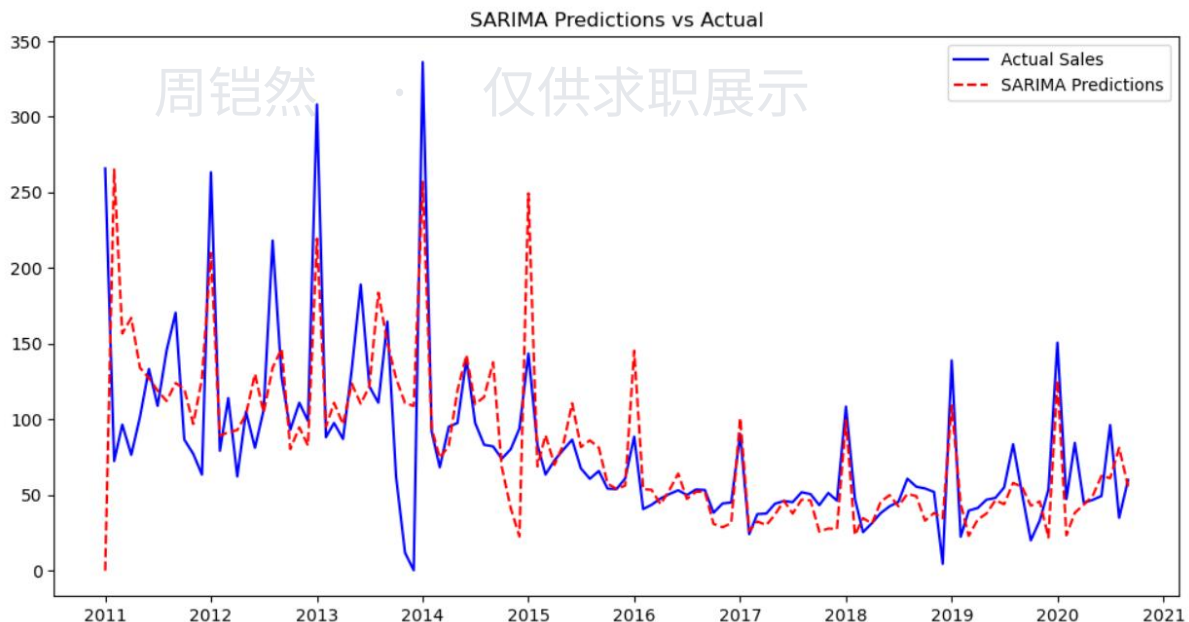


图 10 SARIMA Predictions VS Actual

实际销售和预测销售都显示出季节性趋势，表明 SARIMA 模型有效地捕捉了销售数据中的基本季节模式。有些时期 SARIMA 预测与实际销售非常接近如 2014 年中期至 2016 年，表明在这些时期模型非常有效。然而，在某些时期，如 2011 年初、2017 年底和 2020 年整年，预测与实际销售存在偏差。这些偏差可能由市场变化、异常事件引起，或者是模型参数未能捕捉到销售数据中的某些行为。

预测与实际销售之间的偏差表明了模型可能改进的领域。这可能涉及调整模型参数，加入额外的解释变量（如市场营销支出、经济指标或市场条件变化），或使用可能更好地捕捉数据动态的不同建模方法。

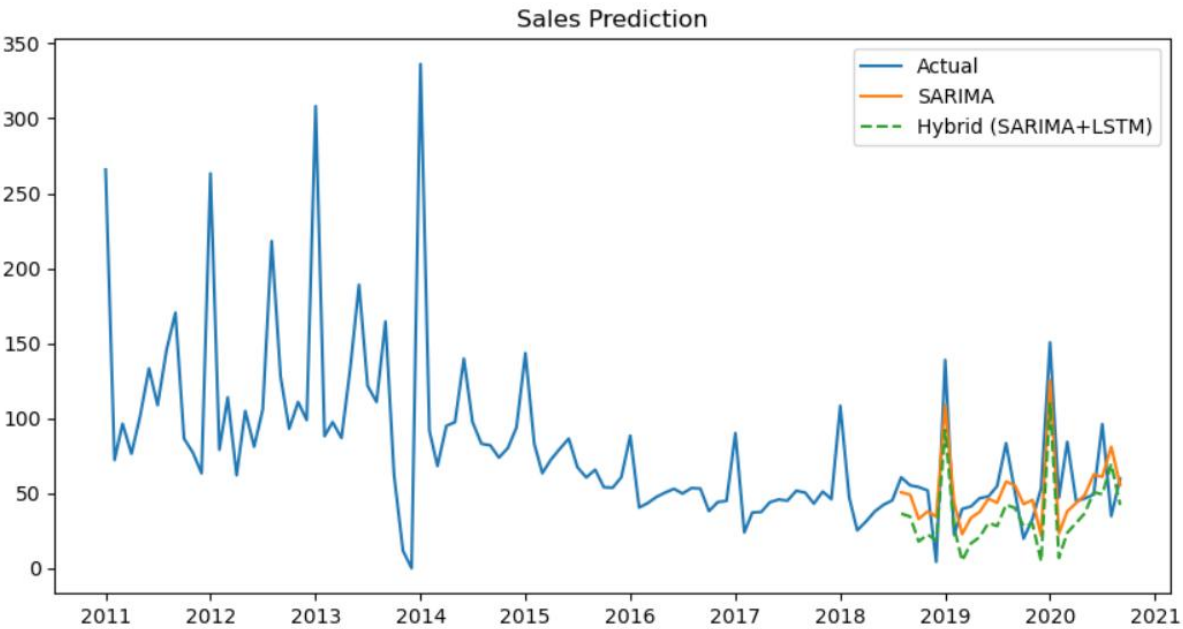


图 11 SARIMA-LSTM 预测效果

初始时，SARIMA 预测似乎跟踪实际销售的总体趋势，但在高峰或低谷期间存在一些明显的偏差。这可能表明模型在适应突然变化或捕捉数据中的高变异性方面存在限制。

在后期年份（2019 年之后）混合模型开始更明显地分化。它似乎更好地适应了销售数据中的急剧升降，与单独的 SARIMA 模型相比，表现出更好的适应性。这表明 LSTM 组件可能增强了模型对最近数据趋势的响应能力，这些趋势可能更加波动或受到之前训练数据中不存在的因素的影响。

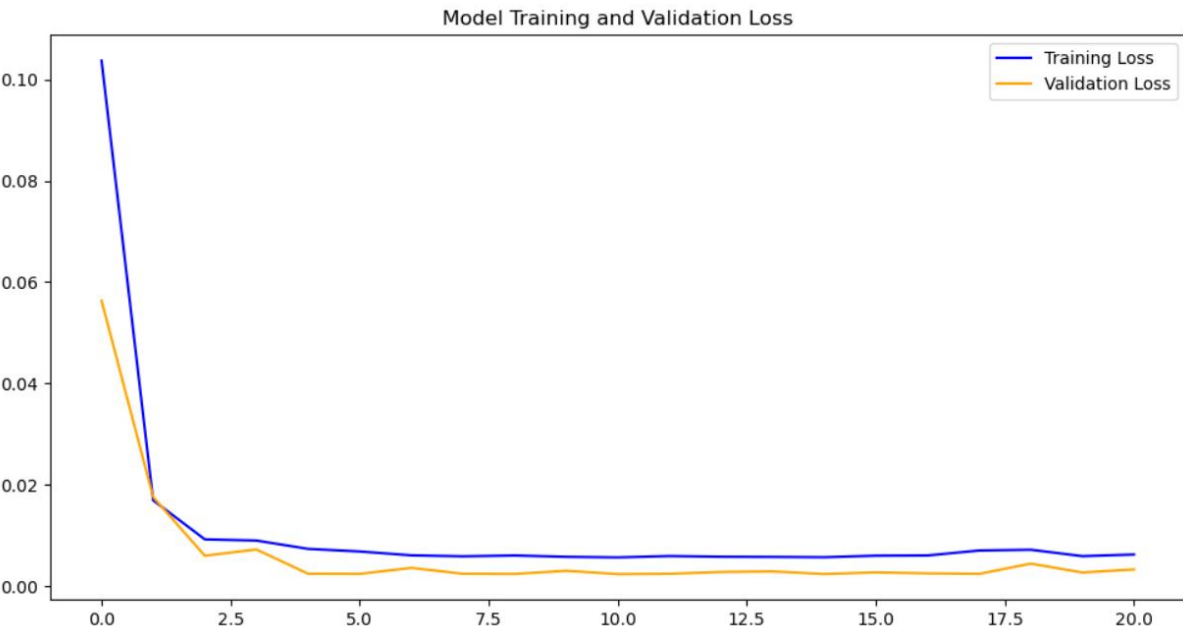


图 12 model training and validation loss

在图表所示的周期结束时，损失已稳定并且非常低，表明模型学得很好，并且在训

训练集和验证集上的误差都很小。

训练损失和验证损失在最初几个训练周期内急剧下降。这表明模型正在从数据集中快速学习，有效地减少其预测的错误。在初期急剧下降之后，两个损失都开始趋于平稳，并且下降速度变得更加缓慢。这表明模型正在接近收敛点，此时想要实现显著改进变得更加困难。整个训练过程中训练损失和验证损失非常接近，这是一个好现象。它表明模型没有过拟合。过拟合的表现是训练损失很低但与更高的验证损失发散。

对于 A2 文件销量预测，以下为关于 Prophet 模型的结果展示：

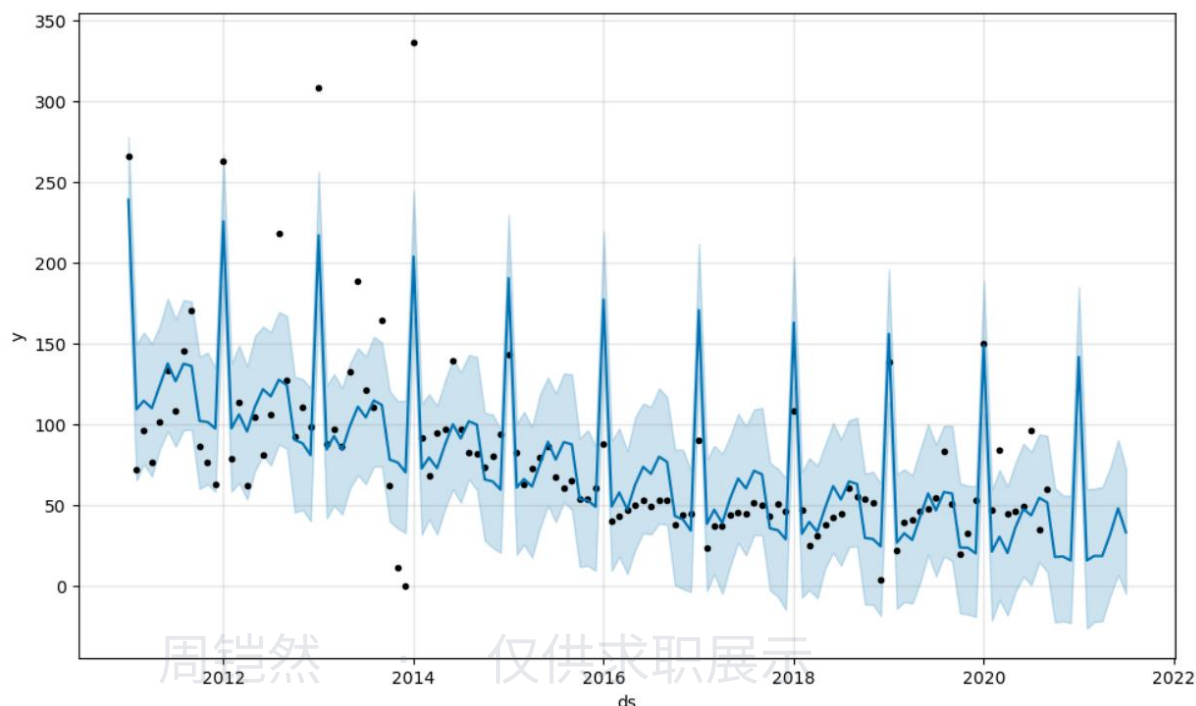


图 13 Prophet 训练过程

黑点（实际销售）这些代表了多年来的实际销售数据点。图表显示出相当大的波动，一些峰值远高于其他，这可能表明季节性高峰或特殊事件驱动的销售增加。蓝线（模型预测）这可能代表了预测模型的预测销售。模型似乎捕捉到了销售数据的整体趋势和一些季节性，但没有捕捉到所有的峰值，尤其是非常高的那些，这表明这些可能是异常值或由模型未包括的因素驱动的。蓝色阴影区域（置信区间）围绕预测线的阴影区域代表预测的置信区间，表明实际值有一定概率落在此范围内。阴影区的宽度表明预测的可变性或不确定性；区域越宽，不确定性越大。

模型似乎表明多年来趋势稳定或略有波动，没有明显的长期向上或向下趋势。模型似乎考虑到了一些季节性变化，这从预测中的规律性模式可以看出。然而，最高的实际销售峰值没有被预测到，这可能表明模型缺少驱动这些峰值的变量或特征。模型在捕捉系列的平均行为方面表现良好，但在极值方面表现不佳。这可能是由于数据的性质或可能的异常值，这些异常值不典型于常规的销售模式。

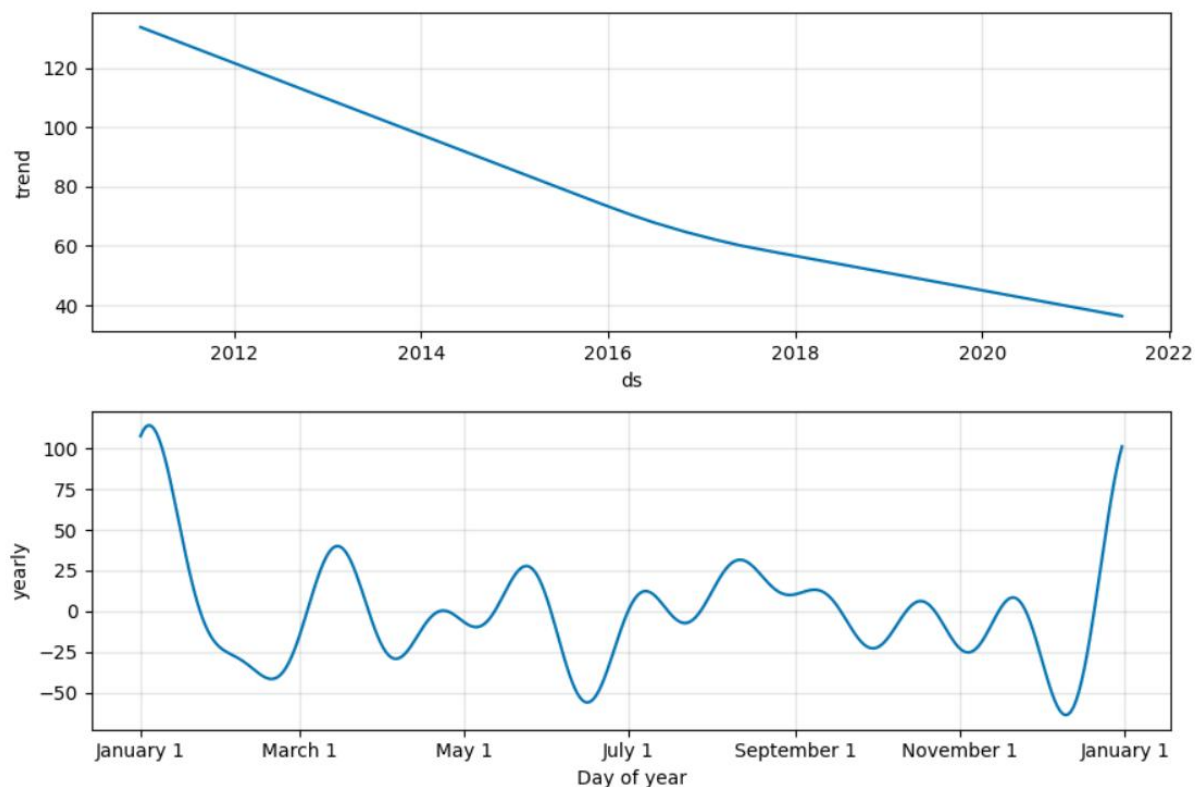


图 14 结合趋势和季节分析

该图表显示了从 2012 年到 2021 年的下降趋势线。持续的下降表明所分析的数据（如销售、用户等）长期减少。这可能表明市场收缩、用户兴趣减少或其他随时间降低的影响因素。识别出下降趋势对于做出战略决策至关重要，如加强营销努力、探索新市场或改变商业模式以抵消这种下降。

下图显示了重复的年度模式，每年初和七月初左右出现显著高峰，年末出现低谷。年初的显著高峰可能与新年促销、季节性销售或其他年度事件相关。年中的高峰可能与年中销售或事件有关，年底的低谷可能表明季节性放缓。理解这种模式有助于计划库存、人员配置和营销，以应对预期的需求增减。

通过结合趋势和季节图表的洞察力，企业可以更好地了解基本趋势和季节模式如何影响其整体绩效。例如，即使存在强烈的季节性高峰，总体下降趋势可能意味着每年高峰的影响正在减少。这些信息对调整商业策略至关重要，如在预期低迷期间增加客户互动努力，或重新评估产品或服务供应。



图 15 A1 和 A2 尝试使用 Prophet 模型预测

5.2 集成学习模型结果展示

针对集成学习模型构建问题，本研究旨在预测 A5 香烟品牌的销量和销售金额。研究开始前，我们已经对数据进行了详细的预处理和特征工程，包括将时间列转换为日期格式、避免科学计数法等。此外，还计算了每年的总销售量和销售金额，生成了平均销售量和平均销售金额的新特征，并考虑了季节性因素。

本研究中涵盖的模型有：多元线性回归、岭回归、LASSO 回归、ElasticNet 回归、KNN、随机森林回归、GBDT、XGBoost 回归、AdaBoost 回归、BP 神经网络回归、长短期记忆神经网络（LSTM）和门控循环单元（GRU）。在训练过程中，我们使用学习曲线来跟踪模型的学习进度。

接下来的图表将详细展示各个模型的训练过程和最终结果，包括模型在训练集和验证集上的学习曲线，以及在预测任务上的评估指标结果。这些图表将为理解和比较各个模型的性能提供全面而深入的视角。图 16 到图 30 将呈现部分模型的训练过程和超参数搜索过程。（以及更加详细的训练过程见附录 0）

表 2 各模型预测效果对比表

模型	最优参数	RMSE	MAE	R ²
多元线性回归	-	1.17859	0.96314	0.6468
岭回归	'alpha': 0.1	1.178486	0.96212	0.846971
LASSO 回归	'alpha': 0.01	1.17666	0.95724	0.74959
ElasticNet 回归	'alpha': 0.01, 'l1_ratio': 1.0	1.176669	0.957248	0.64959
KNN	'n_neighbors': 16	1.26303;1.2579	0.878963;0.882209	0.622 ; 0.25968
随机森林回归	'max_depth': 30, 'min_samples_split': 10, 'n_estimators': 200	1.18059	0.967381	0.7439033
GBDT	'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 300	1.771	0.9369	0.5
XGBoost 回归	'activation': 'tanh', 'hidden_layer_sizes': (50,), 'learning_rate_init': 0.01	1.59016	1.00578	0.8528
AdaBoost 回归	'max_depth':4, 'learning_rate': 1.0, 'n_estimators': 200	1.990	0.999	0.819
BP 神经网络回归	'activation': 'tanh', 'hidden_layer_sizes': (50,), 'learning_rate_init': 0.01	1.590164	1.00578	0.5528
LSTM	-	1.289	-	0.032
GRU	-	1.561	-	0.420

后续的图标将详细展示各个模型的训练过程和结果，以便对各个模型的性能进行深入的理解和比较，部分模型训练过程具体看图 16-图 30，将展示部分集成模型中涵盖的模型的超参数网格搜索过程以及模型的训练过程。

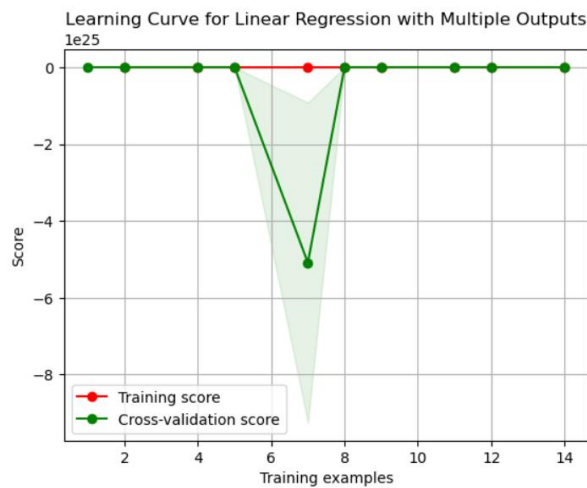


图 16 多元回归训练过程

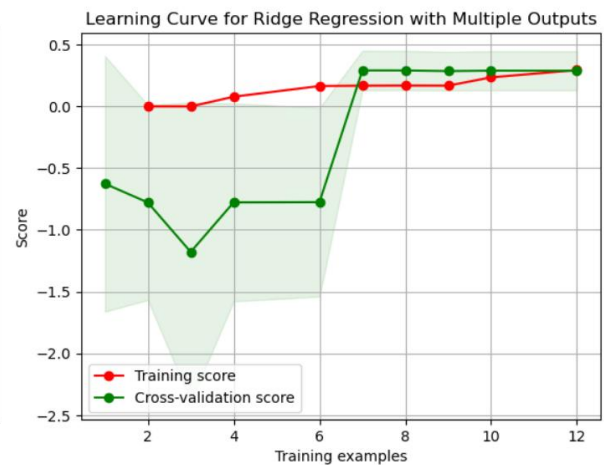


图 17 岭回归训练过程

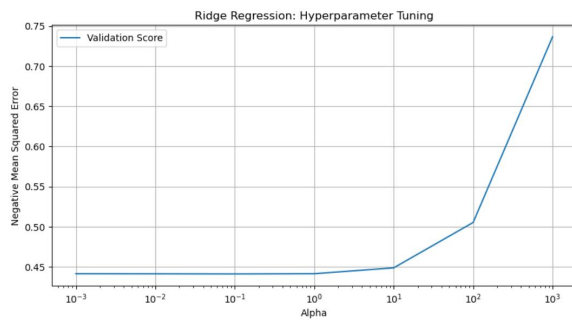


图 18 岭回归超参数网格搜索过程

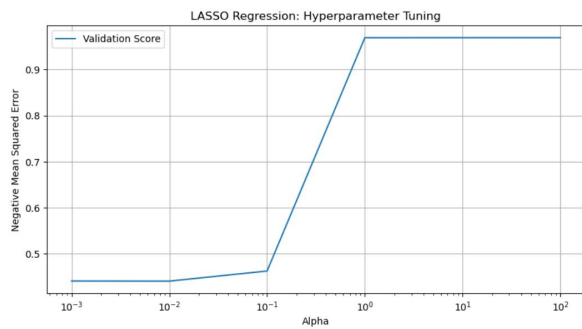


图 19 LASSO 回归超参数网格搜索过程

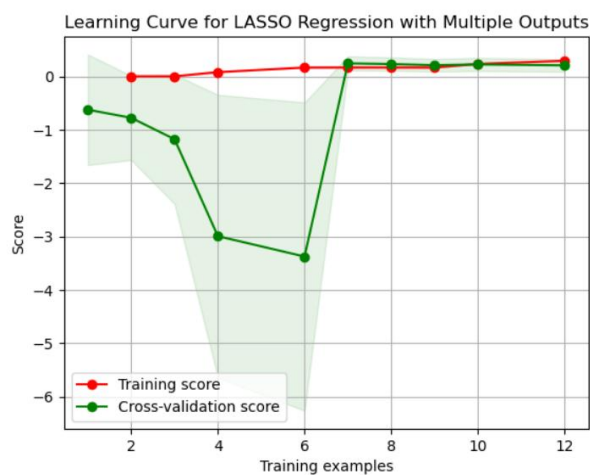


图 20 LASSO 回归训练过程

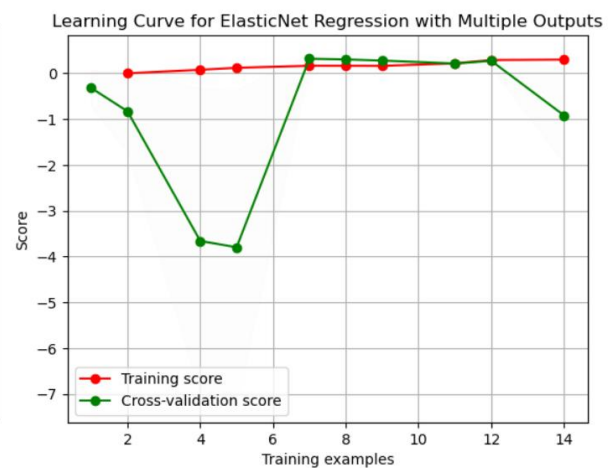


图 21 ElasticNet 回归训练过程

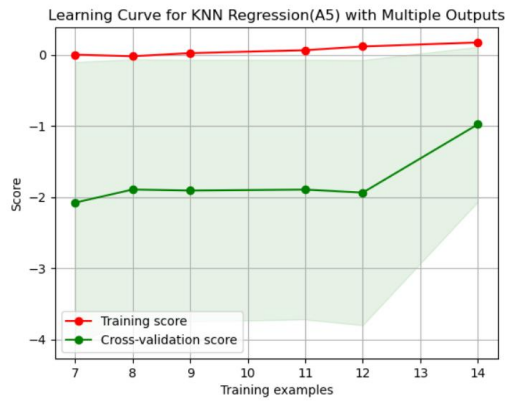


图 22 KNN 模型训练过程



图 23 KNN 超参数网格搜索过程

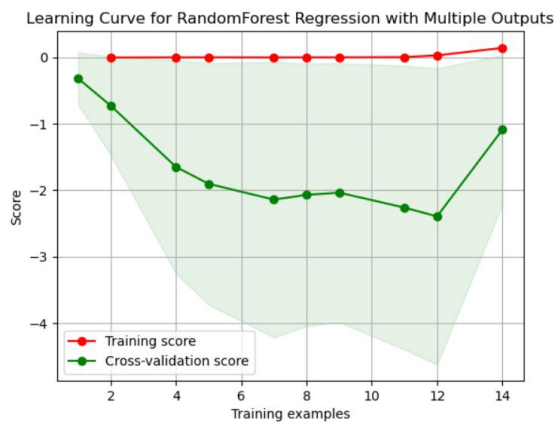


图 24 随机森林训练过程

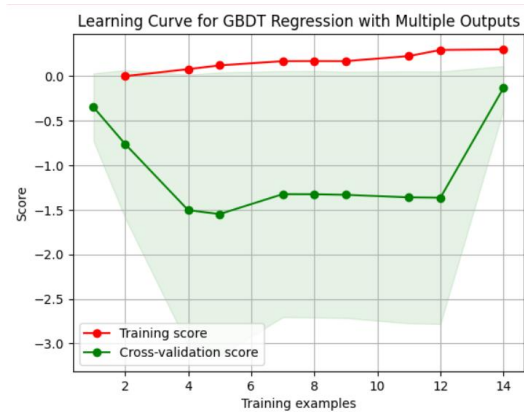


图 25 GBDT 训练过程

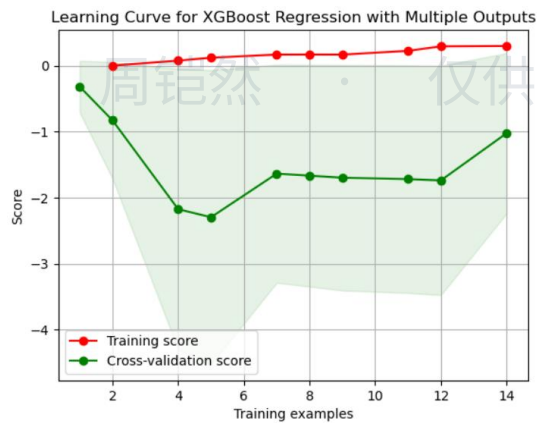


图 26 XGBoost 训练过程

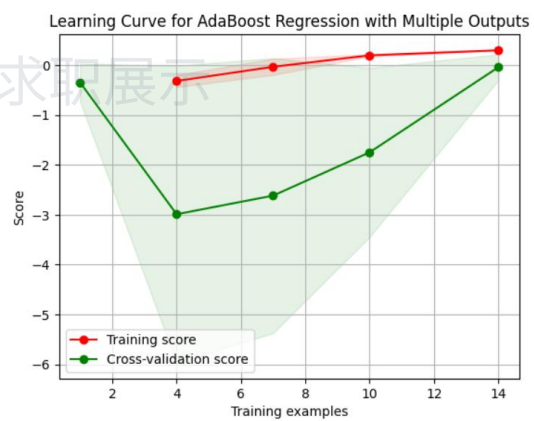


图 27 AdaBoost 训练过程

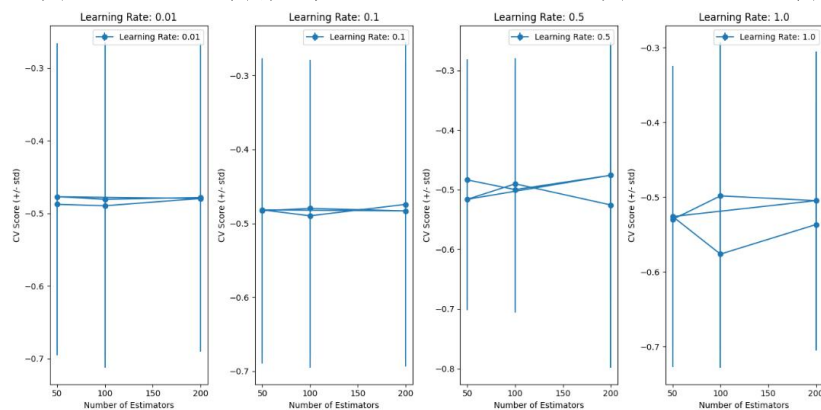


图 28 AdaBoost 超参数网格搜索过程

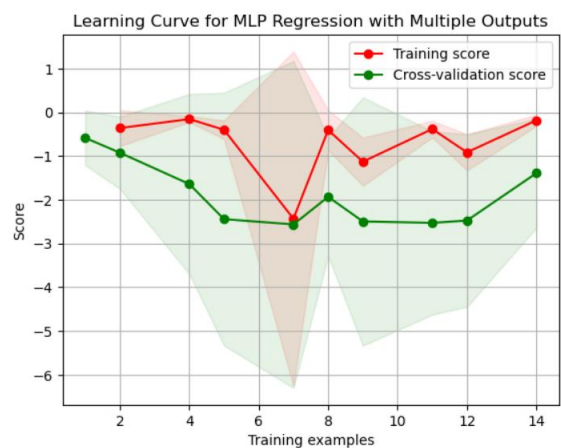


图 29 BP 神经网络回归训练过程

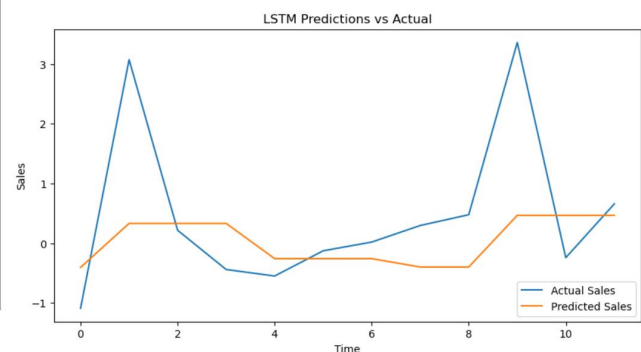


图 30 LSTM 预测与真实对比

第六章 讨论与深度分析

6.1 深度洞察与模型解读

本研究运用了多个模型进行求解，以下将对各模型的预测结果进行深入分析，解释模型的表现和预测结果。

本研究中，我们采用了多种模型进行烟草销量和销售金额的预测。这些模型包括 ARIMA + LSTM 混合模型、Prophet 模型、XGBoost 和决策树模型，以及构建了 Stacking 集成模型。

ARIMA + LSTM 混合模型：这个模型结合了 ARIMA 模型处理长期数据趋势的能力和 LSTM 网络捕捉时间序列短期依赖的能力。该混合模型在处理具有明显季节性和趋势性的销售数据时表现出优异的预测精度，尤其适合于复杂的时间序列数据分析。

Prophet 模型：Prophet 是一个灵活的模型，它可以捕捉季节性变化并处理假日效应，非常适合于具有强季节性特征的数据。它的主要优势在于能够自动识别和适应数据的趋势变化，对于烟草销售数据中的非线性季节性模式尤为有效。

XGBoost 和决策树模型：这些模型在处理非线性关系和复杂交互中表现出色。XGBoost 通过梯度提升框架提高了决策树的性能，是一种强大的预测工具，特别适用于大规模数据集。决策树模型因其易解释的决策路径而受到青睐，能够直观地展示如何基于数据特征做出预测。

Stacking 集成模型：通过结合上述不同的基模型，集成模型旨在利用各个模型的优点，通过一个元模型来整合这些预测，以提高总体的预测准确性。这种方法在提高预测稳定性和减少模型偏差方面非常有效。

6.2 研究限制与潜在改进

尽管使用了多种模型，每个模型都有其局限性。例如，ARIMA 模型不适合处理非线性数据，而 LSTM 和 Prophet 模型的计算成本较高。此外，模型的预测性能高度依赖于数据的质量和特征工程的准确性，数据的噪声和不完整性可能会导致预测误差。

为了进一步提高预测精度和模型的泛化能力,可以考虑引入更先进的特征选择和优化算法,以更好地捕捉数据中的隐藏模式和动态变化。此外,采用更复杂的深度学习模型,如深度信念网络或卷积神经网络,可能会在处理高维数据时提供更好的预测结果。同时,增加模型的解释性,使模型的决策过程更加透明和可信,也是未来研究的一个重要方向。

第七章 结论与未来研究方向

7.1 研究成果与贡献

本研究通过应用多种先进的时间序列预测模型,对烟草销售数据进行了深入分析和预测,提出了有效的预测方法,以增强烟草销售管理和政策制定的科学性和准确性。主要成果和贡献包括:

模型创新与应用点,本研究成功地应用了 ARIMA + LSTM 混合模型、Prophet 模型、XGBoost、决策树模型,以及 Stacking 集成模型。这些模型的结合不仅提高了烟草销售量和销售金额的预测准确性,也展示了不同模型在烟草行业数据分析中的应用潜力和优势。

研究通过理论推导与实际数据的结合,提供了一个全面的视角来理解烟草市场的动态变化。通过模型的比较和分析,揭示了各模型在处理烟草销售数据时的特点和优势。

研究结果可以帮助政府和相关机构更好地制定和调整烟草市场的管理策略,如税收政策、市场准入政策等。准确的销售预测有助于优化资源分配,减少市场波动带来的风险。

通过提高预测准确性,烟草企业可以更有效地管理库存,优化供应链,减少过剩或短缺的情况,从而降低成本并提高经济效益。

7.2 未来研究方向展望

为了进一步提高烟草销售预测的准确性并扩大研究的影响,未来的研究可以考虑以下几个方向:

深度学习技术的应用:探索更多基于深度学习的模型,如卷积神经网络(CNN)和循环神经网络(RNN)的变体,这些模型可能在处理复杂数据集时表现更为出色。

特征工程的深化:进一步研究和开发更先进的特征提取技术,如自动特征学习和特征选择算法,以更好地捕捉烟草销售数据中的潜在规律。

实时预测系统的开发:开发实时数据处理和预测系统,能够即时响应市场变化,为企业和政策制定者提供更及时的决策支持。

跨领域模型融合:探索将烟草销售预测模型与其他经济指标模型进行融合,如宏观经济模型和消费者行为模型,以实现更全面的市场分析和预测。

国际市场的分析:将研究扩展到国际市场,考虑不同国家的文化、法律和经济因素对烟草销售的影响,提升模型的普适性和适应性。

总的来说,通过这些未来研究方向的探索,本研究希望不仅能够提升烟草销售预测的科学性和准确性,还能为相关政策制定和企业战略提供更为坚实的数据支持和理论依据。

参考文献

- [1] GlobalData. China Tobacco Products Market Analysis and Forecast by Product Categories and Segments, Distribution Channel, Competitive Landscape and Consumer Segmentation, 2021-2026 [R]. 2021. 12-45.
- [2] Euromonitor. Tobacco in China | Market Research Report [R]. 2022. 20-38.
- [3] TobaccoFreeKids. The Chinese Tobacco Market and Industry Profile [R]. 2021. 5-15.
- [4] SpringerLink. The Rise of China Tobacco: From Local to Global Player [R]. 2023. 78-102.
- [5] The Examination. China Tobacco, by the numbers [R]. 2023. pp. 2-10.
- [6] 王刚. 季节性因素对中国烟草销售影响的实证分析[J]. 烟草科学与技术, 2019, 52(4): 56-62.
- [7] 乔石, 岑勇, 许丽萍, 张丽娜. 基于 ARMA 模型的烟草工业公司销售预测的研究与实践[J]. Proceedings of the 2016 2nd International Conference on Artificial Intelligence and Industrial Engineering (AIIE 2016), 2016, 546-551.
- [8] WHO. Tobacco tax increase results in decreased tobacco consumption[R]. Beijing: World Health Organization, 2016. 10-13.
- [9] Zheng R, Wang Y, Hu X, et al. Can increases in the cigarette tax rate be linked to cigarette retail prices? Solving mysteries related to the cigarette pricing mechanism in China[C]. In: Tobacco Control. Geneva: World Health Organization, 2011. 45-50.
- [10] Hu TW, Mao ZZ, Shi J, et al. Effects of cigarette tax on cigarette consumption and the Chinese economy[J]. Tobacco Control, 2010, 19(1): 58-64.
- [11] Hu TW, Zhang X, Zheng R. China has raised the tax on cigarettes: what's next?[M]. Geneva: World Health Organization, 2015. 23-25.
- [12] 前瞻产业研究院. 2020 年中国烟草行业发展现状与经营数据回顾[R]. 深圳: 前瞻产业研究院. 2020.67-68
- [13] 新浪财经. 2022 年中国烟草行业发展现状与产销规模分析[J]. 新浪财经. 2022.1-2
- [14] Euromonitor International. Tobacco in China: Market Research Report [R]. Euromonitor International, 2023. This report includes analysis of key supply-side and demand trends, detailed segmentation of products, historic volume and value sizes, and five-year forecasts of market trends and growth. Pages: 1-45.
- [15] World Health Organization (WHO). Tobacco China 2023 Country Profile [R]. WHO, 2023. This document presents data on the tobacco epidemic and selected tobacco control measures adopted in China, part of the WHO report on the global tobacco epidemic. Pages: 1-30.
- [16] 方大山, & 翁伟兵. (2011). Sales Forecasting System for Chinese Tobacco Wholesalers. Procedia Environmental Sciences, 11, 380-386.
- [17] 丁福林, & 孙利民. (2015). Prediction of Tobacco Sales Based on Support Vector Machine. Lecture Notes in Computer Science, 891-896.
- [18] 王敏菲, 童士元, 裴琪, 耿建民, 刘颖, & 孙健. (2023). Sales Forecasting Model Based

- on Ensemble Learning and Its Application in Anomaly Detection. Proceedings of the 2023 7th International Conference on Digital Signal Processing.
- [19]张勇, 钟敏儿, 耿宁, & 蒋云健. (2017). Forecasting electric vehicles sales with univariate and multivariate time series models: The case of China. PLoS ONE, 12.
- [20]Helmini, S., Jihan, N., Jayasinghe, M., & Perera, S. (2019). Sales forecasting using multivariate long short term memory network models. PeerJ Preprints.
- [21]万亮, 罗彪, 纪红梅, & 闫伟伟. (2012). A Type of Combination Forecasting Method Based on Time Series Method and PLS. American Journal of Operations Research, 2, 467-472.
- [22]郭城城, & 黄兰君. (2017). Cigarette Sales Prediction in China Based on Markov Model. DEStech Transactions on Social Science, Education and Human Science.
- [23]魏勇. (2012). Application of Time-series Decomposition with Dummy Variables to Cigarette Sales Forecast. Computer Systems and Applications.
- [24]Zhang, Yong 等. 梯度提升回归在复杂数据集上的预测性能研究[J]. 计算机科学, 2017. 44(6): 216-220.
- [25]钟天乐等. ARIMA-LSTM 混合模型在烟草销售数据分析中的应用[J]. 现代统计科学, 2021. 36(2): 34-45.
- [26]李沛昂. N-BEATS 模型在时间序列预测中的研究与应用[J]. 高等数学研究, 2022. 29(3): 78-85.
- [27]程浩, 赵新星. 基于 CatBoost 与梯度提升回归的销售预测模型比较研究[J]. 计算机与应用, 2022. 42(11): 3042-3049.
- [28]王晓东. Transformer 在时间序列预测中的应用分析[J]. 人工智能技术, 2023. 3(1): 112-120.

附录

附录 0

介绍：数据预处理和特征工程

```

=====数据预处理=====
import pandas as pd

def load_and_convert_data(filename):
    data = pd.read_excel(filename)
    data['年'] = data['月份'].astype(str).str[:4]
    data['月'] = data['月份'].astype(str).str[4:]
    data['样品代码'] = data['样品代码'].astype(str).str.split('.').str[0]
    data['名称'] = data['名称'].astype(str)
    data['销量（箱）'] = data['销量（箱）']
    data['金额（元）'] = data['金额（元）']
    data = data.apply(lambda x: round(x) if pd.notnull(x) else x).astype('Int64')
    data = data.apply(lambda x: round(x) if pd.notnull(x) else x).astype('Int64')
    data = data.drop(columns=['月份'])
    cols = ['年', '月'] + [col for col in data if col not in ['年', '月']]
    data = data[cols]
    return data

files = ['A1.xlsx', 'A2.xlsx', 'A3.xlsx', 'A4.xlsx', 'A5.xlsx']
datasets = {f: load_and_convert_data(f) for f in files}

for name, data in datasets.items():
    cleaned_filename = name.replace('.xlsx', '_1.xlsx')
    data.to_excel(cleaned_filename, index=False)

def remove_low_sales_rows(data):
    low_sales = data[(data['销量（箱）'] < 2) & (data['销量（箱）'].notna())]
    data_cleaned = data[(data['销量（箱）'] >= 2) | (data['销量（箱）'].isna())]
    return data_cleaned

for name, data in datasets.items():
    data = pd.read_excel(name.replace('.xlsx', '_1.xlsx'))
    data_cleaned = remove_low_sales_rows(data)
    filtered_filename = name.replace('.xlsx', '_2.xlsx')
    data_cleaned.to_excel(filtered_filename, index=False)

import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
import warnings

plt.rcParams['font.family'] = ['SimHei']
warnings.filterwarnings('ignore')

def standardize_and_plot(data, filename):
    scaler = StandardScaler()
    features = ["销量（箱）", "金额（元）"]
    data[features] = scaler.fit_transform(data[features])

```

```

fig, ax = plt.subplots(figsize=(7.5, 5))
data[features].plot(kind='box', ax=ax, title=f'{filename} 标准化之后的销
量和金额')
plt.savefig(f'{filename}_after_scaling.png')
plt.close(fig)
return data

for name, data in datasets.items():
    data_filtered = pd.read_excel(name.replace('.xlsx', '_2.xlsx'))
    data_standardized = standardize_and_plot(data_filtered, name.replace('.
xlsx', ''))
    standardized_filename = name.replace('.xlsx', '_3.xlsx')
    data_standardized.to_excel(standardized_filename, index=False)

=====特征工程=====
import pandas as pd

# 文件名列表
file_paths = ['A1_2.xlsx', 'A2_2.xlsx', 'A3_2.xlsx', 'A4_2.xlsx', 'A5_2.xls
x']
dataframes = [pd.read_excel(path) for path in file_paths]

enhanced_dataframes = []

for i, df in enumerate(dataframes):
    # 清洗数据: 移除任何包含 NaN 值的行
    df.dropna(subset=['销量 (箱)', '金额 (元)', '年', '月'], inplace=True)

    # 确保'年'和'月'为整型
    df['年'] = df['年'].astype(int)
    df['月'] = df['月'].astype(int)

    # 创建'年季'字段
    df['年季'] = df['年'].astype(str) + 'Q' + ((df['月
'] - 1) // 3 + 1).astype(str)

    # 创建滞后和滚动窗口特征
    for lag in range(1, 4): # 生成 1 季、2 季、3 季滞后特征
        df[f'销量滞后{lag}季'] = df['销量 (箱)'].shift(lag)
        df[f'金额滞后{lag}季'] = df['金额 (元)'].shift(lag)

    # 生成滚动窗口的平均值和标准差特征
    df['销量 3 季滚动平均'] = df['销量 (箱)'].rolling(window=3).mean()
    df['金额 3 季滚动平均'] = df['金额 (元)'].rolling(window=3).mean()
    df['销量 3 季滚动标准差'] = df['销量 (箱)'].rolling(window=3).std()
    df['金额 3 季滚动标准差'] = df['金额 (元)'].rolling(window=3).std()

    # 再次清洗, 删除因为滞后和滚动特征引入的 NaN

```

```
df.dropna(inplace=True)

# 聚合计算每年的总销量和总金额
annual_sales = df.groupby('年').agg({'销量 (箱)': 'sum', '金额 (元)': 'sum'}).rename(columns={'销量 (箱)': '年销量总和', '金额 (元)': '年金额总和'})
df = df.merge(annual_sales, on='年', how='left')

enhanced_dataframes.append(df)

# 构建保存文件名
save_filename = f'A{i+1}_4.xlsx' # 生成 A1_4, A2_4, ... 等名称
df.to_excel(save_filename, index=False)

# 输出第一个处理后的 DataFrame 以检查
print(enhanced_dataframes[0].head())

-----
import pandas as pd

# 文件名列表
file_paths = ['A1_4.xlsx', 'A2_4.xlsx', 'A3_4.xlsx', 'A4_4.xlsx', 'A5_4.xlsx']

for file_path in file_paths:
    # 读取数据
    df = pd.read_excel(file_path)

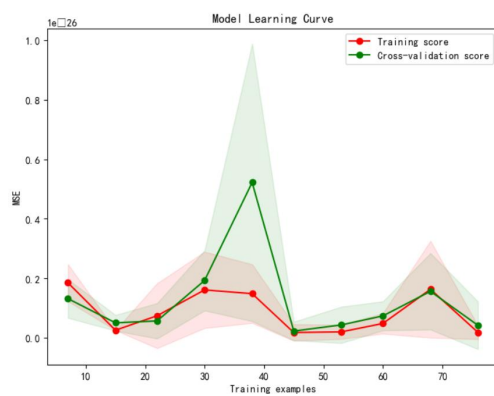
    # 删除 '样品代码' 和 '名称' 列
    if '样品代码' in df.columns and '名称' in df.columns:
        df.drop(['样品代码', '名称'], axis=1, inplace=True)

    # 保存更改到同一文件
    df.to_excel(file_path, index=False)

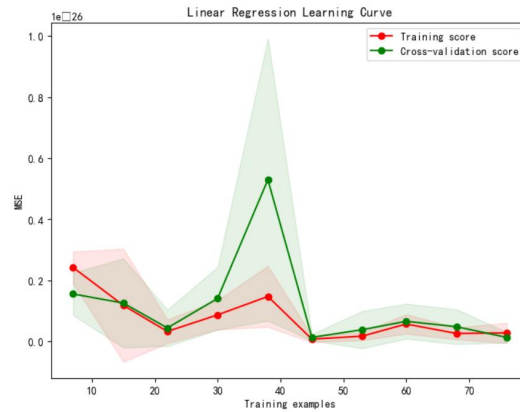
    print(f"Updated {file_path} successfully.")
```

附录 1

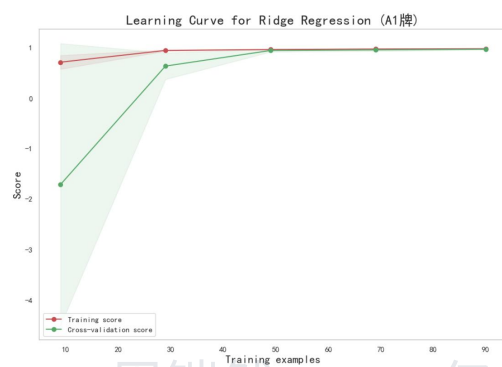
介绍：A1、A2 进行多模型训练取最优模型过程



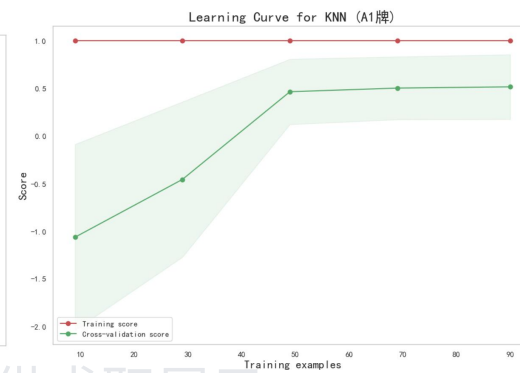
A1 多元回归训练过程



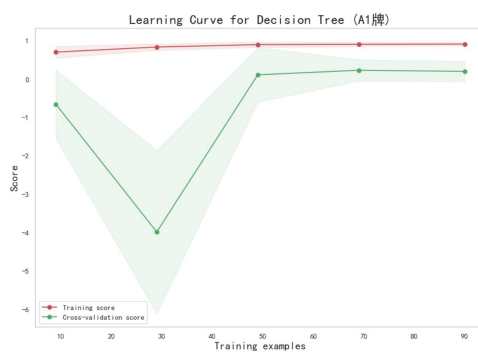
A1 线性回归训练过程



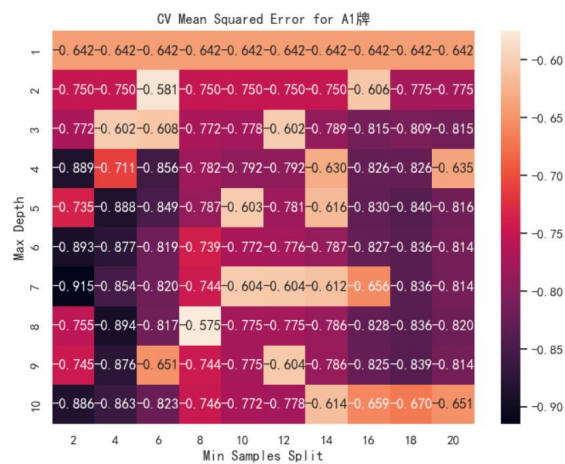
A1 岭回归训练过程



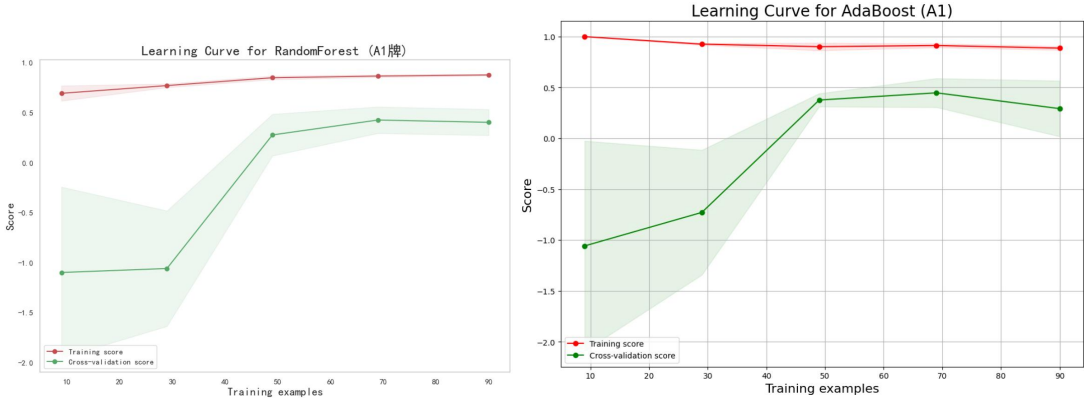
A1 KNN 训练过程



A1 决策树回归训练过程

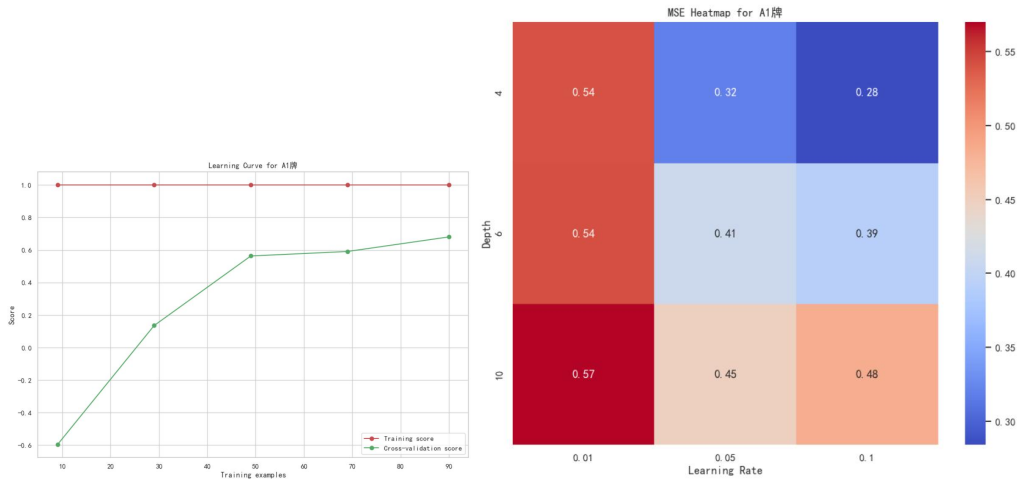


A1 决策树回归超参数组合网格搜索过程



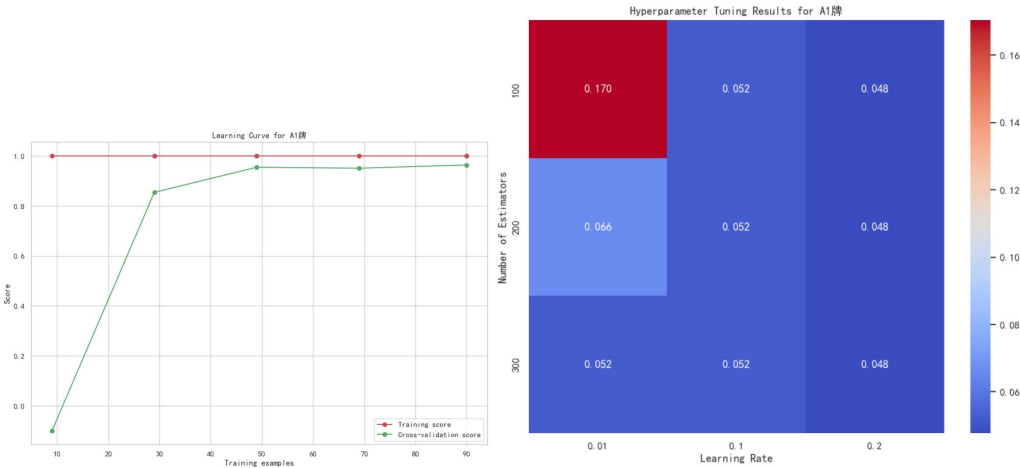
A1 随机森林训练过程

A1 AdaBoost 回归模型训练过程



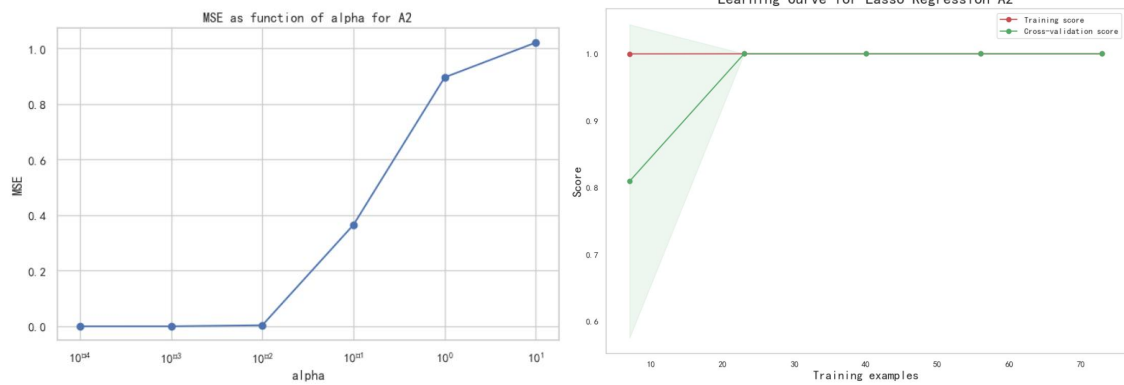
A1 CatBoost 回归训练过程

A1 CatBoost 超参数网格搜索过程

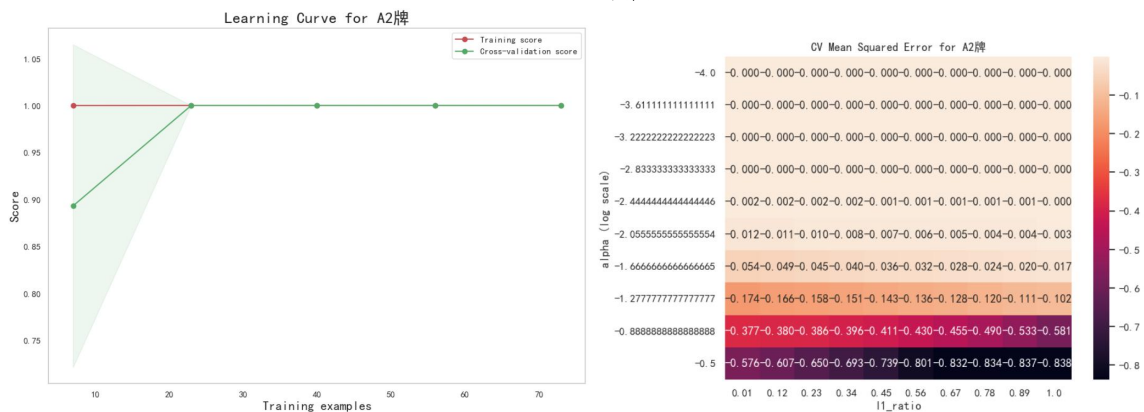


A1 GradientBoosting

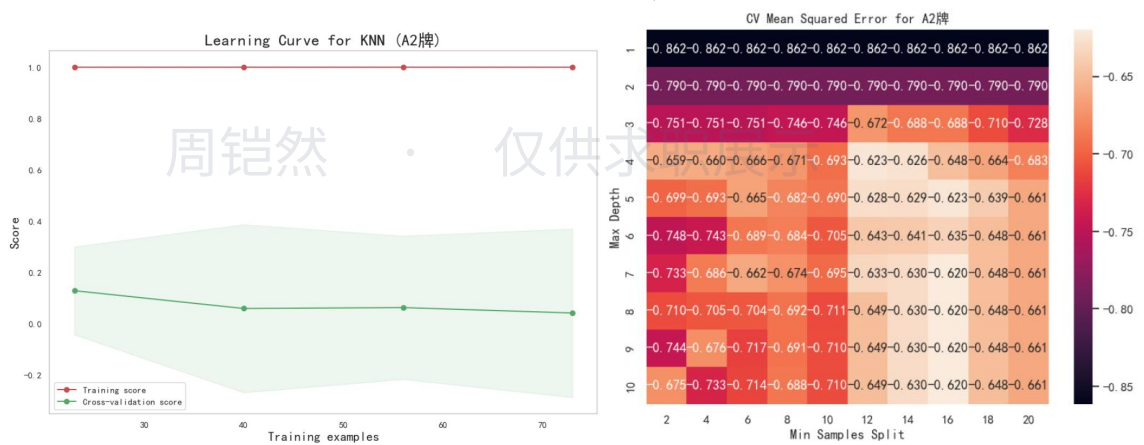
A1 GradientBoosting 超参数网格搜索过程



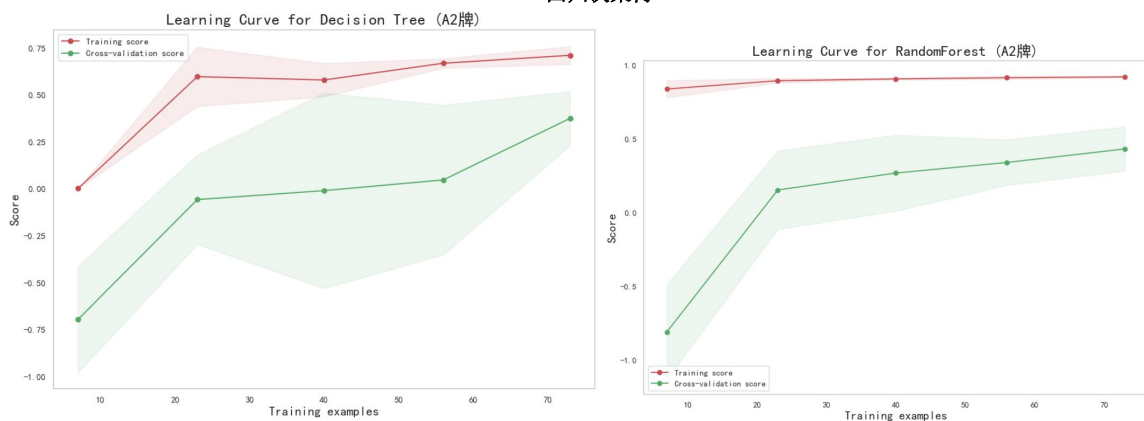
A2 Lasso 回归



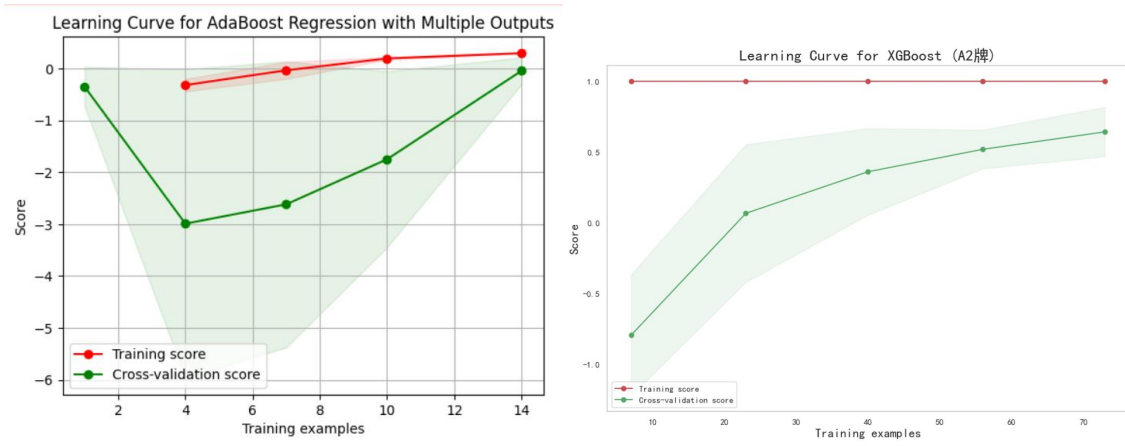
A2 ElasticNet 回归



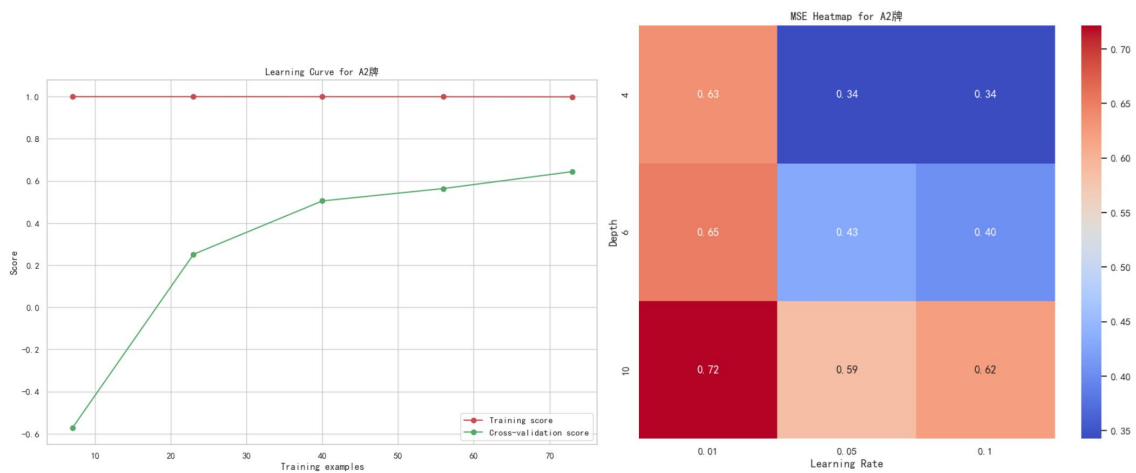
A2 回归决策树



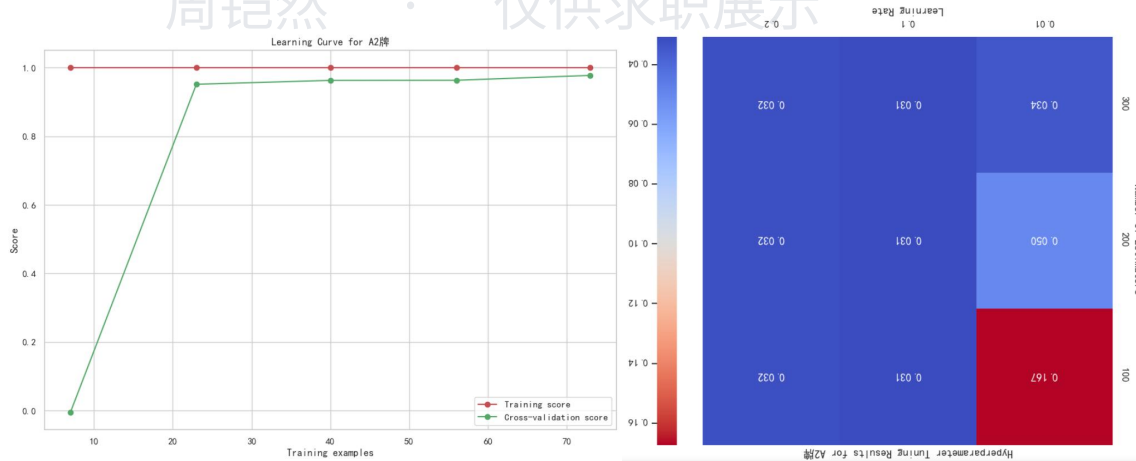
A2 随机森林



A2 AdaBoost&XGBoost



A2 CatBoost



A2 GradientBoosting

附录 1

介绍: A1、A2 模型代码

```
import pandas as pd

-----
data=pd.read_excel('A1.xlsx')
-----
data
-----
data.head(117)
-----

# 绘制各个变量之间的相关性
import matplotlib.pyplot as plt
import seaborn as sns
plt.rcParams['font.sans-serif']=['SimHei'] # 中文支持
plt.rcParams['axes.unicode_minus']=False

import matplotlib.pyplot as plt
import seaborn as sns

# 计算销量的上下四分位数
Q1 = data['销量（箱）'].quantile(0.25)
Q3 = data['销量（箱）'].quantile(0.75)
IQR = Q3 - Q1

# 定义异常值的条件
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# 找出异常值
outliers = data[(data['销量（箱）'] < lower_bound) | (data['销量（箱）'] > upper_bound)]

# 打印异常值的个数
outliers_count = outliers.shape[0]

# 可视化销量数据和异常值
plt.figure(figsize=(12, 6))
sns.boxplot(x=data['销量（箱）'])
plt.title('销量（箱）数据的异常值检测')
plt.xlabel('销量（箱）')
plt.show()

-----
outliers_count
-----

# 计算销量的均值和标准差
mean_sales = data['销量（箱）'].mean()
std_sales = data['销量（箱）'].std()

# 定义 3 倍标准差的上下界限
lower_bound = mean_sales - 3 * std_sales
upper_bound = mean_sales + 3 * std_sales
```

```
# 对数据进行截断
data['销量（箱）'] = data['销量（箱）
'].clip(lower=lower_bound, upper=upper_bound)

# 打印处理后的数据统计信息
data['销量（箱）'].describe()

-----

# 将月份转换为日期类型
data['月份'] = pd.to_datetime(data['月份'], format='%Y%m')

# 构造时序数据
n = 3
time_series_data = []

for i in range(len(data) - n):
    input_data = data['销量（箱）'].iloc[i:i + n].tolist()
    output_data = data['销量（箱）'].iloc[i + n]
    time_series_data.append(input_data + [output_data])

# 转换为 DataFrame
time_series_df = pd.DataFrame(time_series_data, columns=[f'day_{i+1}' for
i in range(n)] + ['target_day'])
time_series_df

-----

time_series_df=time_series_df.dropna()
time_series_df
-----

Y=time_series_df['target_day'].values
X=time_series_df[time_series_df.columns[:-1]].values
-----

-----

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import r2_score, mean_absolute_percentage_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# 数据预处理：标准化
scaler = MinMaxScaler(feature_range=(0, 1))
X_scaled = scaler.fit_transform(X)
Y_scaled = scaler.fit_transform(Y.reshape(-1, 1))

# 划分训练集和测试集
train_size = len(time_series_df) - 20
X_train, X_test = X_scaled[:train_size], X_scaled[train_size:]
Y_train, Y_test = Y_scaled[:train_size], Y_scaled[train_size:]

# 将数据转换为 LSTM 输入格式 [samples, time steps, features]
```

```
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

# 构建 LSTM 模型
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(X_train.shape[1], 1
)))
model.add(LSTM(50))
model.add(Dense(1))

# 编译模型
model.compile(optimizer='adam', loss='mean_squared_error')

# 训练模型
model.fit(X_train, Y_train, epochs=50, batch_size=1, verbose=2)

-----

# 预测训练集和测试集
train_predictions_scaled = model.predict(X_train)
test_predictions_scaled = model.predict(X_test)

# 逆转换预测结果
train_predictions = scaler.inverse_transform(train_predictions_scaled)
test_predictions = scaler.inverse_transform(test_predictions_scaled)
Y_train_actual = scaler.inverse_transform(Y_train.reshape(-1, 1))
Y_test_actual = scaler.inverse_transform(Y_test.reshape(-1, 1))

# 计算 R2和 MAPE
train_r2 = r2_score(Y_train_actual, train_predictions)
test_r2 = r2_score(Y_test_actual, test_predictions)
train_mape = mean_absolute_percentage_error(Y_train_actual, train_predicti
ons)
test_mape = mean_absolute_percentage_error(Y_test_actual, test_predictions
)

# 输出训练数据对比图
plt.figure(figsize=(14, 6))
plt.plot(Y_train_actual, label='Actual')
plt.plot(train_predictions, label='Predicted')
plt.title('Train Data: Actual vs Predicted')
plt.xlabel('Samples')
plt.ylabel('销量 (箱) ')
plt.legend()
plt.show()

-----

# 输出测试数据对比图
plt.figure(figsize=(14, 6))
plt.plot(Y_test_actual, label='Actual')
plt.plot(test_predictions, label='Predicted')
plt.title('Test Data: Actual vs Predicted')
plt.xlabel('Samples')
plt.ylabel('销量 (箱) ')
```

```
plt.legend()
plt.show()

-----
# 打印 R²和 MAPE
print(f'Train R²: {train_r2:.4f}')
print(f'Train MAPE: {train_mape:.4f}')
print(f'Test R²: {test_r2:.4f}')
print(f'Test MAPE: {test_mape:.4f}')
-----

# 数据预处理：标准化
scaler_X = MinMaxScaler(feature_range=(0, 1))
scaler_Y = MinMaxScaler(feature_range=(0, 1))

X_scaled = scaler_X.fit_transform(X)
Y_scaled = scaler_Y.fit_transform(Y.reshape(-1, 1))

# 将数据转换为 LSTM 输入格式 [samples, time steps, features]
X_scaled = np.reshape(X_scaled, (X_scaled.shape[0], X_scaled.shape[1], 1))

# 构建 LSTM 模型
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(X_scaled.shape[1],
1)))
model.add(LSTM(50))
model.add(Dense(1))

# 编译模型
model.compile(optimizer='adam', loss='mean_squared_error')

# 训练模型
model.fit(X_scaled, Y_scaled, epochs=50, batch_size=1, verbose=2)

# 进行未来 20 个数据点的预测
future_predictions = []
last_n_steps = Y_scaled[-n:] # 获取最后一个数据点
for _ in range(20):
    prediction = model.predict(last_n_steps.reshape(1, n, 1))
    future_predictions.append(prediction[0, 0])
    last_n_steps = np.roll(last_n_steps, -1)
    last_n_steps[-1] = prediction

# 逆转换预测结果
future_predictions = scaler_Y.inverse_transform(np.array(future_predictions).reshape(-1, 1))
future_predictions

-----
pd.DataFrame(future_predictions)
-----

# 可视化结果
plt.figure(figsize=(14, 6))
```

```
plt.plot(np.arange(len(Y)), Y, label='Actual Data')
plt.plot(np.arange(len(Y), len(Y) + 20), future_predictions, label='Predicted Future Data')
plt.title('Actual Data and Future Predictions')
plt.xlabel('Samples')
plt.ylabel('销量 (箱)')
plt.legend()
plt.show()

-----

-----

import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import r2_score, mean_absolute_percentage_error
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor, GradientBoostingRegressor, ExtraTreesRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.svm import SVR
from xgboost import XGBRegressor
from sklearn.linear_model import SGDRegressor
import matplotlib.pyplot as plt

# 数据预处理：标准化
scaler = MinMaxScaler(feature_range=(0, 1))
X_scaled = scaler.fit_transform(X)
Y_scaled = scaler.fit_transform(Y.reshape(-1, 1)).flatten()

# 划分训练集和测试集
X_train, X_test, Y_train, Y_test = train_test_split(X_scaled, Y_scaled, test_size=20, shuffle=False)

-----

# 定义所有模型
models = {
    "决策树": DecisionTreeRegressor(),
    "随机森林": RandomForestRegressor(),
    "AdaBoost": AdaBoostRegressor(),
    "梯度提升树": GradientBoostingRegressor(),
    "Extra Trees": ExtraTreesRegressor(),
    "K 近邻": KNeighborsRegressor(),
    "BP 神经网络": MLPRegressor(max_iter=500),
    "支持向量机": SVR(),
    "XGBoost": XGBRegressor(objective='reg:squarederror'),
    "线性回归 (SGD)": SGDRegressor(max_iter=1000, tol=1e-3)
}
```

```
results = {}

# 训练并评估所有模型
for name, model in models.items():
    model.fit(X_train, Y_train)
    predictions = model.predict(X_test)
    r2 = r2_score(Y_test, predictions)
    mape = mean_absolute_percentage_error(Y_test, predictions)
    results[name] = {'R2': r2, 'MAPE': mape}
    print(f'{name} - R2: {r2:.4f}, MAPE: {mape:.4f}')

# 绘制预测结果对比图
plt.figure(figsize=(14, 6))
plt.plot(scaler.inverse_transform(Y_test.reshape(-1, 1)), label='实际值')

plt.plot(scaler.inverse_transform(predictions.reshape(-1, 1)), label='预测值')

plt.title(f'测试数据: 实际值 vs 预测值 ({name})')
plt.xlabel('样本')
plt.ylabel('销量 (箱)')
plt.legend()
plt.show()

-----
周铠然， 仅供求职展示
-----

# 输出所有模型的评估结果
results_df = pd.DataFrame(results).T
results_df

-----

# 绘制 MAPE 柱状图并按降序排序
results_df = results_df.sort_values(by='MAPE', ascending=False)

plt.figure(figsize=(12, 6))
plt.bar(results_df.index, results_df['MAPE'], color='skyblue')
plt.xlabel('模型')
plt.ylabel('MAPE (%)')
plt.title('各模型的 MAPE 对比')
plt.xticks(rotation=45)
plt.show()

-----

# 绘制测试集评估结果对比图
plt.figure(figsize=(14, 6))
for name, model in models.items():
    predictions = model.predict(X_test)
    plt.plot(scaler.inverse_transform(predictions.reshape(-1, 1)), label=f'{name}预测值')

plt.plot(scaler.inverse_transform(Y_test.reshape(-1, 1)), label='实际值')
```

```

', color='black', linewidth=2)
plt.title('测试数据: 实际值 vs 预测值 (所有模型)')
plt.xlabel('样本')
plt.ylabel('销量 (箱)')
plt.legend()
plt.show()

-----

Y_train
-----

mean_absolute_percentage_error(train_predictions, Y_train)
-----

# 训练集评估结果存储
train_results = {}

# 训练并评估所有模型 (训练集)
for name, model in models.items():
    model.fit(X_train, Y_train)
    train_predictions = model.predict(X_train)
    train_r2 = r2_score(Y_train, train_predictions)
    train_mape = mean_absolute_percentage_error(train_predictions, Y_train)

    train_results[name] = {'R²': train_r2, 'MAPE': train_mape}
    print(f'{name} (训练集) - R²: {train_r2:.4f}, MAPE: {train_mape:.4f}')

# 绘制训练集预测结果对比图
plt.figure(figsize=(14, 6))
plt.plot(scaler.inverse_transform(Y_train.reshape(-1, 1)), label='实际值')
plt.plot(scaler.inverse_transform(train_predictions.reshape(-1, 1)), label='预测值')
plt.title(f'训练数据: 实际值 vs 预测值 ({name})')
plt.xlabel('样本')
plt.ylabel('销量 (箱)')
plt.legend()
plt.show()

-----

# 输出训练集所有模型的评估结果
train_results_df = pd.DataFrame(train_results).T

train_results_df

-----

# 绘制训练集 MAPE 柱状图并按降序排序
train_results_df = train_results_df.sort_values(by='MAPE', ascending=False)

plt.figure(figsize=(12, 6))
plt.bar(train_results_df.index, train_results_df['MAPE'], color='lightcoral')
plt.xlabel('模型')

```

```
plt.ylabel('MAPE')
plt.title('各模型的训练集 MAPE 对比')
plt.xticks(rotation=45)
plt.show()

-----
X_scaled
-----

# 选择决策树模型
model = DecisionTreeRegressor()

# 使用全部数据进行训练
model.fit(X_scaled, Y_scaled)

-----
Y_scaled[-n:]
-----

# 向后预测 20 个
future_predictions_scaled = []
last_n_days = Y_scaled[-n:] # 选择最后 n 天的数据作为初始输入

for _ in range(20):
    pred_scaled = model.predict(last_n_days.reshape(1, -1))
    future_predictions_scaled.append(pred_scaled[0])

    # 更新输入, 滑动窗口
    last_n_days = np.append(last_n_days[1:], pred_scaled).reshape(-1, 1)

# 逆转换预测结果
future_predictions = scaler.inverse_transform(np.array(future_predictions_scaled).reshape(-1, 1))
future_predictions

-----
pd.DataFrame(future_predictions)
-----

# 绘制预测结果对比图
plt.figure(figsize=(14, 6))
plt.plot(range(len(Y)), Y, label='实际值')
plt.plot(range(len(Y), len(Y) + 20), future_predictions, label='预测值', color='red')
plt.title('未来 20 天的销量预测')
plt.xlabel('样本')
plt.ylabel('销量 (箱)')
plt.legend()
plt.show()
```

附录 2**介绍：A3、A4 香烟品牌的销售金额进行数据预测**

```
import pandas as pd
data=pd.read_excel('d:/jupyterproject/钉钉杯/A/A3.xlsx')
data
-----
data.head(99)
-----
# 绘制各个变量之间的相关性
import matplotlib.pyplot as plt
import seaborn as sns
plt.rcParams['font.sans-serif']=['SimHei'] # 中文支持
plt.rcParams['axes.unicode_minus']=False

import matplotlib.pyplot as plt
import seaborn as sns

# 计算销量的上下四分位数
Q1 = data['金额（元）'].quantile(0.25)
Q3 = data['金额（元）'].quantile(0.75)
IQR = Q3 - Q1

# 定义异常值的条件
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# 找出异常值
outliers = data[(data['金额（元）'] < lower_bound) | (data['金额（元）'] > upper_bound)]

# 打印异常值的个数
outliers_count = outliers.shape[0]

# 可视化销量数据和异常值
plt.figure(figsize=(12, 6))
sns.boxplot(x=data['金额（元）'])
plt.title('金额（元）数据的异常值检测')
plt.xlabel('金额（元）')
plt.show()
-----
outliers_count
-----
# 计算销量的均值和标准差
mean_sales = data['金额（元）'].mean()
std_sales = data['金额（元）'].std()

# 定义 3 倍标准差的上下界限
lower_bound = mean_sales - 3 * std_sales
upper_bound = mean_sales + 3 * std_sales

# 对数据进行截断
data['金额（元）'] = data['金额（元）'].clip(lower=lower_bound, upper=upper_bound)
```

```
# 打印处理后的数据统计信息
data['金额（元）'].describe()

-----

# 将月份转换为日期类型
data['月份'] = pd.to_datetime(data['月份'], format='%Y%m')

# 构造时序数据
n = 3
time_series_data = []

for i in range(len(data) - n):
    input_data = data['金额（元）'].iloc[i:i + n].tolist()
    output_data = data['金额（元）'].iloc[i + n]
    time_series_data.append(input_data + [output_data])

# 转换为 DataFrame
time_series_df = pd.DataFrame(time_series_data, columns=[f'day_{i+1}' for i
in range(n)] + ['target_day'])
time_series_df

-----

time_series_df=time_series_df.dropna()
time_series_df

-----

Y=time_series_df['target_day'].values
X=time_series_df[time_series_df.columns[:-1]].values

-----

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import r2_score, mean_absolute_percentage_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# 数据预处理：标准化
scaler = MinMaxScaler(feature_range=(0, 1))
X_scaled = scaler.fit_transform(X)
Y_scaled = scaler.fit_transform(Y.reshape(-1, 1))

# 划分训练集和测试集
train_size = len(time_series_df) - 20
X_train, X_test = X_scaled[:train_size], X_scaled[train_size:]
Y_train, Y_test = Y_scaled[:train_size], Y_scaled[train_size:]

# 将数据转换为 LSTM 输入格式 [samples, time steps, features]
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

# 构建 LSTM 模型
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(X_train.shape[1], 1)
)))
model.add(LSTM(50))
```

```
model.add(Dense(1))

# 编译模型
model.compile(optimizer='adam', loss='mean_squared_error')

# 训练模型
model.fit(X_train, Y_train, epochs=50, batch_size=1, verbose=2)

-----

# 预测训练集和测试集
train_predictions_scaled = model.predict(X_train)
test_predictions_scaled = model.predict(X_test)

# 逆转换预测结果
train_predictions = scaler.inverse_transform(train_predictions_scaled)
test_predictions = scaler.inverse_transform(test_predictions_scaled)
Y_train_actual = scaler.inverse_transform(Y_train.reshape(-1, 1))
Y_test_actual = scaler.inverse_transform(Y_test.reshape(-1, 1))

# 计算 R2和 MAPE
train_r2 = r2_score(Y_train_actual, train_predictions)
test_r2 = r2_score(Y_test_actual, test_predictions)
train_mape = mean_absolute_percentage_error(Y_train_actual, train_predictions)
ns) test_mape = mean_absolute_percentage_error(Y_test_actual, test_predictions)

# 输出训练数据对比图
plt.figure(figsize=(14, 6))
plt.plot(Y_train_actual, label='Actual')
plt.plot(train_predictions, label='Predicted')
plt.title('Train Data: Actual vs Predicted')
plt.xlabel('Samples')
plt.ylabel('金额 (元)')
plt.legend()
plt.show()

-----

# 输出测试数据对比图
plt.figure(figsize=(14, 6))
plt.plot(Y_test_actual, label='Actual')
plt.plot(test_predictions, label='Predicted')
plt.title('Test Data: Actual vs Predicted')
plt.xlabel('Samples')
plt.ylabel('金额 (元)')
plt.legend()
plt.show()

-----

# 打印 R2和 MAPE
print(f'Train R2: {train_r2:.4f}')
print(f'Train MAPE: {train_mape:.4f}')
print(f'Test R2: {test_r2:.4f}')
```

```
print(f'Test MAPE: {test_mape:.4f}')
-----

# 数据预处理：标准化
scaler_X = MinMaxScaler(feature_range=(0, 1))
scaler_Y = MinMaxScaler(feature_range=(0, 1))

X_scaled = scaler_X.fit_transform(X)
Y_scaled = scaler_Y.fit_transform(Y.reshape(-1, 1))

# 将数据转换为 LSTM 输入格式 [samples, time steps, features]
X_scaled = np.reshape(X_scaled, (X_scaled.shape[0], X_scaled.shape[1], 1))

# 构建 LSTM 模型
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(X_scaled.shape[1],
1)))
model.add(LSTM(50))
model.add(Dense(1))

# 编译模型
model.compile(optimizer='adam', loss='mean_squared_error')

# 训练模型
model.fit(X_scaled, Y_scaled, epochs=50, batch_size=1, verbose=2)

# 进行未来 20 个数据点的预测
future_predictions = []
last_n_steps = Y_scaled[-n:] # 获取最后一个数据点
for _ in range(20):
    prediction = model.predict(last_n_steps.reshape(1, n, 1))
    future_predictions.append(prediction[0, 0])
    last_n_steps = np.roll(last_n_steps, -1)
    last_n_steps[-1] = prediction

# 逆转换预测结果
future_predictions = scaler_Y.inverse_transform(np.array(future_predictions
).reshape(-1, 1))
future_predictions

-----

pd.DataFrame(future_predictions)
-----

# 可视化结果
plt.figure(figsize=(14, 6))
plt.plot(np.arange(len(Y)), Y, label='Actual Data')
plt.plot(np.arange(len(Y), len(Y) + 20), future_predictions, label='Predict
ed Future Data')
plt.title('Actual Data and Future Predictions')
plt.xlabel('Samples')
plt.ylabel('金额（元）')
plt.legend()
plt.show()

-----

import numpy as np
```

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import r2_score, mean_absolute_percentage_error
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor, GradientBoostingRegressor, ExtraTreesRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.svm import SVR
from xgboost import XGBRegressor
from sklearn.linear_model import SGDRegressor
import matplotlib.pyplot as plt

# 数据预处理：标准化
scaler = MinMaxScaler(feature_range=(0, 1))
X_scaled = scaler.fit_transform(X)
Y_scaled = scaler.fit_transform(Y.reshape(-1, 1)).flatten()

# 划分训练集和测试集
X_train, X_test, Y_train, Y_test = train_test_split(X_scaled, Y_scaled, test_size=20, shuffle=False)

-----

# 定义所有模型
models = {
    "决策树": DecisionTreeRegressor(),
    "随机森林": RandomForestRegressor(),
    "AdaBoost": AdaBoostRegressor(),
    "梯度提升树": GradientBoostingRegressor(),
    "Extra Trees": ExtraTreesRegressor(),
    "K 近邻": KNeighborsRegressor(),
    "BP 神经网络": MLPRegressor(max_iter=500),
    "支持向量机": SVR(),
    "XGBoost": XGBRegressor(objective='reg:squarederror'),
    "线性回归 (SGD)": SGDRegressor(max_iter=1000, tol=1e-3)
}

results = {}

# 训练并评估所有模型
for name, model in models.items():
    model.fit(X_train, Y_train)
    predictions = model.predict(X_test)
    r2 = r2_score(Y_test, predictions)
    mape = mean_absolute_percentage_error(Y_test, predictions)
    results[name] = {'R²': r2, 'MAPE': mape}
print(f'{name} - R²: {r2:.4f}, MAPE: {mape:.4f}')
```

```
# 绘制预测结果对比图
plt.figure(figsize=(14, 6))
plt.plot(scaler.inverse_transform(Y_test.reshape(-1, 1)), label='实际值')
plt.plot(scaler.inverse_transform(predictions.reshape(-1, 1)), label='预测值')
plt.title(f'测试数据: 实际值 vs 预测值 ({name})')
plt.xlabel('样本')
plt.ylabel('金额 (元)')
plt.legend()
plt.show()

-----

# 输出所有模型的评估结果
results_df = pd.DataFrame(results).T
results_df

-----

# 绘制 MAPE 柱状图并按降序排序
results_df = results_df.sort_values(by='MAPE', ascending=False)

plt.figure(figsize=(12, 6))
plt.bar(results_df.index, results_df['MAPE'], color='skyblue')
plt.xlabel('模型')
plt.ylabel('MAPE (%)')
plt.title('各模型的 MAPE 对比')
plt.xticks(rotation=45)
plt.show()

-----

# 绘制测试集评估结果对比图
plt.figure(figsize=(14, 6))
for name, model in models.items():
    predictions = model.predict(X_test)
    plt.plot(scaler.inverse_transform(predictions.reshape(-1, 1)), label=f'{name}预测值')

    plt.plot(scaler.inverse_transform(Y_test.reshape(-1, 1)), label='实际值', color='black', linewidth=2)
plt.title('测试数据: 实际值 vs 预测值 (所有模型)')
plt.xlabel('样本')
plt.ylabel('金额 (元)')
plt.legend()
plt.show()

-----

# 训练集评估结果存储
train_results = {}

# 训练并评估所有模型 (训练集)
for name, model in models.items():
    model.fit(X_train, Y_train)
    train_predictions = model.predict(X_train)
    train_r2 = r2_score(Y_train, train_predictions)
```

```
train_mape = mean_absolute_percentage_error(train_predictions, Y_train)
train_results[name] = {'R2': train_r2, 'MAPE': train_mape}
print(f'{name} (训练集) - R2: {train_r2:.4f}, MAPE: {train_mape:.4f}')

# 绘制训练集预测结果对比图
plt.figure(figsize=(14, 6))
plt.plot(scaler.inverse_transform(Y_train.reshape(-1, 1)), label='实际
值')
plt.plot(scaler.inverse_transform(train_predictions.reshape(-1, 1)), label='预测值')
plt.title(f'训练数据: 实际值 vs 预测值 ({name})')
plt.xlabel('样本')
plt.ylabel('金额 (元)')
plt.legend()
plt.show()

-----

# 输出训练集所有模型的评估结果
train_results_df = pd.DataFrame(train_results).T

train_results_df

-----

# 绘制训练集 MAPE 柱状图并按降序排序
train_results_df = train_results_df.sort_values(by='MAPE', ascending=False)

plt.figure(figsize=(12, 6))
plt.bar(train_results_df.index, train_results_df['MAPE'], color='lightcoral')
plt.xlabel('模型')
plt.ylabel('MAPE')
plt.title('各模型的训练集 MAPE 对比')
plt.xticks(rotation=45)
plt.show()

-----

# 选择决策树模型
model = DecisionTreeRegressor()

# 使用全部数据进行训练
model.fit(X_scaled, Y_scaled)

-----

Y_scaled[-n:]
-----

# 向后预测 20 个
future_predictions_scaled = []
last_n_days = Y_scaled[-n:] # 选择最后 n 天的数据作为初始输入

for _ in range(20):
    pred_scaled = model.predict(last_n_days.reshape(1, -1))
    future_predictions_scaled.append(pred_scaled[0])
```

```

# 更新输入，滑动窗口
last_n_days = np.append(last_n_days[1:], pred_scaled).reshape(-1, 1)

# 逆转换预测结果
future_predictions = scaler.inverse_transform(np.array(future_predictions_scaled).reshape(-1, 1))
future_predictions

-----
pd.DataFrame(future_predictions)
-----

# 绘制预测结果对比图
plt.figure(figsize=(14, 6))
plt.plot(range(len(Y)), Y, label='实际值')
plt.plot(range(len(Y), len(Y) + 20), future_predictions, label='预测值', color='red')
plt.title('未来 20 天的销量预测')
plt.xlabel('样本')
plt.ylabel('金额（元）')
plt.legend()
plt.show()

```

附录 3

介绍：集成学习模型构建代码

```

import pandas as pd

# 读取 A5 数据集
data = pd.read_csv('data.csv')
data

-----

# 删除'样品代码'和'名称'列
data = data.drop(columns=['样品代码', '名称'])
-----

pd.options.display.float_format = '{:.2f}'.format

data

-----

from sklearn.preprocessing import StandardScaler
# 初始化标准化器
scaler = StandardScaler()

# 选择需要标准化的列
columns_to_standardize = ['销量（箱）', '金额（元）', '每年总销量（箱）', '平均每月销量（箱）', '每年总金额（元）', '平均每月金额（元）', '季度总销量（箱）', '季度总金额（元）']

```

```
# 进行标准化处理
data[columns_to_standardize] = scaler.fit_transform(data[columns_to_standardize])

# 获取标准化过程中的均值和标准差
means = scaler.mean_
scales = scaler.scale_

-----
means
-----
scales
-----
data
-----

import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import TimeSeriesSplit, learning_curve
import matplotlib.pyplot as plt

# 假设 data 是已经加载并预处理过的 DataFrame
# 特征和目标变量
X = data[['年份', '季度', '每年总销量（箱）', '平均每月销量（箱）', '每年总金额（元）', '平均每月金额（元）', '季度总销量（箱）', '季度总金额（元）']]
y = data[['销量（箱）', '金额（元）']]

# 保持时间顺序，将最后 11 步作为测试集
n_test = 11
X_train, X_test = X[:-n_test], X[-n_test:]
y_train, y_test = y[:-n_test], y[-n_test:]

# 初始化多元线性回归模型
model = LinearRegression()

# 训练模型
model.fit(X_train, y_train)

# 预测训练集和测试集
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

# 评估模型性能
def evaluate_model(y_true, y_pred):
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    mae = mean_absolute_error(y_true, y_pred)
    r_squared = r2_score(y_true, y_pred)
    return {'RMSE': rmse, 'MAE': mae, 'R-squared': r_squared}

train_metrics = evaluate_model(y_train, y_train_pred)
test_metrics = evaluate_model(y_test, y_test_pred)

print("训练集评估指标:", train_metrics)
print("\n测试集评估指标:", test_metrics)
```

```

# 使用 5 折交叉验证进行评估
tscv = TimeSeriesSplit(n_splits=5)
train_sizes, train_scores, test_scores = learning_curve(model, X_train, y_
train, cv=tscv,
scoring='r2', n_jo
bs=-1,
train_sizes=np.lin
space(0.1, 1.0, 10))

# 计算得分的平均值和标准差
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)

plt.figure()
plt.title('Learning Curve for Linear Regression with Multiple Outputs')
plt.xlabel('Training examples')
plt.ylabel('Score')
plt.grid()

plt.fill_between(train_sizes, train_scores_mean - train_scores_std, train_
scores_mean + train_scores_std, alpha=0.1, color='r')
plt.fill_between(train_sizes, test_scores_mean - test_scores_std, test_sco
res_mean + test_scores_std, alpha=0.1, color='g')
plt.plot(train_sizes, train_scores_mean, 'o-', color='r', label='Training
score')
plt.plot(train_sizes, test_scores_mean, 'o-', color='g', label='Cross-vali
dation score')

plt.legend(loc='best')
plt.show()

-----
import pandas as pd
import numpy as np
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt

# 假设 data 是已经加载并预处理过的 DataFrame
# 特征和目标变量
X = data[['年份', '季度', '每年总销量 (箱)', '平均每月销量 (箱)', '每年总金额
(元)', '平均每月金额 (元)', '季度总销量 (箱)', '季度总金额 (元)']]
y = data[['销量 (箱)', '金额 (元)']]

# 保持时间顺序, 将最后 11 步作为测试集
n_test = 11
X_train, X_test = X[:-n_test], X[-n_test:]
y_train, y_test = y[:-n_test], y[-n_test:]

# 设置超参数搜索范围
param_grid = {
    'alpha': np.logspace(-3, 3, 7) # 范围从 0.001 到 1000
}

```

```
# 初始化岭回归模型
ridge = Ridge()

# 使用 GridSearchCV 进行超参数搜索
grid_search = GridSearchCV(ridge, param_grid, cv=5, scoring='neg_mean_squared_error', return_train_score=True)

# 拟合 GridSearchCV
grid_search.fit(X_train, y_train)

# 输出最佳参数
print("Best parameters:", grid_search.best_params_)
print("Best cross-validation score: {:.2f}".format(-grid_search.best_score_))

# 绘制结果
plt.figure(figsize=(10, 5))
plt.title("Ridge Regression: Hyperparameter Tuning")
plt.xlabel("Alpha")
plt.ylabel("Negative Mean Squared Error")
plt.semilogx(grid_search.param_grid['alpha'], -grid_search.cv_results_['mean_test_score'], label='Validation Score')
plt.legend()
plt.grid(True)
plt.show()

-----
import pandas as pd
import numpy as np
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import TimeSeriesSplit, learning_curve
import matplotlib.pyplot as plt

# 使用超参数搜索结果
best_alpha = grid_search.best_params_['alpha']
model = Ridge(alpha=best_alpha)

# 训练模型
model.fit(X_train, y_train)

# 预测训练集和测试集
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

# 评估模型性能
def evaluate_model(y_true, y_pred):
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    mae = mean_absolute_error(y_true, y_pred)
    r_squared = r2_score(y_true, y_pred)
    return {'RMSE': rmse, 'MAE': mae, 'R-squared': r_squared}

train_metrics = evaluate_model(y_train, y_train_pred)
```

```
test_metrics = evaluate_model(y_test, y_test_pred)

print("训练集评估指标:", train_metrics)
print("\n 测试集评估指标:", test_metrics)

# 使用 5 折交叉验证进行评估
tscv = TimeSeriesSplit(n_splits=6)
train_sizes, train_scores, test_scores = learning_curve(model, X_train, y_train, cv=tscv,
                                                         scoring='r2', n_jobs=-1,
                                                         train_sizes=np.linspace(0.1, 1.0, 10))

# 计算得分的平均值和标准差
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)

plt.figure()
plt.title('Learning Curve for Ridge Regression with Multiple Outputs')
plt.xlabel('Training examples')
plt.ylabel('Score')
plt.grid()

plt.fill_between(train_sizes, train_scores_mean - train_scores_std, train_scores_mean + train_scores_std, alpha=0.1, color='r')
plt.fill_between(train_sizes, test_scores_mean - test_scores_std, test_scores_mean + test_scores_std, alpha=0.1, color='g')
plt.plot(train_sizes, train_scores_mean, 'o-', color='r', label='Training score')
plt.plot(train_sizes, test_scores_mean, 'o-', color='g', label='Cross-validation score')

plt.legend(loc='best')
plt.show()

-----
import pandas as pd
import numpy as np
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt

# 假设 data 是已经加载并预处理过的 DataFrame
# 特征和目标变量
X = data[['年份', '季度', '每年总销量（箱）', '平均每月销量（箱）', '每年总金额（元）', '平均每月金额（元）', '季度总销量（箱）', '季度总金额（元）']]
y = data[['销量（箱）', '金额（元）']]

# 保持时间顺序，将最后 11 步作为测试集
n_test = 11
X_train, X_test = X[:-n_test], X[-n_test:]
y_train, y_test = y[:-n_test], y[-n_test:]
```

```
# 设置超参数搜索范围
param_grid = {
    'alpha': np.logspace(-3, 2, 6) # 从 0.001 到 100 调整范围和数量
}

# 初始化 LASSO 回归模型
lasso = Lasso()

# 使用 GridSearchCV 进行超参数搜索
grid_search = GridSearchCV(lasso, param_grid, cv=5, scoring='neg_mean_squared_error', return_train_score=True)

# 拟合 GridSearchCV
grid_search.fit(X_train, y_train)

# 输出最佳参数
print("Best parameters:", grid_search.best_params_)
print("Best cross-validation score: {:.2f}".format(-grid_search.best_score_))

# 绘制结果
plt.figure(figsize=(10, 5))
plt.title("LASSO Regression: Hyperparameter Tuning")
plt.xlabel("Alpha")
plt.ylabel("Negative Mean Squared Error")
plt.semilogx(grid_search.param_grid['alpha'], -grid_search.cv_results_['mean_test_score'], label='Validation Score')
plt.legend()
plt.grid(True)
plt.show()

-----
import pandas as pd
import numpy as np
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import TimeSeriesSplit, learning_curve
import matplotlib.pyplot as plt

# 使用超参数搜索结果
best_alpha = grid_search.best_params_['alpha']
model = Lasso(alpha=best_alpha)

# 训练模型
model.fit(X_train, y_train)

# 预测训练集和测试集
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

# 评估模型性能
def evaluate_model(y_true, y_pred):
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
```

```

    mae = mean_absolute_error(y_true, y_pred)
    r_squared = r2_score(y_true, y_pred)
    return {'RMSE': rmse, 'MAE': mae, 'R-squared': r_squared}

train_metrics = evaluate_model(y_train, y_train_pred)
test_metrics = evaluate_model(y_test, y_test_pred)

print("训练集评估指标:", train_metrics)
print("\n 测试集评估指标:", test_metrics)

# 使用 5 折交叉验证进行评估
tscv = TimeSeriesSplit(n_splits=6)
train_sizes, train_scores, test_scores = learning_curve(model, X_train, y_
train, cv=tscv,
scoring='r2', n_jo
bs=-1,
train_sizes=np.lin
space(0.1, 1.0, 10))

# 计算得分的平均值和标准差
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)

plt.figure()
plt.title('Learning Curve for LASSO Regression with Multiple Outputs')
plt.xlabel('Training examples')
plt.ylabel('Score')
plt.grid()

plt.fill_between(train_sizes, train_scores_mean - train_scores_std, train_
scores_mean + train_scores_std, alpha=0.1, color='r')
plt.fill_between(train_sizes, test_scores_mean - test_scores_std, test_sco
res_mean + test_scores_std, alpha=0.1, color='g')
plt.plot(train_sizes, train_scores_mean, 'o-', color='r', label='Training
score')
plt.plot(train_sizes, test_scores_mean, 'o-', color='g', label='Cross-vali
dation score')

plt.legend(loc='best')
plt.show()

-----
import pandas as pd
import numpy as np
from sklearn.linear_model import ElasticNet
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
import warnings

# 忽略所有警告
warnings.filterwarnings('ignore')
# 特征和目标变量
X = data[['年份', '季度', '每年总销量（箱）', '平均每月销量（箱）', '每年总金额

```

```
(元)', '平均每月金额 (元)', '季度总销量 (箱)', '季度总金额 (元)']]]
y = data[['销量 (箱)', '金额 (元)']]

# 保持时间顺序, 将最后 11 步作为测试集
n_test = 11
X_train, X_test = X[:-n_test], X[-n_test:]
y_train, y_test = y[:-n_test], y[-n_test:]

# 设置超参数搜索范围
param_grid = {
    'alpha': np.logspace(-3, 2, 6), # 调整范围和数量
    'l1_ratio': np.linspace(0.1, 1, 10)
}

# 初始化 ElasticNet 回归模型
elastic_net = ElasticNet()

# 使用 GridSearchCV 进行超参数搜索
grid_search = GridSearchCV(elastic_net, param_grid, cv=5, scoring='neg_mean_squared_error', return_train_score=True)

# 拟合 GridSearchCV
grid_search.fit(X_train, y_train)

# 输出最佳参数
print("Best parameters:", grid_search.best_params_)
print("Best cross-validation score: {:.2f}".format(-grid_search.best_score_))

----- 周铠然 · 仅供求职展示 -----
import pandas as pd
import numpy as np
from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import TimeSeriesSplit, learning_curve
import matplotlib.pyplot as plt

# 使用超参数搜索结果
best_params = grid_search.best_params_
model = ElasticNet(alpha=best_params['alpha'], l1_ratio=best_params['l1_ratio'])

# 训练模型
model.fit(X_train, y_train)

# 预测训练集和测试集
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

# 评估模型性能
def evaluate_model(y_true, y_pred):
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    mae = mean_absolute_error(y_true, y_pred)
    r_squared = r2_score(y_true, y_pred)
```

```

        return {'RMSE': rmse, 'MAE': mae, 'R-squared': r_squared}

    train_metrics = evaluate_model(y_train, y_train_pred)
    test_metrics = evaluate_model(y_test, y_test_pred)

    print("训练集评估指标:", train_metrics)
    print("\n 测试集评估指标:", test_metrics)

    # 使用 5 折交叉验证进行评估
    tscv = TimeSeriesSplit(n_splits=5)
    train_sizes, train_scores, test_scores = learning_curve(model, X_train, y_
train, cv=tscv,
                                                    scoring='r2', n_jo
bs=-1,
                                                    train_sizes=np.lin
space(0.1, 1.0, 10))

    # 计算得分的平均值和标准差
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    plt.figure()
    plt.title('Learning Curve for ElasticNet Regression with Multiple Outputs'
)
    plt.xlabel('Training examples')
    plt.ylabel('Score')
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std, train_
scores_mean + train_scores_std, alpha=0.01, color='r')
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std, test_sco
res_mean + test_scores_std, alpha=0.01, color='g')
    plt.plot(train_sizes, train_scores_mean, 'o-', color='r', label='Training
score')
    plt.plot(train_sizes, test_scores_mean, 'o-', color='g', label='Cross-vali
dation score')

    plt.legend(loc='best')
    plt.show()

    -----
    import pandas as pd
    from sklearn.neighbors import KNeighborsRegressor
    from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_sc
ore
    from sklearn.model_selection import TimeSeriesSplit, GridSearchCV, learnin
g_curve
    import matplotlib.pyplot as plt
    import numpy as np

    # 假设 data 是已经加载并预处理过的 DataFrame
    # 特征和目标变量
    X = data[['年份', '季度', '每年总销量（箱）', '平均每月销量（箱）', '每年总金额

```

```
(元)', '平均每月金额 (元)', '季度总销量 (箱)', '季度总金额 (元)']]]
y = data[['销量 (箱)', '金额 (元)']]

# 保持时间顺序, 将最后 11 步作为测试集
n_test = 10
X_train, X_test = X[:-n_test], X[-n_test:]
y_train, y_test = y[:-n_test], y[-n_test:]

# 设置超参数搜索范围
param_grid = {'n_neighbors': np.arange(3, 20)}

# 初始化 KNN 回归模型
knn = KNeighborsRegressor()

# 使用 GridSearchCV 进行超参数搜索
grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='neg_mean_square
d_error', return_train_score=True)

# 拟合 GridSearchCV
grid_search.fit(X_train, y_train)

# 输出最佳参数
print("Best parameters:", grid_search.best_params_)
print("Best cross-validation score: {:.2f}".format(-grid_search.best_score
_))

# 绘制结果
plt.figure(figsize=(10, 5))
plt.title("KNN Regression: Hyperparameter Tuning")
plt.xlabel("Number of Neighbors")
plt.ylabel("Negative Mean Squared Error")
plt.plot(param_grid['n_neighbors'], -grid_search.cv_results_['mean_test_sc
ore'], label='Validation Score')
plt.plot(param_grid['n_neighbors'], -grid_search.cv_results_['mean_train_s
core'], label='Training Score')
plt.legend()
plt.grid(True)
plt.show()

-----
import pandas as pd
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_sc
ore
from sklearn.model_selection import TimeSeriesSplit, learning_curve
import matplotlib.pyplot as plt
import numpy as np

# 特征和目标变量
X = data[['年份', '季度', '每年总销量 (箱)', '平均每月销量 (箱)', '每年总金额
(元)', '平均每月金额 (元)', '季度总销量 (箱)', '季度总金额 (元)']]
y = data[['销量 (箱)', '金额 (元)']] # 多输出目标

# 保持时间顺序, 将最后 11 步作为测试集
n_test = 11
```

```

X_train, X_test = X[:-n_test], X[-n_test:]
y_train, y_test = y[:-n_test], y[-n_test:]

# 初始化 KNN 回归模型
model = KNeighborsRegressor(n_neighbors=7) # n_neighbors 可以调整

# 训练模型
model.fit(X_train, y_train)

# 预测训练集和测试集
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

# 评估模型性能, 需要针对每个目标分别评估
def evaluate_model(y_true, y_pred):
    metrics = {}
    for i, target in enumerate(y_true.columns):
        rmse = np.sqrt(mean_squared_error(y_true[target], y_pred[:, i]))
        mae = mean_absolute_error(y_true[target], y_pred[:, i])
        r_squared = r2_score(y_true[target], y_pred[:, i])
        metrics[target] = {'RMSE': rmse, 'MAE': mae, 'R-squared': r_squared}
    return metrics

train_metrics = evaluate_model(y_train, y_train_pred)
test_metrics = evaluate_model(y_test, y_test_pred)

print("训练集评估指标:")
for target, scores in train_metrics.items():
    print(f"{target}: RMSE: {scores['RMSE']}, MAE: {scores['MAE']}, R-squared: {scores['R-squared']}")

print("\n测试集评估指标:")
for target, scores in test_metrics.items():
    print(f"{target}: RMSE: {scores['RMSE']}, MAE: {scores['MAE']}, R-squared: {scores['R-squared']}")

# 使用 5 折交叉验证进行评估
tscv = TimeSeriesSplit(n_splits=5)
train_sizes, train_scores, test_scores = learning_curve(model, X_train, y_train, cv=tscv,
                                                         scoring='r2', n_jobs=-1,
                                                         train_sizes=np.linspace(0.1, 1.0, 10))

# 计算得分的平均值和标准差
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)

plt.figure()
plt.title('Learning Curve for KNN Regression(A5) with Multiple Outputs')
plt.xlabel('Training examples')

```

```
plt.ylabel('Score')
plt.grid()

plt.fill_between(train_sizes, train_scores_mean - train_scores_std, train_scores_mean + train_scores_std, alpha=0.1, color='r')
plt.fill_between(train_sizes, test_scores_mean - test_scores_std, test_scores_mean + test_scores_std, alpha=0.1, color='g')
plt.plot(train_sizes, train_scores_mean, 'o-', color='r', label='Training score')
plt.plot(train_sizes, test_scores_mean, 'o-', color='g', label='Cross-validation score')

plt.legend(loc='best')
plt.show()

-----
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt

# 假设 data 是已经加载并预处理过的 DataFrame
# 特征和目标变量
X = data[['年份', '季度', '每年总销量（箱）', '平均每月销量（箱）', '每年总金额（元）', '平均每月金额（元）', '季度总销量（箱）', '季度总金额（元）']]
y = data[['销量（箱）', '金额（元）']]

# 保持时间顺序，将最后 11 步作为测试集
n_test = 11
X_train, X_test = X[:-n_test], X[-n_test:]
y_train, y_test = y[:-n_test], y[-n_test:]

# 设置超参数搜索范围
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10]
}

# 初始化随机森林回归模型
rf = RandomForestRegressor()

# 使用 GridSearchCV 进行超参数搜索
grid_search = GridSearchCV(rf, param_grid, cv=5, scoring='neg_mean_squared_error', return_train_score=True)

# 拟合 GridSearchCV
grid_search.fit(X_train, y_train)

# 输出最佳参数
print("Best parameters:", grid_search.best_params_)
print("Best cross-validation score: {:.2f}".format(-grid_search.best_score_))
```

```

-----
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import TimeSeriesSplit, learning_curve
import matplotlib.pyplot as plt

# 使用超参数搜索结果
best_params = grid_search.best_params_
model = RandomForestRegressor(n_estimators=best_params['n_estimators'],
                              max_depth=best_params['max_depth'],
                              min_samples_split=best_params['min_samples_s
plit'])

# 训练模型
model.fit(X_train, y_train)

# 预测训练集和测试集
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

# 评估模型性能
def evaluate_model(y_true, y_pred):
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    mae = mean_absolute_error(y_true, y_pred)
    r_squared = r2_score(y_true, y_pred)
    return {'RMSE': rmse, 'MAE': mae, 'R-squared': r_squared}

train_metrics = evaluate_model(y_train, y_train_pred)
test_metrics = evaluate_model(y_test, y_test_pred)

print("训练集评估指标:", train_metrics)
print("\n 测试集评估指标:", test_metrics)

# 使用 5 折交叉验证进行评估
tscv = TimeSeriesSplit(n_splits=5)
train_sizes, train_scores, test_scores = learning_curve(model, X_train, y_
train, cv=tscv,
                                                    scoring='r2', n_jo
bs=-1,
                                                    train_sizes=np.lin
space(0.1, 1.0, 10))

# 计算得分的平均值和标准差
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)

plt.figure()
plt.title('Learning Curve for RandomForest Regression with Multiple Output
s')
plt.xlabel('Training examples')

```

```
plt.ylabel('Score')
plt.grid()

plt.fill_between(train_sizes, train_scores_mean - train_scores_std, train_scores_mean + train_scores_std, alpha=0.1, color='r')
plt.fill_between(train_sizes, test_scores_mean - test_scores_std, test_scores_mean + test_scores_std, alpha=0.1, color='g')
plt.plot(train_sizes, train_scores_mean, 'o-', color='r', label='Training score')
plt.plot(train_sizes, test_scores_mean, 'o-', color='g', label='Cross-validation score')

plt.legend(loc='best')
plt.show()

-----
import pandas as pd
import numpy as np
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt

# 假设 data 是已经加载并预处理过的 DataFrame
# 特征和目标变量
X = data[['年份', '季度', '每年总销量（箱）', '平均每月销量（箱）', '每年总金额（元）', '平均每月金额（元）', '季度总销量（箱）', '季度总金额（元）']]
y = data['销量（箱）'] # 选择 '销量（箱）' 作为预测目标

# 保持时间顺序，将最后 11 步作为测试集
n_test = 11
X_train, X_test = X[:-n_test], X[-n_test:]
y_train, y_test = y[:-n_test], y[-n_test:]

# 设置超参数搜索范围
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5]
}

# 初始化 GBDT 回归模型
gbdt = GradientBoostingRegressor()

# 使用 GridSearchCV 进行超参数搜索
grid_search = GridSearchCV(gbdt, param_grid, cv=5, scoring='neg_mean_squared_error', return_train_score=True)

# 拟合 GridSearchCV
grid_search.fit(X_train, y_train)

# 输出最佳参数
print("Best parameters:", grid_search.best_params_)
print("Best cross-validation score: {:.2f}".format(-grid_search.best_score_))
```

```

# 使用最佳参数进行预测
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

# 计算测试集的均方误差
test_mse = mean_squared_error(y_test, y_pred)
print("Test MSE:", test_mse)

-----

import pandas as pd
import numpy as np
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import TimeSeriesSplit, learning_curve
import matplotlib.pyplot as plt

# 使用超参数搜索结果
best_params = grid_search.best_params_
model = GradientBoostingRegressor(n_estimators=best_params['n_estimators'],
                                  learning_rate=best_params['learning_rate'],
                                  max_depth=best_params['max_depth'])

# 训练模型
model.fit(X_train, y_train)

# 预测训练集和测试集
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

# 评估模型性能
def evaluate_model(y_true, y_pred):
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    mae = mean_absolute_error(y_true, y_pred)
    r_squared = r2_score(y_true, y_pred)
    return {'RMSE': rmse, 'MAE': mae, 'R-squared': r_squared}

train_metrics = evaluate_model(y_train, y_train_pred)
test_metrics = evaluate_model(y_test, y_test_pred)

print("训练集评估指标:", train_metrics)
print("\n 测试集评估指标:", test_metrics)

# 使用 5 折交叉验证进行评估
tscv = TimeSeriesSplit(n_splits=5)
train_sizes, train_scores, test_scores = learning_curve(model, X_train, y_train, cv=tscv,
                                                         scoring='r2', n_jobs=-1,
                                                         train_sizes=np.linspace(0.1, 1.0, 10))

# 计算得分的平均值和标准差

```

```
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)

plt.figure()
plt.title('Learning Curve for GBDT Regression with Multiple Outputs')
plt.xlabel('Training examples')
plt.ylabel('Score')
plt.grid()

plt.fill_between(train_sizes, train_scores_mean - train_scores_std, train_scores_mean + train_scores_std, alpha=0.1, color='r')
plt.fill_between(train_sizes, test_scores_mean - test_scores_std, test_scores_mean + test_scores_std, alpha=0.1, color='g')
plt.plot(train_sizes, train_scores_mean, 'o-', color='r', label='Training score')
plt.plot(train_sizes, test_scores_mean, 'o-', color='g', label='Cross-validation score')

plt.legend(loc='best')
plt.show()

-----
import pandas as pd
import numpy as np
from xgboost import XGBRegressor
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt

# 假设 data 是已经加载并预处理过的 DataFrame
# 特征和目标变量
X = data[['年份', '季度', '每年总销量（箱）', '平均每月销量（箱）', '每年总金额（元）', '平均每月金额（元）', '季度总销量（箱）', '季度总金额（元）']]
y = data[['销量（箱）', '金额（元）']]

# 保持时间顺序，将最后 11 步作为测试集
n_test = 11
X_train, X_test = X[:-n_test], X[-n_test:]
y_train, y_test = y[:-n_test], y[-n_test:]

# 设置超参数搜索范围
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [3, 5, 7]
}

# 初始化 XGBoost 回归模型
xgb = XGBRegressor(objective='reg:squarederror')

# 使用 GridSearchCV 进行超参数搜索
grid_search = GridSearchCV(xgb, param_grid, cv=5, scoring='neg_mean_squared_error', return_train_score=True)
```

```

# 拟合 GridSearchCV
grid_search.fit(X_train, y_train)

# 输出最佳参数
print("Best parameters:", grid_search.best_params_)
print("Best cross-validation score: {:.2f}".format(-grid_search.best_score_))

-----
import pandas as pd
import numpy as np
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import TimeSeriesSplit, learning_curve
import matplotlib.pyplot as plt

# 使用超参数搜索结果
best_params = grid_search.best_params_
model = XGBRegressor(n_estimators=best_params['n_estimators'],
                    learning_rate=best_params['learning_rate'],
                    max_depth=best_params['max_depth'],
                    objective='reg:squarederror')

# 训练模型
model.fit(X_train, y_train)

# 预测训练集和测试集
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

# 评估模型性能
def evaluate_model(y_true, y_pred):
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    mae = mean_absolute_error(y_true, y_pred)
    r_squared = r2_score(y_true, y_pred)
    return {'RMSE': rmse, 'MAE': mae, 'R-squared': r_squared}

train_metrics = evaluate_model(y_train, y_train_pred)
test_metrics = evaluate_model(y_test, y_test_pred)

print("训练集评估指标:", train_metrics)
print("\n测试集评估指标:", test_metrics)

# 使用 5 折交叉验证进行评估
tscv = TimeSeriesSplit(n_splits=5)
train_sizes, train_scores, test_scores = learning_curve(model, X_train, y_train, cv=tscv,
                                                         scoring='r2', n_jobs=-1,
                                                         train_sizes=np.linspace(0.1, 1.0, 10))

# 计算得分的平均值和标准差
train_scores_mean = np.mean(train_scores, axis=1)

```

```
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)

plt.figure()
plt.title('Learning Curve for XGBoost Regression with Multiple Outputs')
plt.xlabel('Training examples')
plt.ylabel('Score')
plt.grid()

plt.fill_between(train_sizes, train_scores_mean - train_scores_std, train_scores_mean + train_scores_std, alpha=0.1, color='r')
plt.fill_between(train_sizes, test_scores_mean - test_scores_std, test_scores_mean + test_scores_std, alpha=0.1, color='g')
plt.plot(train_sizes, train_scores_mean, 'o-', color='r', label='Training score')
plt.plot(train_sizes, test_scores_mean, 'o-', color='g', label='Cross-validation score')

plt.legend(loc='best')
plt.show()

-----
from sklearn.ensemble import AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV, TimeSeriesSplit
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# 假设数据已经被加载和预处理
data = pd.read_csv('/kaggle/input/202400000/data.csv')

# 选择特征和目标变量
X = data[['年份', '季度', '每年总销量（箱）', '平均每月销量（箱）', '每年总金额（元）', '平均每月金额（元）', '季度总销量（箱）', '季度总金额（元）']]
y = data['销量（箱）'] # 假设我们只关注销量

# 数据划分
n_test = 11
X_train, X_test = X[:-n_test], X[-n_test:]
y_train, y_test = y[:-n_test], y[-n_test:]

# 设置超参数搜索范围
param_grid = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.5, 1.0],
    'base_estimator': [DecisionTreeRegressor(max_depth=3), DecisionTreeRegressor(max_depth=4)]
}

# 初始化 AdaBoost 回归模型
ada = AdaBoostRegressor()
```

```
# 使用 GridSearchCV 进行超参数搜索
grid_search = GridSearchCV(ada, param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train) # 确保这一步被正确执行

# 使用超参数搜索结果
best_params = grid_search.best_params_
model = AdaBoostRegressor(**best_params)

# 训练模型
model.fit(X_train, y_train)

# 预测训练集和测试集
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

# 评估模型性能
def evaluate_model(y_true, y_pred):
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    mae = mean_absolute_error(y_true, y_pred)
    r_squared = r2_score(y_true, y_pred)
    return {'RMSE': rmse, 'MAE': mae, 'R-squared': r_squared}

train_metrics = evaluate_model(y_train, y_train_pred)
test_metrics = evaluate_model(y_test, y_test_pred)

print("训练集评估指标:", train_metrics)
print("\n测试集评估指标:", test_metrics)

# 使用 5 折交叉验证进行评估
tscv = TimeSeriesSplit(n_splits=5)
train_sizes, train_scores, test_scores = learning_curve(model, X_train, y_train, cv=tscv, scoring='r2', n_jobs=-1)

# 计算得分的平均值和标准差
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)

plt.figure()
plt.title('Learning Curve for AdaBoost Regression with Multiple Outputs')
plt.xlabel('Training examples')
plt.ylabel('Score')
plt.grid()

plt.fill_between(train_sizes, train_scores_mean - train_scores_std, train_scores_mean + train_scores_std, alpha=0.1, color='r')
plt.fill_between(train_sizes, test_scores_mean - test_scores_std, test_scores_mean + test_scores_std, alpha=0.1, color='g')
plt.plot(train_sizes, train_scores_mean, 'o-', color='r', label='Training score')
plt.plot(train_sizes, test_scores_mean, 'o-', color='g', label='Cross-validation score')
```

```
plt.legend(loc='best')
plt.show()

-----

import matplotlib.pyplot as plt
import numpy as np

# 提取特定参数的得分
scores_mean = results['mean_test_score']
scores_std = results['std_test_score']
n_estimators = results['param_n_estimators']
learning_rates = results['param_learning_rate']

# 绘制每个学习率的 n_estimators 对性能的影响
learning_rate_values = np.unique(learning_rates)
plt.figure(figsize=(14, 7))

for i, rate in enumerate(learning_rate_values):
    # 找到当前学习率对应的索引
    indices = np.where(learning_rates == rate)
    # 绘制这些索引的平均分数和标准差
    plt.subplot(1, len(learning_rate_values), i + 1)
    plt.errorbar(n_estimators[indices], scores_mean[indices], yerr=scores_std[indices], label=f'Learning Rate: {rate}', fmt='-o')
    plt.title(f'Learning Rate: {rate}')
    plt.xlabel('Number of Estimators')
    plt.ylabel('CV Score (+/- std)')
    plt.legend(loc="best")

plt.tight_layout()
plt.show()

-----

import pandas as pd
import numpy as np
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt

# 假设 data 是已经加载并预处理过的 DataFrame
# 特征和目标变量
X = data[['年份', '季度', '每年总销量 (箱)', '平均每月销量 (箱)', '每年总金额 (元)', '平均每月金额 (元)', '季度总销量 (箱)', '季度总金额 (元)']]
y = data[['销量 (箱)', '金额 (元)']]

# 保持时间顺序, 将最后 11 步作为测试集
n_test = 11
X_train, X_test = X[:-n_test], X[-n_test:]
y_train, y_test = y[:-n_test], y[-n_test:]

# 设置超参数搜索范围
param_grid = {
    'hidden_layer_sizes': [(50,), (100,), (50, 50)],
    'activation': ['tanh', 'relu'],
```

```
        'learning_rate_init': [0.001, 0.01, 0.1]
    }

    # 初始化 MLP 回归模型
    mlp = MLPRegressor(max_iter=1000)

    # 使用 GridSearchCV 进行超参数搜索
    grid_search = GridSearchCV(mlp, param_grid, cv=5, scoring='neg_mean_square
d_error', return_train_score=True)

    # 拟合 GridSearchCV
    grid_search.fit(X_train, y_train)

    # 输出最佳参数
    print("Best parameters:", grid_search.best_params_)
    print("Best cross-validation score: {:.2f}".format(-grid_search.best_score
_))

    -----
import pandas as pd
import numpy as np
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_sc
ore
from sklearn.model_selection import TimeSeriesSplit, learning_curve
import matplotlib.pyplot as plt

    # 使用超参数搜索结果
    best_params = grid_search.best_params_
    model = MLPRegressor(hidden_layer_sizes=best_params['hidden_layer_sizes'],
                          activation=best_params['activation'],
                          learning_rate_init=best_params['learning_rate_init'],
                          max_iter=1000)

    # 训练模型
    model.fit(X_train, y_train)

    # 预测训练集和测试集
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    # 评估模型性能
    def evaluate_model(y_true, y_pred):
        rmse = np.sqrt(mean_squared_error(y_true, y_pred))
        mae = mean_absolute_error(y_true, y_pred)
        r_squared = r2_score(y_true, y_pred)
        return {'RMSE': rmse, 'MAE': mae, 'R-squared': r_squared}

    train_metrics = evaluate_model(y_train, y_train_pred)
    test_metrics = evaluate_model(y_test, y_test_pred)

    print("训练集评估指标:", train_metrics)
    print("\n 测试集评估指标:", test_metrics)

    # 使用 5 折交叉验证进行评估
```

```

tscv = TimeSeriesSplit(n_splits=5)
train_sizes, train_scores, test_scores = learning_curve(model, X_train, y_train, cv=tscv,
                                                         scoring='r2', n_jobs=-1,
                                                         train_sizes=np.linspace(0.1, 1.0, 10))

# 计算得分的平均值和标准差
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)

plt.figure()
plt.title('Learning Curve for MLP Regression with Multiple Outputs')
plt.xlabel('Training examples')
plt.ylabel('Score')
plt.grid()

plt.fill_between(train_sizes, train_scores_mean - train_scores_std, train_scores_mean + train_scores_std, alpha=0.1, color='r')
plt.fill_between(train_sizes, test_scores_mean - test_scores_std, test_scores_mean + test_scores_std, alpha=0.1, color='g')
plt.plot(train_sizes, train_scores_mean, 'o-', color='r', label='Training score')
plt.plot(train_sizes, test_scores_mean, 'o-', color='g', label='Cross-validation score')

plt.legend(loc='best')
plt.show()

```

```

import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import LSTM, Dense

# 假设 data 是已经加载并预处理过的 DataFrame
# 特征和目标变量
X = data[['年份', '季度', '每年总销量 (箱)', '平均每月销量 (箱)', '每年总金额 (元)', '平均每月金额 (元)', '季度总销量 (箱)', '季度总金额 (元)']].values
y = data[['销量 (箱)', '金额 (元)']].values

# 保持时间顺序, 将最后 12 个月作为测试集
n_test = 12
X_train, X_test = X[:-n_test], X[-n_test:]
y_train, y_test = y[:-n_test], y[-n_test:]

# LSTM 需要输入格式 [samples, time steps, features]
X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))

# 定义模型
model = Sequential()

```

```
model.add(LSTM(50, activation='relu', input_shape=(1, X_train.shape[2])))
model.add(Dense(2)) # 输出层有两个单元, 对应两个目标变量
model.compile(optimizer='adam', loss='mean_squared_error')

# 训练模型
model.fit(X_train, y_train, epochs=40, batch_size=1, validation_data=(X_test, y_test), verbose=2)

-----
# 预测
y_pred = model.predict(X_test)

# 计算评估指标
from sklearn.metrics import mean_squared_error, r2_score
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print('Test RMSE: %.3f' % rmse)
r2 = r2_score(y_test, y_pred)
print('Test R2 Score: %.3f' % r2)

-----
import matplotlib.pyplot as plt

# 绘制实际值和预测值
plt.figure(figsize=(10, 5))
plt.plot(y_test[:, 0], label='Actual Sales')
plt.plot(y_pred[:, 0], label='Predicted Sales')
plt.title('LSTM Predictions vs Actual')
plt.xlabel('Time')
plt.ylabel('Sales')
plt.legend()
plt.show()

-----
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import GRU, Dense

# 假设数据已经是经过标准化处理并加载到 data DataFrame 中
# 特征和目标变量
X = data[['年份', '季度', '每年总销量 (箱)', '平均每月销量 (箱)', '每年总金额 (元)', '平均每月金额 (元)', '季度总销量 (箱)', '季度总金额 (元)']].values
y = data[['销量 (箱)', '金额 (元)']].values

# 保持时间顺序, 将最后 12 个月作为测试集
n_test = 12
X_train, X_test = X[:-n_test], X[-n_test:]
y_train, y_test = y[:-n_test], y[-n_test:]

# 重新格式化输入数据为 [samples, time steps, features]
X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))

# 定义模型
model = Sequential()
```

```
model.add(GRU(50, activation='relu', input_shape=(1, X_train.shape[2])))
model.add(Dense(2)) # 输出层有两个单元, 对应两个目标变量
model.compile(optimizer='adam', loss='mean_squared_error')

# 训练模型
model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test), verbose=2)

-----
# 预测
y_pred = model.predict(X_test)

# 计算评估指标
from sklearn.metrics import mean_squared_error, r2_score
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print('Test RMSE: %.3f' % rmse)
r2 = r2_score(y_test, y_pred)
print('Test R2 Score: %.3f' % r2)

-----
import matplotlib.pyplot as plt

# 绘制实际值和预测值
plt.figure(figsize=(10, 5))
plt.plot(y_test[:, 0], label='Actual Sales')
plt.plot(y_pred[:, 0], label='Predicted Sales')
plt.title('GRU Predictions vs Actual')
plt.xlabel('Time')
plt.ylabel('Sales')
plt.legend()
plt.show()

-----
# 将三维数据转换为二维数据
X_train_2d = X_train.reshape((X_train.shape[0], X_train.shape[2]))
X_test_2d = X_test.reshape((X_test.shape[0], X_test.shape[2]))

print("New X_train shape:", X_train_2d.shape) # 应输出 (83, 8)
print("New X_test shape:", X_test_2d.shape)   # 应输出 (12, 8)

-----
from sklearn.ensemble import StackingRegressor
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# 假设模型实例已经正确创建
models = [
    ('linear', linear_model),
    ('ridge', ridge_model),
    ('lasso', lasso_model),
    ('elastic', elastic_model),
    ('knn', knn_model),
    ('rf', rf_model),
```

```
( 'gbdt', gbd_t_model),
( 'xgboost', xgb_model),
( 'adaboost', ada_model),
( 'bpnn', bpnn_model)
# LSTM 和 GRU 如果需要整合, 可以先单独预测, 然后将预测结果作为特征或者独立处
理
]

# 分别为每个目标变量创建堆叠模型
stacked_ensemble_1 = StackingRegressor(
    estimators=models,
    final_estimator=LinearRegression()
)

stacked_ensemble_2 = StackingRegressor(
    estimators=models,
    final_estimator=LinearRegression()
)

# 训练第一个目标变量的模型
stacked_ensemble_1.fit(X_train_2d, y_train[:, 0])

# 训练第二个目标变量的模型
stacked_ensemble_2.fit(X_train_2d, y_train[:, 1])

# 进行预测
y_pred_1 = stacked_ensemble_1.predict(X_test_2d)
y_pred_2 = stacked_ensemble_2.predict(X_test_2d)

# 组合预测结果
y_pred = np.column_stack((y_pred_1, y_pred_2))

# 计算评估指标
rmse1 = np.sqrt(mean_squared_error(y_test[:, 0], y_pred[:, 0]))
r21 = r2_score(y_test[:, 0], y_pred[:, 0])
print('Test RMSE 1:', rmse1)
print('Test R2 Score 1:', r21)

rmse2 = np.sqrt(mean_squared_error(y_test[:, 1], y_pred[:, 1]))
r22 = r2_score(y_test[:, 1], y_pred[:, 1])
print('Test RMSE 2:', rmse2)
print('Test R2 Score 2:', r22)
```