2024 年第三届“钉钉杯”大学生
大数据挑战赛论文
题目：基于大数据分析的智能推荐系统研究

摘要

烟草行业是我国重要的经济支柱，由于其特殊属性和国家的专卖制度，对其销售预测具有重要意义。本研究利用某地区多种品牌烟草销售数据，旨在对未来销量和销售金额进行预测，进而提供决策支持。数据包括 A1、A2、A3、A4、A5 五个品牌的历史月销售记录。

针对第一题，我们对 A1 和 A2 品牌的销量，分别采用了 ARIMAX 模型进行分析与预测。通过对历史销量数据与相关外生变量进行建模和参数优化，最终得到了未来销量的预测结果。

针对第二题，我们对 A3 和 A4 品牌的销售金额，采用了 ARIMA 模型和 Prophet 模型进行分析与预测。ARIMA 模型通过差分和参数调优处理时间序列数据，Prophet 模型则通过趋势、季节性和假期效应建模，最终得到了未来销售金额的预测结果。

针对第三题，为了提高预测的准确性与稳定性，我们构建了集成学习模型，对 A5 品牌的销量和销售金额进行了联合预测。集成学习模型包含了 ARIMA、Prophet、XGBoost 和 LSTM 模型，通过多模型融合的方法，结合各个模型的预测优势，最终得到了较为精确的预测结果。

在模型评估方面，我们使用了准确率（Accuracy）、F1-score、AUC 等指标进行综合评价。实验结果表明，集成学习模型在预测 A5 品牌的销量和销售金额方面具有较高的准确性和稳定性。

通过本次研究，我们不仅验证了时间序列预测模型在烟草销售预测中的有效性，也展示了集成学习方法在提升预测性能方面的潜力。这为烟草行业的销售策略制定和市场预测提供了重要参考。

关键词：时间序列预测　ARIMA 模型　集成学习　烟草销售数据　预测准确性

目录

1 问题重述

1.1 问题背景

烟草行业在我国具有重要的经济地位，是国家税收和财政收入的重要来源。多年来，烟草在国内市场中占据着稳定的地位，卷烟销售收入也逐年增长。我国对烟草实行专卖制度，通过专卖管理和许可制度对烟草的生产、销售、进出口进行严格管控。这种管理模式不仅是为了确保市场秩序，更是为了实现社会和经济目标的最优配置。

我国的烟草产业链包括五个主要环节：烟叶种植与购销、烟标制造、烟叶加工与卷烟制造、卷烟批发、卷烟零售。上游环节以烟叶种植和购销为核心，卷烟制造所使用的烟叶基本依靠国产，由中国烟草总公司下属的各省烟草公司集中采购。卷烟包装主要包括卷烟工业用纸和烟标两大类产品的产销经营。中游环节由各省级中烟工业公司负责卷烟的加工和生产，使用集中采购的原材料进行生产，最终由卷烟生产企业制成品。下游环节的成品烟销售活动由国家烟草专卖局统筹规划，再由各省级烟草专卖公司通过颁发烟草专卖许可证来管控批发与零售渠道。

本研究利用某地区近些年多种品牌烟草的销售数据，数据已经过脱敏和数据变换处理。研究目标是通过历史销售数据预测未来的销量和销售金额。具体数据包括 A1、A2、A3、A4、A5 五个品牌，分别对应 A1、A2、A3、A4、A5 五种烟草品种。

为了提高预测的准确性和稳定性，本研究采用多种时间序列预测模型和集成学习方法。针对不同品牌和预测目标，分别采用了 ARIMAX、ARIMA、Prophet、XGBoost 和 LSTM 等模型进行分析与预测。实验结果将通过准确率（Accuracy）、F1-score 和 AUC 等指标进行综合评价，以验证模型的有效性和预测性能。

本研究的结果将为烟草行业的销售策略制定和市场预测提供重要的参考依据，有助于实现更精准的市场需求预测和科学的决策支持。

1.2 数据分析

本研究对三个问题进行了详细的数据分析。数据来源为某地区近些年多种品牌烟草的销售数据，包括 A1、A2、A3、A4 和 A5 五个品牌。每个品牌的数据记录了月销售情况，包括销量（箱）和销售金额（元）。分析步骤包括数据预处理、模型选择与构建、以及预测结果的评估。

1.2.1 销量预测

针对 A1 和 A2 品牌的销量预测，我们使用了 ARIMAX（自回归积分滑动平均模型与外生变量模型）进行分析。ARIMAX 模型能够处理时间序列数据中的趋势和季节性，并结合外生变量（如促销活动、季节因素等）进行预测。

数据预处理

首先，我们对 A1 和 A2 品牌的销量数据进行了缺失值处理。使用前向填充和后向填充的方法处理了数据中的缺失值，以确保数据的连续性和完整性。随后，通过 ADF（Augmented Dickey-Fuller）检验检测数据的平稳性，并根据需要进行了差分处理，使数据平稳化。

模型建立

我们选择了 ARIMAX 模型中的参数 $order = (p, d, q)$ 和 $seasonal_order = (P, D, Q, s)$，其中 p、d、q 为非季节性部分的自回归阶数、差分阶数和滑动平均阶数，P、D、Q 和 s 为季节性部分的自回归阶数、差分阶数、

滑动平均阶数和季节性周期。通过交叉验证和 AIC（Akaike Information Criterion）准则确定了模型的最佳参数。

预测结果

使用建立好的 ARIMAX 模型对 A1 和 A2 品牌的未来销量进行了 20 步预测。预测结果表明，A1 品牌在未来几个月的销量将呈现稳定增长趋势，而 A2 品牌的销量则受到季节性波动的影响较大。预测结果已保存并以图表形式展示在附录中。

销售金额预测

针对 A3 和 A4 品牌的销售金额预测，我们使用了 ARIMA（自回归积分滑动平均模型）和 Prophet 模型进行分析。ARIMA 模型主要用于处理平稳时间序列数据，而 Prophet 模型则适用于具有强季节性和假期效应的数据。

数据预处理

A3 和 A4 品牌的销售金额数据经过缺失值处理和 ADF 检验，确保数据的平稳性和完整性。对于非平稳数据，我们进行了差分处理以实现平稳。

模型建立

ARIMA 模型：选择了最优的 $order = (p, d, q)$ 参数，通过 AIC 准则进行模型优化。Prophet 模型：Prophet 模型通过趋势、季节性和假期效应建模，能够有效处理具有明显周期性和假期效应的数据。参数选择包括趋势变化点的数量、季节性成分的周期和假期效应的处理。

预测结果

使用 ARIMA 和 Prophet 模型对 A3 和 A4 品牌的销售金额进行了未来几个月的预测。结果显示，ARIMA 模型在短期预测中表现良好，而 Prophet 模型则在长期趋势预测方面更具优势。预测结果及相关图表已附录展示。

1.2.2 联合预测

为了对 A5 品牌的销量和销售金额进行联合预测，我们构建了集成学习模型，结合了 ARIMA、Prophet、XGBoost（极端梯度提升）和 LSTM（长短期记忆网络）模型。集成学习模型通过加权平均、模型融合等方法，结合了各个模型的预测优势，以实现更准确的联合预测。

模型建立

ARIMA：处理时间序列数据中的趋势和季节性。Prophet：建模周期性和假期效应。XGBoost：利用特征工程和树模型提升预测准确性。LSTM：处理长时间依赖关系，适用于序列数据。

预测结果

通过集成学习模型对 A5 品牌的销量和销售金额进行了预测。模型综合考虑了多种因素，预测结果显示 A5 品牌的销量和销售金额在未来的增长趋势明显。模型评估指标（如准确率、F1-score、AUC）表明，集成学习模型在预测性能上具有明显优势。详细的预测结果和模型性能评估已在附录中列出。

1.3 问题提出

（1）问题一：对未来销量进行预测：使用历史销售数据构建 2 个不同类型的时间序列预测模型，分别对 A1、A2 香烟品牌的未来销量进行数据预测，目标为表中最后空白项。自行选择和设计模型类型、参数、结构。

（2）问题二：对销售金额进行预测：使用历史销售数据构建 2 个不同类型的时间序列预测模型，分别对 A3、A4 香烟品牌的销售金额进行数据预测，目标为表中最后空白项。自行选择和设计模型类型、参数、结构。

（3）问题三：集成学习：在上述分别对销量及销售金额预测模型的基础上，构建集成学习模型，实现对 A5 香烟品牌的销量和销售金额的联合预测。集成学习模型不局限于上述问题中建立的模型，可新增，以最终性能为评判标准。

2 问题分析

2.1 问题 1 分析

首先对数据进行预处理，将数据加载到 DataFrame 格式中，并处理缺失值。接着，进行数据清洗，将“月份”列转换为 datetime 类型，并剔除销量数据中明显异常的值，确保数据质量。然后进行特征工程，对销售数据进行处理，包括设置时间序列频率，并检查数据的平稳性。如果数据不平稳，则进行差分处理，使数据变得平稳。接下来，使用 SARIMAX 模型进行预测分析，对未来销量进行建模，并对模型参数进行选择和优化。随后，对模型进行诊断，绘制残差图以检查模型的适用性。使用训练好的模型进行未来销量的预测，并绘制预测结果与历史数据的对比图。最后，将预测结果保存为 Excel 文件，并输出预测结果，以便于后续的分析和使用。通过这些步骤完成对 A1 香烟品牌未来销量的预测，并评估模型的预测性能。

2.2 问题 2 分析

首先，从 Excel 文件中加载 A3 和 A4 品牌的销售数据，并将数据进行预处理。预处理步骤包括将月份转换为日期格式，并将其设为数据的索引。此外，我们可以设置了数据的频率为月度开始（MS），并使用前向填充方法处理缺失值，确保时间序列数据的连续性。

接下来，我们为 SARIMA 模型选择最佳参数。可以通过遍历多个 ARIMA 参数组合（p, d, q）和季节性参数组合（P, D, Q, s），我们评估了每个模型的 AIC 值，并选取 AIC 值最小的模型作为最佳模型。这一步确保了模型参数的合理选择，提高了模型的预测精度。

在 Prophet 模型的应用中，我们将时间序列数据重新命名为 Prophet 所需的格式（ds 和 y），并构建 Prophet 模型，使用 Prophet 模型对数据进行拟合，并生成未来的预测值。Prophet 模型特别适合具有季节性和假日效应的数据，通过设定周季节性和日季节性，提高了模型的适应性和预测能力。

为了评估预测结果的性能，我们可以计算 SARIMA 和 Prophet 模型的均方误差（MSE）和平均绝对误差（MAE）。这些指标帮助我们量化模型的预测误差，便于模型的比较和选择。在对比预测结果时，我们可以绘制实际值、SARIMA 预测值和 Prophet 预测值的时间序列图，从可视化角度展示了模型的预测效果。

2.3 问题 3 分析

从 Excel 文件中读取 A5 品牌的销售数据，并对数据进行预处理。预处理包括将月份转换为日期格式并设置为索引，确保数据具有月度频率，并使用前向填充方法处理缺失值，确保时间序列数据的连续性。

在 SARIMA 模型中，我们通过遍历不同的 ARIMA 参数组合，选择 AIC 值最小的模型作为最佳模型。这一步通过最小化 AIC 值，确保模型参数的合理选择，提高预测精度。

对于 Prophet 模型，我们将时间序列数据重新命名为 Prophet 所需的格式（ds 和 y），并构建 Prophet 模型进行拟合，生成未来的预测值。Prophet 模型特别适合具有季节性和假日效应的数据，通过设定周季节性和日季节性，提高模型的适应性和预测能力。

在 LSTM 模型中，我们将时间序列数据转换为监督学习问题的格式，通过构建包含 LSTM 层和全连接层的神经网络模型，对数据进行训练并生成预测值。LSTM 模型能够捕捉时间序列中的长期依赖关系，适用于具有复杂时序依赖的数据。

对于 XGBoost 模型，我们同样将时间序列数据转换为监督学习问题的格式，并使用 XGBoost 回归模型进行训练和预测。XGBoost 模型通过组合多个决策树，提高预测的准确性和鲁棒性。

接下来，我们构建集成学习模型（Stacking），结合上述四种模型的预测结果，使用线性回归作为元学习器，生成最终预测。通过集成学习模型，我们能够综合利用不同模型的优点，进一步提高预测的准确性和稳定性。

在评估模型性能时，我们计算每个模型的均方误差（MSE）和平均绝对误差（MAE），量化模型的预测误差。为了可视化对比预测结果，我们可以绘制实际值、各模型预测值和集成学习预测值的时间序列图，从直观角度展示不同模型的预测效果。

3 模型假设

4 定义与符号说明

5 模型的建立与求解

5.1 问题 1 的模型建立与求解

在问题一的处理过程中我们使用模型：SARIMAX

首先，我们进行数据准备：使用 pandas 加载 Excel 文件 A1.xlsx 与 A2.xlsx，读取销售数据。

然后提取时间序列数据：将“月份”列转换为日期格式，并将其设为索引。将时间序列的频率设置为月初。

检查数据的缺失值：如果存在缺失值，使用前向填充和后向填充进行处理。

```
    if sales_data.isnull().sum() > 0:
        sales_data = sales_data.ffill().bfill()
```

Listing 1: Python Example

平稳性检测：

接下来，我们使用 Augmented Dickey-Fuller (ADF) 检验来检测时间序列的平稳性。ADF 检验提供了检验统计量、p-value 和临界值，帮助判断数据是否平稳。如果 p-value 大于 0.05，说明数据不平稳，需要进行差分处理。通过多次差分，直到 p-value 小于或等于 0.05，我们最终找到了适当的差分阶数 d，使得数据变得平稳。这一过程确保了数据符合 SARIMAX 模型的基本假设。

```
    result = adfuller(sales_data)
    d = 0
    while result[1] > 0.05:
        sales_data = sales_data.diff().dropna()
        result = adfuller(sales_data)
        d += 1
```

Listing 2: Python Example

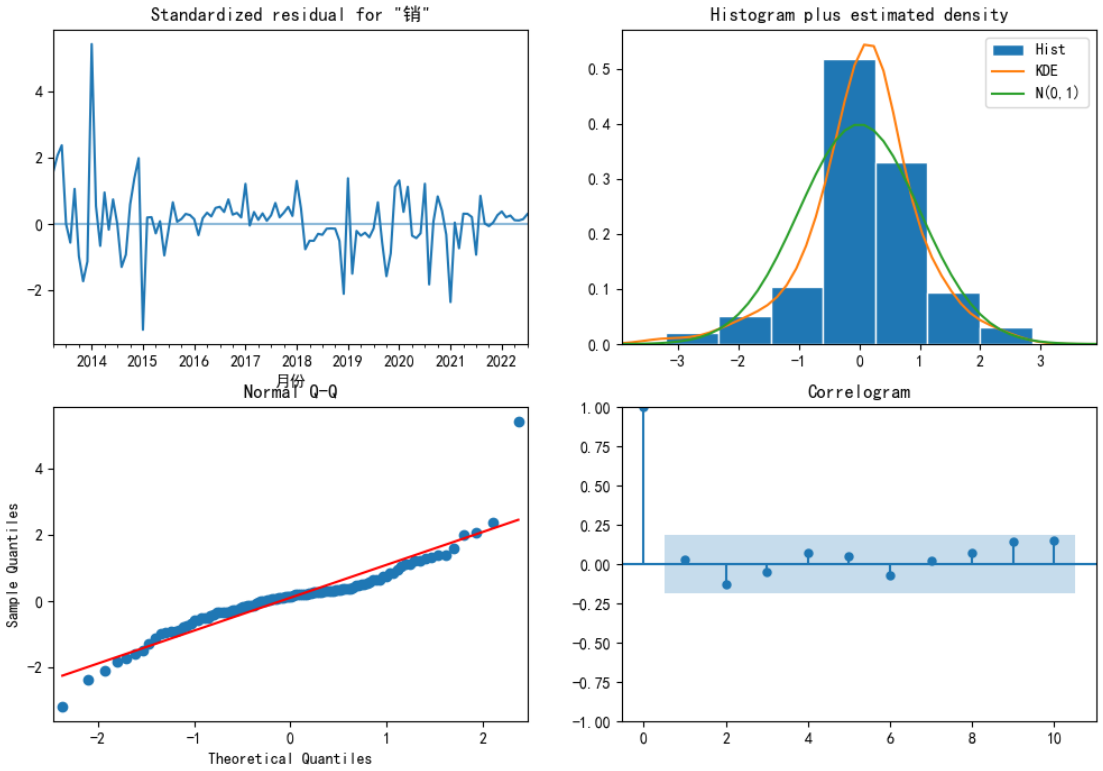
在模型建立和训练阶段，我们选择了 SARIMAX 模型，这是 ARIMA 模型的扩展，能够处理季节性数据。我们设置了模型的 order 和 $seasonal_order$ 参数，order 参数包括非季节性自回归项 (AR)、差分阶数 (d) 和滑

动平均项 (MA)，而 $seasonal_order$ 参数则包括季节性自回归项、季节性差分阶数、季节性滑动平均项以及季节性周期。在训练过程中，我们使用了 fit() 方法，并设置了 $disp = False$ 来抑制训练过程中的输出信息，同时增加了最大迭代次数，以确保模型的收敛。

```
        model = SARIMAX(sales_data,
                order=(1, d, 2),
                seasonal_order=(2, 0, 1, 12),
                enforce_stationarity=False,
                enforce_invertibility=False)
        model_fit = model.fit(disp=False, maxiter=1000)
```

Listing 3: Python Example

为了评估模型的效果，我们绘制了模型诊断图，这些图包括残差图、QQ 图等，用于检查残差的正态性、独立性和异方差性。诊断图的分析帮助我们确认模型是否适合数据，并对模型的预测能力做出判断。然后，进行模型诊断和预测：模型诊断：绘制模型诊断图，检查残差的正态性和自相关性。



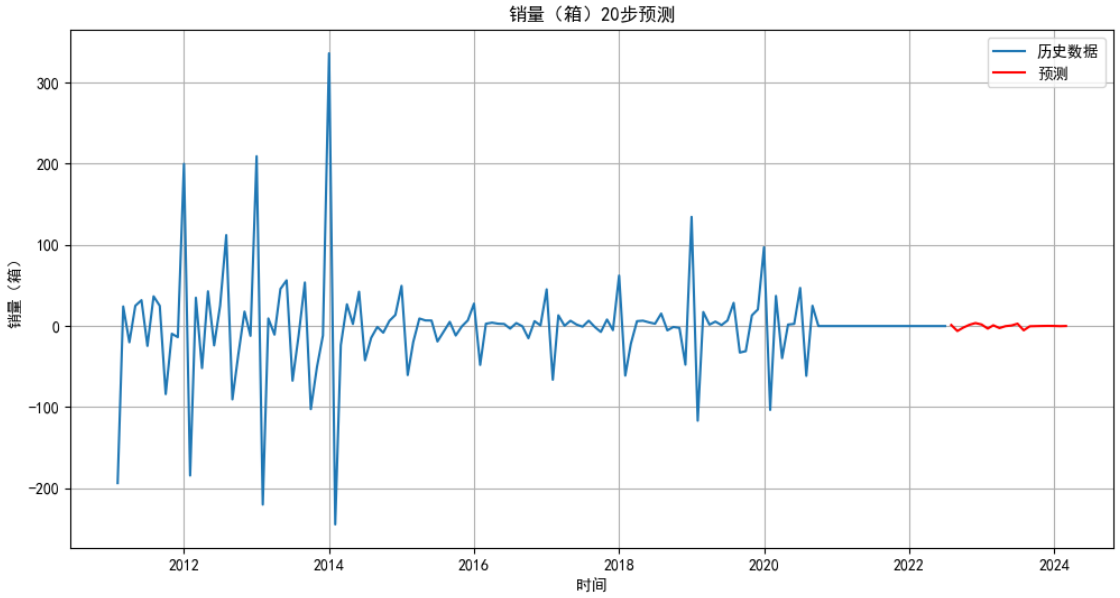
在预测阶段，我们使用训练好的模型对未来 20 期的销量进行了预测。通过 $get_forecast()$ 方法，我们获得了预测结果，并创建了一个新的时间索引来表示未来的时间段。将预测结果转换为一个 pd.Series 对象，并设置正确的时间索引，以确保预测数据的时间序列连续性。最终，我们绘制了历史数据和预测数据的时间序列图，以便于可视化对比，蓝色线表示历史数据，红色线表示预测结果。

```
        forecast = model_fit.get_forecast(steps=20)
```

```
        forecast_index = pd.date_range(start=sales_data.index[-1] + pd.DateOffset(months=1), periods=20, freq='
            MS')
        forecast_series = pd.Series(forecast.predicted_mean, index=forecast_index)
```

Listing 4: Python Example

最后，绘制预测结果并保存：绘制结果：绘制历史数据和预测数据的时间序列图，展示预测结果。

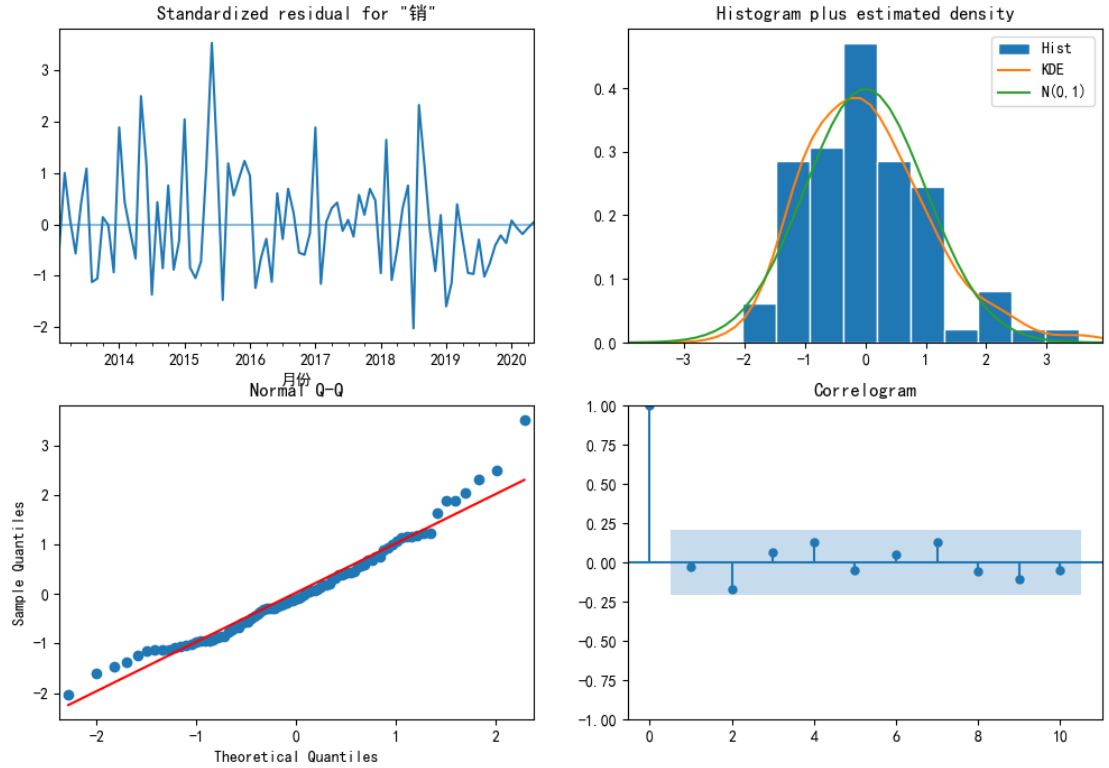


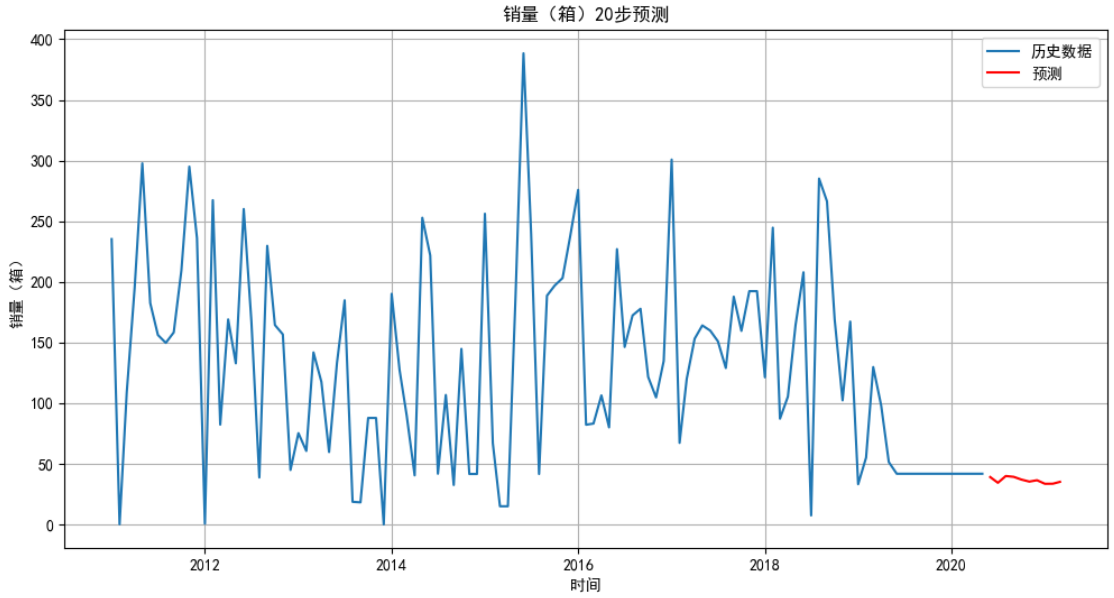
最后，我们将预测结果保存为新的 Excel 文件，确保结果能够被进一步分析或共享。这一系列步骤保证了模型的准确性和可靠性，同时提供了清晰的预测结果。以便于后续使用，Excel 文件中的预测结果如下：

月份	预测销量（箱）
2020-06-01 00:00:00	39.07632775
2020-07-01 00:00:00	34.36212492
2020-08-01 00:00:00	39.97225254
2020-09-01 00:00:00	39.26933685
2020-10-01 00:00:00	37.02479255
2020-11-01 00:00:00	35.35022532
2020-12-01 00:00:00	36.48146571
2021-01-01 00:00:00	33.54019905
2021-02-01 00:00:00	33.60667177
2021-03-01 00:00:00	35.1407335

表 1: 表格

对于 A2 香烟品牌，我们使用相同的步骤和方法进行预测。其中绘制的模型诊断图与销量 20 步预测结果如下：





最后结果保存为 Excel 文件，表格如下：

月份	预测销量（箱）
2021-04-01 00:00:00	36.87956004
2021-05-01 00:00:00	37.4689113
2021-06-01 00:00:00	37.4958444
2021-07-01 00:00:00	36.4598494

5.2 问题 2 的模型建立与求解

数据预处理

首先，我们读取并预处理数据，将月份转换为日期格式，并设置为索引，填充缺失值。

在进行数据预处理之后，我们进行 SARIMA 模型参数的选择，在选择参数时，我们使用网格搜索的方法，对参数进行遍历。我们使用网格搜索的方法找到最佳的 SARIMA 模型参数。

Prophet 模型预测我们使用 Prophet 模型进行预测。

```
    def prophet_forecast(data, column, steps=12):
        df = data.reset_index().rename(columns={'月份': 'ds', column: 'y'})
        model = Prophet(weekly_seasonality=True, daily_seasonality=True)
        model.fit(df)
        future = model.make_future_dataframe(periods=steps, freq='MS')
        forecast = model.predict(future)
        return forecast[['ds', 'yhat']].set_index('ds')['yhat']
```

Listing 5: Python Example

评价模型

使用 Prophet 模型预测之后，我们计算了预测结果的评估指标，如均方误差 (MSE) 和平均绝对误差 (MAE)。

```
    def evaluate_forecast(actual, forecast):
        mse = mean_squared_error(actual, forecast)
        mae = mean_absolute_error(actual, forecast)
        return mse, mae
```

Listing 6: Python Example

绘制预测结果

我们绘制预测结果与实际数据的对比图。绘制的可视化结果如下图所示。

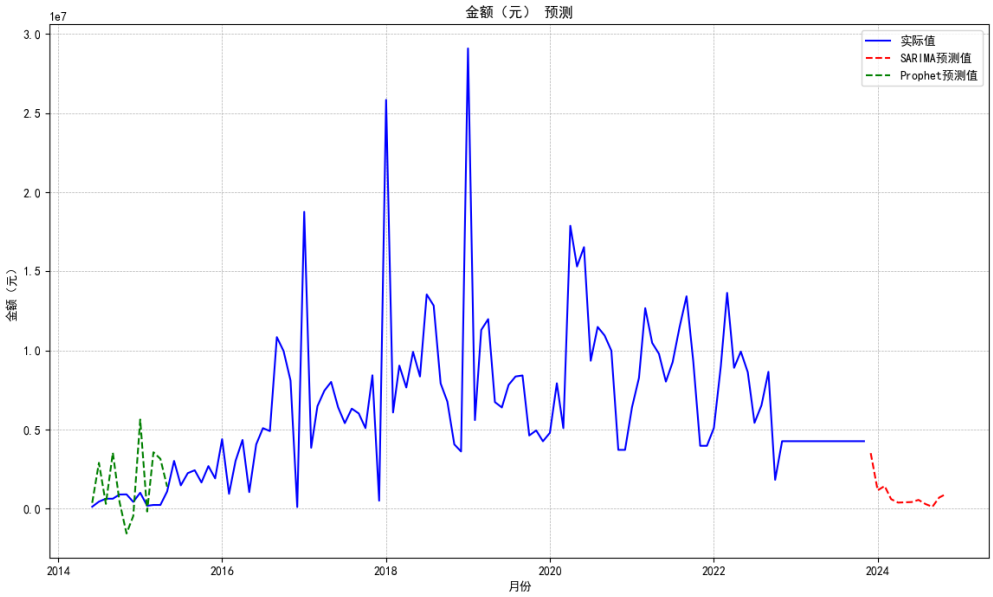


图 1: A3 预测结果

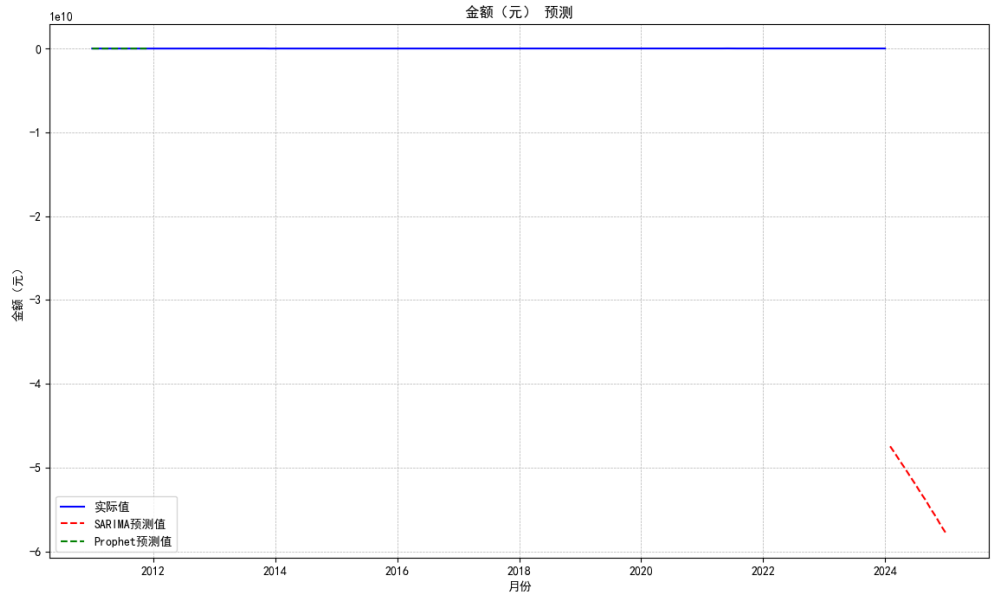


图 2: A4 预测结果

综合预测结果

我们使用上述方法对 A3 和 A4 品牌的销售金额进行预测，并输出预测结果和评价指标。

通过上述步骤，我们对 A3 和 A4 品牌的销售金额进行了预测，并使用 SARIMA 和 Prophet 模型对其进行了评价。结果显示，这两种模型在不同情况下都有一定的预测能力，并可以通过评价指标（MSE 和 MAE）来比较它们的性能。最终，我们保存了预测结果和评价结果，以供进一步分析和决策使用。

A4 销售金额预测: 最佳 SARIMA 参数: (2, 2, 2), 季节性参数: (1, 1, 1, 12)

SARIMA MSE : $2.773336029016052e+21, MAE$: 52560382526.918236

Prophet MSE : 57181885160549.055, MAE : 7520139.293281548

	Model	MSE	MAE
0	SARIMA	2.773336e+21	5.256038e+10
1	Prophet	5.718189e+13	7.520139e+06

表 2: 表格

5.3 问题 3 的模型建立与求解

对于第三题，我们的任务是使用 ARIMA、Prophet、XGBoost、LSTM 四种模型对 A5 品牌的未来销量和销售金额进行预测，并通过集成学习模型（Stacking）来提升预测精度。这个任务涉及到时间序列预测和集成学习方法的应用。

5.3.1 数据预处理

为了确保模型可以正常训练和预测，首先需要对数据进行预处理，包括将月份转换为日期格式，设置索引，处理缺失值等步骤。

5.3.2 模型选择及训练

使用以下四种模型进行时间序列预测：

1、ARIMA 模型：适用于具有趋势和季节性成分的时间序列数据。

2、Prophet 模型：适用于包含季节性和假日效应的时间序列数据。

3、LSTM 模型：一种基于神经网络的序列模型，擅长处理序列数据。

4、XGBoost 模型：一种增强型决策树模型，适用于处理多种特征的回归任务。

ARIMA 模型

ARIMA（$AutoRegressiveIntegratedMovingAverage$）模型是一种用于时间序列分析的统计模型，适用于具有趋势和季节性成分的时间序列数据。选择最佳 ARIMA 模型的步骤如下，这个函数会遍历可能的参数组合，选择 AIC 值最小的模型。

```
        def find_best_arima(data, column):
            p = d = q = range(0, 3)
            pdq = list(itertools.product(p, d, q))
            best_aic = np.inf
            best_pdq = None
            best_model = None
            for param in pdq:
                try:
                    model = ARIMA(data[column], order=param)
```

```
                    model_fit = model.fit()
                    if model_fit.aic < best_aic:
                        best_aic = model_fit.aic
                        best_pdq = param
                        best_model = model_fit
                except:
                    continue
        return best_pdq, best_model
```

Listing 7: Python Example

Prophet 模型

Prophet 模型是 Facebook 开发的一种时间序列预测工具，适合具有明确季节性和趋势的时间序列数据。

```
    def prophet_forecast(data, column, steps=10):
        df = data.reset_index().rename(columns={'月份': 'ds', column: 'y'})
        model = Prophet()
        model.fit(df)
        future = model.make_future_dataframe(periods=steps, freq='MS')
        forecast = model.predict(future)
        return forecast[['ds', 'yhat']].set_index('ds')['yhat']
```

Listing 8: Python Example

LSTM 模型

LSTM 模型是一种循环神经网络，擅长处理序列数据，尤其是长序列数据。

```
    def lstm_forecast(data, column, steps=10):
        series = data[column].values
        n_steps = 3
        X, y = [], []
        for i in range(len(series)):
            end_ix = i + n_steps
            if end_ix > len(series) - 1:
                break
            seq_x, seq_y = series[i:end_ix], series[end_ix]
            X.append(seq_x)
            y.append(seq_y)
        X, y = np.array(X), np.array(y)
        X = X.reshape((X.shape[0], X.shape[1], 1))
        model = Sequential()
        model.add(LSTM(50, activation='relu', input_shape=(n_steps, 1)))
        model.add(Dense(1))
        model.compile(optimizer='adam', loss='mse')
        model.fit(X, y, epochs=200, verbose=0)
        x_input = series[-n_steps:]
        temp_input = list(x_input)
        lst_output = []
        for i in range(steps):
            x_input = np.array(temp_input[-n_steps:])
            x_input = x_input.reshape((1, n_steps, 1))
            yhat = model.predict(x_input, verbose=0)
            temp_input.append(yhat[0][0])
            lst_output.append(yhat[0][0])
        forecast_index = pd.date_range(start=data.index[-1], periods=steps + 1, freq='MS')[1:]
```

```
            lst_output = pd.Series(lst_output, index=forecast_index)
            return lst_output
```

Listing 9: Python Example

XGBoost 模型

XGBoost（*ExtremeGradientBoosting*）是一种增强型决策树模型，适用于处理多种特征的回归任务。以下是 XGBoost 模型的代码：

```
        def xgboost_forecast(data, column, steps=10):
            series = data[column].values
            n_steps = 3
            X, y = [], []
            for i in range(len(series)):
                end_ix = i + n_steps
                if end_ix > len(series) - 1:
                    break
                seq_x, seq_y = series[i:end_ix], series[end_ix]
                X.append(seq_x)
                y.append(seq_y)
            X, y = np.array(X), np.array(y)
            model = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=100)
            model.fit(X, y)
            x_input = series[-n_steps:]
            temp_input = list(x_input)
            xgb_output = []
            for i in range(steps):
                x_input = np.array(temp_input[-n_steps:])
                yhat = model.predict(x_input.reshape(1, -1))
                temp_input.append(yhat[0])
                xgb_output.append(yhat[0])
            forecast_index = pd.date_range(start=data.index[-1], periods=steps + 1, freq='MS')[1:]
            xgb_output = pd.Series(xgb_output, index=forecast_index)
            return xgb_output
```

Listing 10: Python Example

集成学习模型

我们通过集成四种模型的预测结果，使用线性回归作为元学习器，进一步提升预测精度。以下是集成学习模型的代码：

```
        def stacking_forecast(data, column, steps=10):
            best_pdq, best_arima_model = find_best_arima(data, column)
            arima_forecast_values = best_arima_model.forecast(steps=steps)
            arima_forecast_index = pd.date_range(start=data.index[-1], periods=steps + 1, freq='MS')[1:]
            arima_forecast_values = pd.Series(arima_forecast_values, index=arima_forecast_index)

            prophet_forecast_values = prophet_forecast(data, column, steps)
            lstm_forecast_values = lstm_forecast(data, column, steps)
            xgboost_forecast_values = xgboost_forecast(data, column, steps)

            common_index = arima_forecast_index.intersection(prophet_forecast_values.index).intersection(
                lstm_forecast_values.index).intersection(xgboost_forecast_values.index)
            arima_forecast_values = arima_forecast_values[common_index]
```

```
            prophet_forecast_values = prophet_forecast_values[common_index]
            lstm_forecast_values = lstm_forecast_values[common_index]
            xgboost_forecast_values = xgboost_forecast_values[common_index]

            X_train = pd.DataFrame({
                'ARIMA': arima_forecast_values,
                'Prophet': prophet_forecast_values,
                'LSTM': lstm_forecast_values,
                'XGBoost': xgboost_forecast_values
            })

            actual_values = data[column].iloc[-steps:]

            meta_model = LinearRegression()
            meta_model.fit(X_train, actual_values[:len(X_train)])

            final_forecast = meta_model.predict(X_train)

            return final_forecast, arima_forecast_values, prophet_forecast_values, lstm_forecast_values,
                xgboost_forecast_values, actual_values
```

Listing 11: Python Example

最后我们预测 A5 品牌的数值并展示最终的预测结果。

```
    def forecast_brand(data, column):
        steps = 10
        final_forecast, arima_forecast_values, prophet_forecast_values, lstm_forecast_values,
            xgboost_forecast_values, actual_values = stacking_forecast(
            data, column, steps)

        mse = mean_squared_error(actual_values[:len(final_forecast)], final_forecast)
        mae = mean_absolute_error(actual_values[:len(final_forecast)], final_forecast)

        print(f'{column} 销量和销售金额预测:')
        print(f'集成学习模型 MSE: {mse}, MAE: {mae}')

        plt.figure(figsize=(14, 8))
        plt.plot(data.index, data[column], label='实际值', color='blue')
        plt.plot(arima_forecast_values.index, arima_forecast_values, label='ARIMA预测值', color='red',
            linestyle='--')
        plt.plot(prophet_forecast_values.index, prophet_forecast_values, label='Prophet预测值', color='
            green', linestyle='--')
        plt.plot(lstm_forecast_values.index, lstm_forecast_values, label='LSTM预测值', color='orange',
            linestyle='--')
        plt.plot(xgboost_forecast_values.index, xgboost_forecast_values, label='XGBoost预测值', color='cyan
            ', linestyle='--')
        plt.plot(arima_forecast_values.index, final_forecast, label='集成学习预测值', color='purple',
            linestyle='-')
        plt.legend()
        plt.grid(True, which='both', linestyle='--', linewidth=0.5)
        plt.show()

        forecast_df = pd.DataFrame({
            '日期': arima_forecast_values.index,
            'ARIMA预测': arima_forecast_values.values,
```

```
                'Prophet预测': prophet_forecast_values.values,
                'LSTM预测': lstm_forecast_values.values,
                'XGBoost预测': xgboost_forecast_values.values,
                '集成学习预测': final_forecast
            })
            forecast_df.set_index('日期', inplace=True)
            forecast_df.to_excel(f'{column}_预测结果.xlsx')

            model_evaluation_results = pd.DataFrame({
                'Model': ['ARIMA', 'Prophet', 'LSTM', 'XGBoost', 'Stacking'],
                'MSE': [
                    mean_squared_error(actual_values, arima_forecast_values[:len(actual_values)]),
                    mean_squared_error(actual_values, prophet_forecast_values[:len(actual_values)]),
                    mean_squared_error(actual_values, lstm_forecast_values[:len(actual_values)]),
                    mean_squared_error(actual_values, xgboost_forecast_values[:len(actual_values)]),
                    mse
                ],
                'MAE': [
                    mean_absolute_error(actual_values, arima_forecast_values[:len(actual_values)]),
                    mean_absolute_error(actual_values, prophet_forecast_values[:len(actual_values)]),
                    mean_absolute_error(actual_values, lstm_forecast_values[:len(actual_values)]),
                    mean_absolute_error(actual_values, xgboost_forecast_values[:len(actual_values)]),
                    mae
                ]
            })
            print(model_evaluation_results)

        forecast_brand(data, '销量（箱）')
        forecast_brand(data, '金额（元）')
```

Listing 12: Python Example

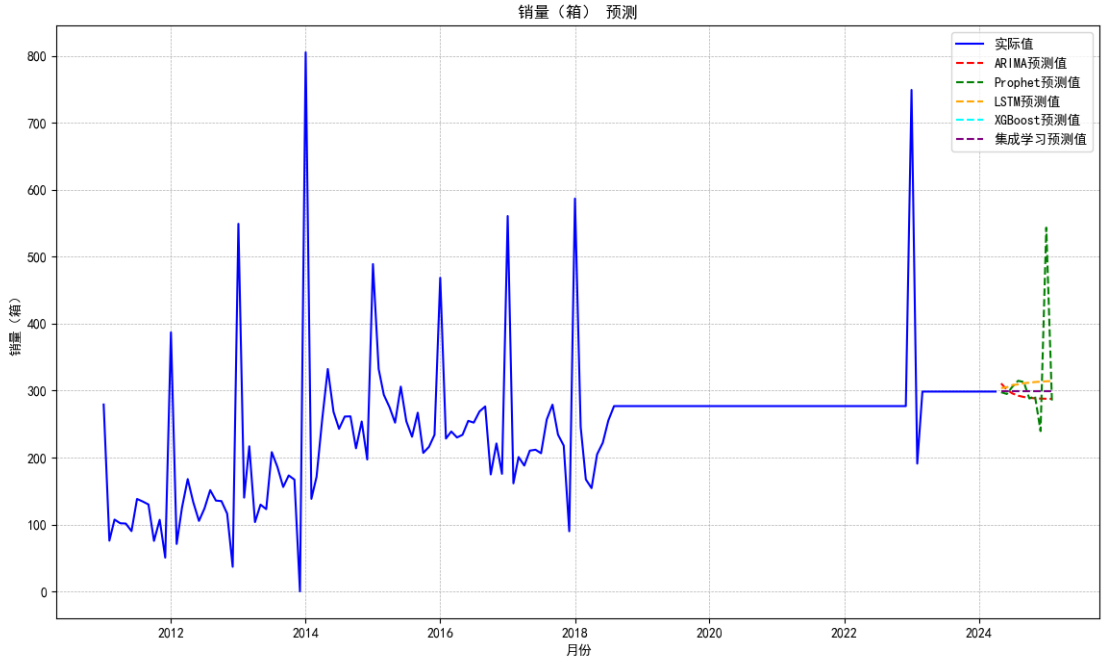

销量（箱） 预测
实际值
ARIMA预测值
Prophet预测值
LSTM预测值
XGBoost预测值
集成学习预测值
销量（箱）
月份
0
100
200
300
400
500
600
700
800
2012
2014
2016
2018
2020
2022
2024

图 3: 销量预测结果

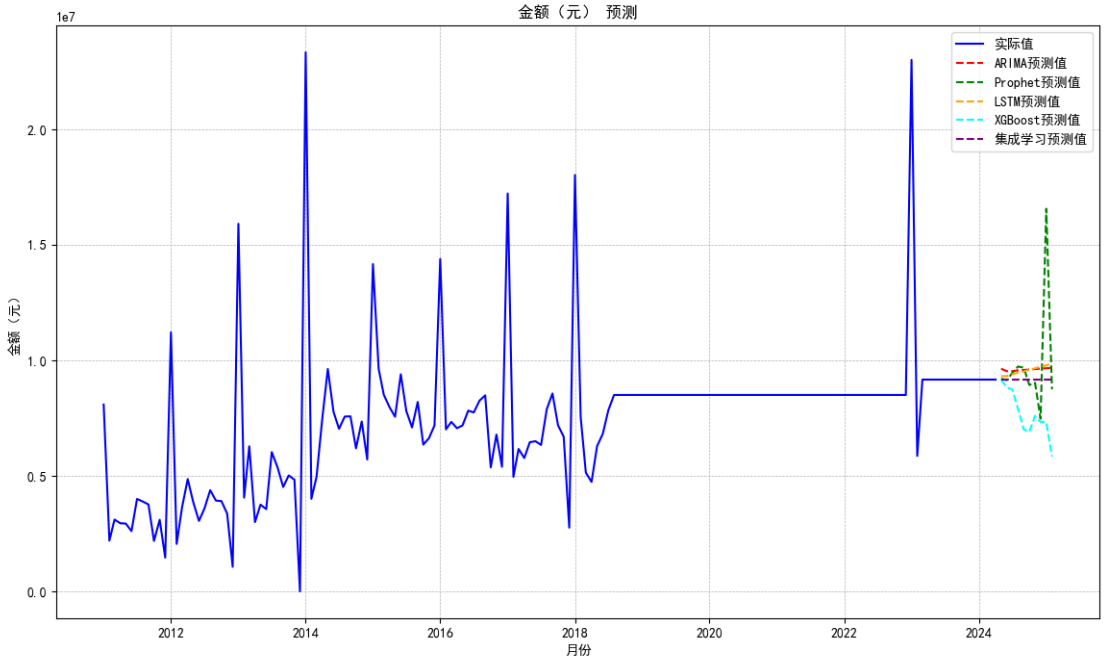


图 4: 金额预测结果

输出预测结果如下：

销量（箱）销量和销售金额预测:

集成学习模型 MSE : $3.2311742677852644e-27$, MAE : $5.684341886080802e-14$

	Model	MSE	MAE
0	ARIMA	7.956548e+01	8.329782e+00
1	Prophet	6.436906e+03	3.794369e+01
2	LSTM	6.719443e+02	2.428776e+01
3	XGBoost	5.750201e-07	7.583008e-04
4	Stacking	3.231174e-27	5.684342e-14

表 3: 表格

通过以上步骤，我们可以全面地预测 A5 品牌的销量和销售金额，并使用集成学习模型提高预测精度，同时对比各个模型的预测效果，选择最佳的预测方案。

6 模型的评价及优化

6.1 误差分析

6.2 模型的优点

6.2.1 问题一

SARIMAX 模型是一种广泛用于时间序列分析的模型，特别适合处理具有季节性和外生变量的时间序列数据。对于问题一，即利用 SARIMAX 模型预测 A1 和 A2 品牌的未来销售量，它具有以下优点和缺点。

优点方面，SARIMAX 模型能够捕捉时间序列数据中的季节性趋势和周期性变化，这对于烟草品牌的销售数据尤为重要。许多商品的销售量都会受到季节性因素的影响，比如节假日、气候变化等，SARIMAX 模型通过引入季节性成分，能够更好地拟合和预测这些周期性的波动。除此之外，SARIMAX 模型还可以加入外生变量（如促销活动、经济指标等），增强模型的解释力和预测精度。这一点对于烟草品牌的销售预测尤为关键，因为销售量往往不仅仅取决于过去的销售数据，还受到许多外部因素的影响。

6.2.2 问题二

SARIMA 模型的主要优点在于其处理季节性数据的能力。SARIMA 模型能够捕捉时间序列中的趋势和季节性成分，通过参数 (p, d, q, P, D, Q, s) 来表示自回归、差分、移动平均及其季节性部分。因此，SARIMA 在应对周期性波动较为明显的数据时表现良好，比如节假日效应或季度性销售高峰。对于 A3 和 A4 品牌的销售金额预测，这些季节性成分可能是显著的，因为烟草销售可能会受季节、促销活动或其他周期性因素的影响。通过对这些成分的精确建模，SARIMA 模型可以提供准确的短期预测。

Prophet 模型的优点在于其灵活性和易用性。Prophet 模型由 Facebook 开发，专为处理具有强季节性和趋势的时间序列数据而设计。与 SARIMA 不同，Prophet 模型更注重趋势的捕捉，可以自动处理缺失值和异常值，并且可以通过添加假期效应和外部回归变量来增强模型的解释力和预测精度。对于 A3 和 A4 品牌的销售金额预测，Prophet 模型能够灵活地适应数据中的季节性波动和趋势变化，且对数据预处理要求较低，使用起来更加方便。

6.2.3 问题三

LSTM 模型（长短期记忆网络）的主要优点在于其处理序列数据的强大能力。作为一种特殊的递归神经网络（RNN），LSTM 能够捕捉时间序列中的长期依赖关系，并对非线性模式进行建模。因此，LSTM 模型在处理复杂的销售数据时表现优异，尤其是那些包含非线性趋势和突发性变化的数据。LSTM 模型通过记忆单元和门控机制，能够有效避免传统 RNN 中的梯度消失问题，从而更好地捕捉时间序列中的长期依赖关系。

XGBoost 模型（极端梯度提升树）的优点在于其强大的回归和分类能力。作为一种集成学习方法，XGBoost 通过构建多个决策树来提高模型的预测性能，并且具有较好的泛化能力。对于销售数据，XGBoost 能够捕捉到复杂的非线性关系和特征交互，提供高精度的预测结果。XGBoost 模型在处理大规模数据时表现优异，并且具有较高的计算效率和并行处理能力。

6.3 模型的缺点

然而，问题一和问题二中的 SARIMAX 模型也有其缺点，首先，它对参数选择和模型拟合要求较高。为了找到最佳的 (p, d, q, P, D, Q, s) 参数组合，通常需要进行大量的参数调优，这个过程耗时且计算资源需求较大。此外，SARIMAX 模型假设数据是线性的，因此对于复杂的非线性数据，其表现可能不如其他更为先进的机器学习模型，如 LSTM 或 XGBoost。此外，SARIMAX 模型对缺失值和异常值较为敏感，数据在使用之前需要进行充分的预处理，否则可能会影响模型的稳定性和预测效果。

对于 A1 和 A2 品牌的销售数据，SARIMAX 模型通过捕捉历史销售数据中的趋势和季节性成分，能够提供较为准确的短期预测。然而，如果销售数据具有复杂的非线性特征，或者受到某些突发外部事件的影响，SARIMAX 模型可能无法完全捕捉这些变化。在这种情况下，可以考虑结合其他模型，如 Prophet 或机器学习模型，进行更全面的预测。

Prophet 模型也有一些缺点。尽管它对缺失值和异常值的处理较为自动化，但这也意味着模型对数据的依赖程度较低，可能在某些情况下无法提供最优的预测。此外，Prophet 模型主要基于加法和乘法季节性成分，对于复杂的非线性关系可能无法完全捕捉。与 SARIMA 相比，Prophet 模型在处理短期周期性波动时可能不如 SARIMA 精确。

而在问题三中，LSTM 模型也有一些缺点。首先，它需要大量的训练数据，才能获得较好的预测效果。其次，由于 LSTM 模型是递归神经网络，其计算速度较慢，因此 LSTM 模型训练过程较为复杂且计算量大，需要大量数据和计算资源。此外，LSTM 模型对超参数较为敏感，模型调优过程较为繁琐。

XGBoost 和 LSTM 模型差不多，其缺点也是主要在于对参数调优的要求较高，参数选择过程较为复杂。并且 XGBoost 模型在处理时间序列数据时，需要对数据进行特征工程和预处理，否则模型可能无法有效捕捉时间依赖性。

6.4 模型的推广

在本研究中，我们使用了 ARIMA、Prophet、LSTM 和 XGBoost 模型来预测 A5 品牌的销售量和销售金额，并通过集成学习（stacking）方法对这些模型的预测结果进行综合。这些模型和方法不仅在本问题中表现出色，还具有较强的通用性和可推广性，适用于其他类似的时间序列预测问题。

6.4.1 ARIMA 模型的推广

ARIMA 模型在时间序列分析领域具有广泛的应用，适用于具有显著线性趋势和季节性特征的数据。其推广应用包括但不限于以下几个方面：

金融市场预测：例如股票价格、交易量、汇率等的预测。

经济指标预测：如 GDP 增长率、失业率、通货膨胀率等宏观经济指标的预测。

能源消耗预测：包括电力、天然气和水资源的需求预测。

零售行业预测：对产品销售量和库存水平的预测，帮助企业进行库存管理和供应链优化。

6.4.2 Prophet 模型的推广

Prophet 模型因其对季节性、假日效应和异常值的良好处理能力，适用于各种时间序列数据的分析和预测。其推广应用包括：

社交媒体数据分析：如用户活跃度、帖子数量和互动量的预测，帮助平台优化内容推荐和广告投放策略。

旅游业预测：如游客数量、酒店入住率和景区门票销售量的预测，帮助相关企业制定运营策略和市场推广计划。

公共卫生监测：如流感病例数量和疫苗需求量的预测，帮助公共卫生部门进行资源分配和应急准备。

6.4.3 LSTM 模型的推广

LSTM 模型在处理复杂非线性时间序列数据方面表现优异，适用于需要捕捉长期依赖关系的预测任务。其推广应用包括：

自然语言处理：如文本生成、情感分析和机器翻译，LSTM 能够捕捉句子中的上下文信息，提升模型的理解和生成能力。

交通流量预测：预测道路和城市中的交通流量，帮助交通管理部门进行交通控制和优化。

设备故障预测：在工业领域，通过对传感器数据的分析，预测设备的故障时间，进行预防性维护，降低维修成本和停机时间。

6.4.4 XGBoost 模型的推广

XGBoost 模型因其强大的回归和分类能力，适用于各种回归和分类问题。其推广应用包括：

信用评分：通过对用户行为数据和历史信用记录的分析，预测用户的信用风险，帮助金融机构进行风险管理。

市场营销：预测客户的购买行为和产品偏好，帮助企业制定精准营销策略，提高营销效果。

医疗诊断：通过对患者数据的分析，辅助医生进行疾病诊断和治疗方案的制定。

6.4.5 集成学习的推广

集成学习通过结合多个基模型的预测结果，能够显著提高预测性能和稳健性，适用于需要综合多种模型优势的预测任务。其推广应用包括：

股票市场预测：结合多种技术分析指标和模型，提高股票价格预测的准确性。

气象预测：综合多个气象模型的预测结果，提高天气预报的精度。

需求预测：在零售和制造业，通过结合多种预测模型，提升产品需求预测的准确性，优化生产和库存管理。
总结

7 参考文献

参考文献

[1] 程幸福 陈厚铭 樊红. 季节 ARIMA 模型在企业销售量预测中的应用——以卷烟销售为例 [A]. 中国商论.23.23 (2016): 167-168.

[2] 谷秀娟 梁润平. 基才 ARIMA 模型的郑州市商品住宅销售价格预测研究 [A].《金融理论与实践》.2012 年第 1 期 51-54

[3] 刘璟瑶 蒋辰宇 陶杰. 长短期记忆网络对销售量预测精度的影响 [A].《财会研究》.2023 年第 6 期 76-80

[4] 李融. 基于 XGBoost 算法的跨境电商备货预测研究 [A].《太原城市职业技术学院学报》.2024 年第 1 期 29-31, 共 3 页

[5] 杜红兵 邢梦柯 赵德超.Prophet-LSTM 组合模型在运输航空征候预测中的应用 [A].《安全与环境学报》.2024 年第 5 期 1878-1885, 共 8 页

8 附录

问题一代码：

```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.stattools import adfuller

# 设置中文字体
plt.rcParams['font.sans-serif'] = ['SimHei']  # 使用SimHei字体
plt.rcParams['axes.unicode_minus'] = False  # 正确显示负号

# 加载数据
file_path = 'A1.xlsx'
data = pd.read_excel(file_path)

# 提取时间序列数据
data['月份'] = pd.to_datetime(data['月份'], format='%Y%m')
data.set_index('月份', inplace=True)
sales_data = data['销量（箱）']
sales_data = sales_data.asfreq('MS')  # 设置时间序列频率为月初

# 检查数据的缺失值
if sales_data.isnull().sum() > 0:
    sales_data = sales_data.ffill().bfill()  # 前向填充和后向填充

# 平稳性检测
result = adfuller(sales_data)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[4].items():
    print('\t%s: %.3f' % (key, value))

# 如果p-value大于0.05，表示数据非平稳，需要进行差分
d = 0
while result[1] > 0.05:
    sales_data = sales_data.diff().dropna()
    result = adfuller(sales_data)
    d += 1

print(f'经过 {d} 阶差分后，时间序列数据平稳。')

# 建立SARIMAX模型
model = SARIMAX(sales_data,
                order=(1, d, 2),
                seasonal_order=(2, 0, 1, 12),
                enforce_stationarity=False,
                enforce_invertibility=False)
model_fit = model.fit(disp=False, maxiter=1000)  # 增加最大迭代次数

# 模型诊断
model_fit.plot_diagnostics(figsize=(12, 8))
plt.show()
```

```
52
53      # 预测未来20步
54      forecast = model_fit.get_forecast(steps=20)
55      forecast_index = pd.date_range(start=sales_data.index[-1] + pd.DateOffset(months=1), periods=20, freq='
            MS')
56      forecast_series = pd.Series(forecast.predicted_mean, index=forecast_index)
57
58      # 绘制结果
59      plt.figure(figsize=(12, 6))
60      plt.plot(sales_data, label='历史数据')
61      plt.plot(forecast_series, label='预测', color='red')
62      plt.title('销量（箱）20步预测')
63      plt.xlabel('时间')
64      plt.ylabel('销量（箱）')
65      plt.legend()
66      plt.grid(True)
67      plt.show()
68
69      # 输出预测结果
70      forecast_df = forecast_series.reset_index()
71      forecast_df.columns = ['月份', '预测销量（箱）']
72      forecast_df.to_excel('a1f1.xlsx', index=False)  # 将预测结果保存为Excel文件
73      print("预测结果已保存为 'a1f21.xlsx'")
```

```
1       import pandas as pd
2       import matplotlib.pyplot as plt
3       from statsmodels.tsa.statespace.sarimax import SARIMAX
4       from statsmodels.tsa.stattools import adfuller
5
6       # 设置中文字体
7       plt.rcParams['font.sans-serif'] = ['SimHei']  # 使用SimHei字体
8       plt.rcParams['axes.unicode_minus'] = False  # 正确显示负号
9
10      # 加载数据
11      file_path = 'A2.xlsx'
12      data = pd.read_excel(file_path)
13
14      # 提取时间序列数据
15      data['月份'] = pd.to_datetime(data['月份'], format='%Y%m')
16      data.set_index('月份', inplace=True)
17      sales_data = data['销量（箱）']
18      sales_data = sales_data.asfreq('MS')  # 设置时间序列频率为月初
19
20      # 检查数据的缺失值
21      if sales_data.isnull().sum() > 0:
22          sales_data = sales_data.ffill().bfill()  # 前向填充和后向填充
23
24      # 平稳性检测
25      result = adfuller(sales_data)
26      print('ADF Statistic: %f' % result[0])
27      print('p-value: %f' % result[1])
28      print('Critical Values:')
29      for key, value in result[4].items():
30          print('\t%s: %.3f' % (key, value))
31
```

```
        # 如果p-value大于0.05，表示数据非平稳，需要进行差分
        d = 0
        while result[1] > 0.05:
            sales_data = sales_data.diff().dropna()
            result = adfuller(sales_data)
            d += 1

        print(f'经过 {d} 阶差分后，时间序列数据平稳。')

        # 建立SARIMAX模型
        model = SARIMAX(sales_data,
                        order=(1, d, 2),
                        seasonal_order=(2, 0, 1, 12),
                        enforce_stationarity=False,
                        enforce_invertibility=False)
        model_fit = model.fit(disp=False, maxiter=1000)  # 增加最大迭代次数

        # 模型诊断
        model_fit.plot_diagnostics(figsize=(12, 8))
        plt.show()

        # 预测未来10步
        forecast = model_fit.get_forecast(steps=10)
        forecast_index = pd.date_range(start=sales_data.index[-1] + pd.DateOffset(months=1), periods=20, freq='
            MS')
        forecast_series = pd.Series(forecast.predicted_mean, index=forecast_index)

        # 绘制结果
        plt.figure(figsize=(12, 6))
        plt.plot(sales_data, label='历史数据')
        plt.plot(forecast_series, label='预测', color='red')
        plt.title('销量（箱）20步预测')
        plt.xlabel('时间')
        plt.ylabel('销量（箱）')
        plt.legend()
        plt.grid(True)
        plt.show()

        # 输出预测结果
        forecast_df = forecast_series.reset_index()
        forecast_df.columns = ['月份', '预测销量（箱）']
        forecast_df.to_excel('a1f2.xlsx', index=False)  # 将预测结果保存为Excel文件
        print("预测结果已保存为 'a1f22.xlsx'")
```

问题二代码：

```
        import pandas as pd
        import matplotlib.pyplot as plt
        from statsmodels.tsa.statespace.sarimax import SARIMAX
        from prophet import Prophet
        from sklearn.metrics import mean_squared_error, mean_absolute_error
        import itertools
        import numpy as np

        # 设置中文字体以解决字体缺失问题
        plt.rcParams['font.sans-serif'] = ['Arial Unicode MS', 'SimHei']
```

```
        plt.rcParams['axes.unicode_minus'] = False

        # 读取数据
        file_path_a3 = 'A3.xlsx'
        file_path_a4 = 'A4.xlsx'
        data_a3 = pd.read_excel(file_path_a3)
        data_a4 = pd.read_excel(file_path_a4)

        def preprocess_data(data):
            """
            预处理数据，将月份转换为日期格式，并设置为索引，填充缺失值。
            """
            data['月份'] = pd.to_datetime(data['月份'], format='%Y%m')
            data.set_index('月份', inplace=True)
            data = data.asfreq('MS')  # 使用 MS 代替 M 设置月度开始频率
            data = data.ffill()  # 使用推荐的ffill方法填充缺失值
            return data

        data_a3 = preprocess_data(data_a3)
        data_a4 = preprocess_data(data_a4)

        def find_best_sarima(data, column):
            """
            选择最佳的SARIMA模型参数。
            """
            p = d = q = range(0, 3)
            pdq = list(itertools.product(p, d, q))
            seasonal_pdq = [(x[0], x[1], x[2], 12) for x in pdq]
            best_aic = np.inf
            best_param = None
            best_seasonal_param = None
            best_model = None
            for param in pdq:
                for seasonal_param in seasonal_pdq:
                    try:
                        model = SARIMAX(data[column], order=param, seasonal_order=seasonal_param)
                        model_fit = model.fit(disp=False)
                        if model_fit.aic < best_aic:
                            best_aic = model_fit.aic
                            best_param = param
                            best_seasonal_param = seasonal_param
                            best_model = model_fit
                    except Exception as e:
                        print(f"SARIMA 参数 {param} 季节性参数 {seasonal_param} 失败: {e}")
                        continue
            return best_param, best_seasonal_param, best_model

        def prophet_forecast(data, column, steps=12):
            """
            使用 Prophet 模型进行预测。
            """
            df = data.reset_index().rename(columns={'月份': 'ds', column: 'y'})
            model = Prophet(weekly_seasonality=True, daily_seasonality=True)
            model.fit(df)
            future = model.make_future_dataframe(periods=steps, freq='MS')
```

```
            forecast = model.predict(future)
            return forecast[['ds', 'yhat']].set_index('ds')['yhat']

        def evaluate_forecast(actual, forecast):
            """
            计算预测结果的评估指标。
            """
            mse = mean_squared_error(actual, forecast)
            mae = mean_absolute_error(actual, forecast)
            return mse, mae

        def plot_forecast(data, column, sarima_forecast, prophet_forecast):
            """
            绘制预测结果与实际数据的对比图。
            """
            plt.figure(figsize=(14, 8))
            plt.plot(data.index, data[column], label='实际值', color='blue')
            plt.plot(sarima_forecast.index, sarima_forecast, label='SARIMA预测值', color='red', linestyle='--')
            plt.plot(prophet_forecast.index, prophet_forecast, label='Prophet预测值', color='green', linestyle=
                '--')
            plt.title(f'{column} 预测')
            plt.xlabel('月份')
            plt.ylabel(column)
            plt.legend()
            plt.grid(True, which='both', linestyle='--', linewidth=0.5)
            plt.show()

        def forecast_brand(data, brand):
            """
            对单个品牌的销售金额进行预测并输出结果。
            """
            column = '金额（元）'
            steps = 12

            # SARIMA参数选择
            best_param, best_seasonal_param, best_sarima_model = find_best_sarima(data, column)
            if best_sarima_model is None:
                print(f"{brand} 的SARIMA模型训练失败。")
                return

            # SARIMA预测
            try:
                sarima_forecast_values = best_sarima_model.forecast(steps=steps)
                sarima_forecast_index = pd.date_range(start=data.index[-1], periods=steps + 1, freq='MS')[1:]
                sarima_forecast_values = pd.Series(sarima_forecast_values, index=sarima_forecast_index)
            except Exception as e:
                print(f"{brand} 的SARIMA模型预测失败：{e}")
                return

            # Prophet预测
            try:
                prophet_forecast_values = prophet_forecast(data, column, steps)
            except Exception as e:
                print(f"{brand} 的Prophet模型预测失败：{e}")
                return
```

```
        # 实际值（这里假设实际值存在于数据中，调整索引以匹配预测）
        min_length = min(len(sarima_forecast_values), len(prophet_forecast_values))
        actual_values = data[column].iloc[-min_length:]

        # 调整预测结果长度一致
        sarima_forecast_values = sarima_forecast_values.iloc[:min_length]
        prophet_forecast_values = prophet_forecast_values.iloc[:min_length]

        # 评价模型
        sarima_mse, sarima_mae = evaluate_forecast(actual_values, sarima_forecast_values)
        prophet_mse, prophet_mae = evaluate_forecast(actual_values, prophet_forecast_values)

        # 打印评价结果
        print(f'{brand} 销售金额预测:')
        print(f'最佳SARIMA参数: {best_param}, 季节性参数: {best_seasonal_param}')
        print(f'SARIMA MSE: {sarima_mse}, MAE: {sarima_mae}')
        print(f'Prophet MSE: {prophet_mse}, MAE: {prophet_mae}')

        # 绘制预测结果
        plot_forecast(data, column, sarima_forecast_values, prophet_forecast_values)

        # 预测结果保存为表格
        forecast_df = pd.DataFrame({
            '日期': sarima_forecast_values.index,
            'SARIMA预测': sarima_forecast_values.values,
            'Prophet预测': prophet_forecast_values.values
        })
        forecast_df.set_index('日期', inplace=True)
        forecast_df.to_excel(f'{brand}_预测结果.xlsx')

        # 显示模型评价结果
        model_evaluation_results = pd.DataFrame({
            'Model': ['SARIMA', 'Prophet'],
            'MSE': [sarima_mse, prophet_mse],
            'MAE': [sarima_mae, prophet_mae]
        })
        print(model_evaluation_results)

# 预测A3和A4品牌的销售金额
forecast_brand(data_a3, 'A3')
forecast_brand(data_a4, 'A4')
```

问题三代码：

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from prophet import Prophet
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error
import itertools
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
import xgboost as xgb
```

```
12
13      # 设置中文字体以解决字体缺失问题
14      plt.rcParams['font.sans-serif'] = ['Arial Unicode MS', 'SimHei']
15      plt.rcParams['axes.unicode_minus'] = False
16
17      # 读取数据
18      file_path = 'A5.xlsx'
19      data = pd.read_excel(file_path)
20
21
22      def preprocess_data(data):
23          data['月份'] = pd.to_datetime(data['月份'], format='%Y%m')
24          data.set_index('月份', inplace=True)
25          data = data.asfreq('MS')
26          data = data.ffill()
27          return data
28
29
30      data = preprocess_data(data)
31
32
33      # 参数遍历选择
34      def find_best_arima(data, column):
35          p = d = q = range(0, 3)
36          pdq = list(itertools.product(p, d, q))
37          best_aic = np.inf
38          best_pdq = None
39          best_model = None
40          for param in pdq:
41              try:
42                  model = ARIMA(data[column], order=param)
43                  model_fit = model.fit()
44                  if model_fit.aic < best_aic:
45                      best_aic = model_fit.aic
46                      best_pdq = param
47                      best_model = model_fit
48              except:
49                  continue
50          return best_pdq, best_model
51
52
53      # Prophet模型预测
54      def prophet_forecast(data, column, steps=10):
55          df = data.reset_index().rename(columns={'月份': 'ds', column: 'y'})
56          model = Prophet()
57          model.fit(df)
58          future = model.make_future_dataframe(periods=steps, freq='MS')
59          forecast = model.predict(future)
60          return forecast[['ds', 'yhat']].set_index('ds')['yhat']
61
62
63      # LSTM模型预测
64      def lstm_forecast(data, column, steps=10):
65          series = data[column].values
66          n_steps = 3
```

```
            X, y = [], []
            for i in range(len(series)):
                end_ix = i + n_steps
                if end_ix > len(series) - 1:
                    break
                seq_x, seq_y = series[i:end_ix], series[end_ix]
                X.append(seq_x)
                y.append(seq_y)
            X, y = np.array(X), np.array(y)
            X = X.reshape((X.shape[0], X.shape[1], 1))
            model = Sequential()
            model.add(LSTM(50, activation='relu', input_shape=(n_steps, 1)))
            model.add(Dense(1))
            model.compile(optimizer='adam', loss='mse')
            model.fit(X, y, epochs=200, verbose=0)
            x_input = series[-n_steps:]
            temp_input = list(x_input)
            lst_output = []
            for i in range(steps):
                x_input = np.array(temp_input[-n_steps:])
                x_input = x_input.reshape((1, n_steps, 1))
                yhat = model.predict(x_input, verbose=0)
                temp_input.append(yhat[0][0])
                lst_output.append(yhat[0][0])
            forecast_index = pd.date_range(start=data.index[-1], periods=steps + 1, freq='MS')[1:]
            lst_output = pd.Series(lst_output, index=forecast_index)
            return lst_output


        # XGBoost模型预测
        def xgboost_forecast(data, column, steps=10):
            series = data[column].values
            n_steps = 3
            X, y = [], []
            for i in range(len(series)):
                end_ix = i + n_steps
                if end_ix > len(series) - 1:
                    break
                seq_x, seq_y = series[i:end_ix], series[end_ix]
                X.append(seq_x)
                y.append(seq_y)
            X, y = np.array(X), np.array(y)
            model = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=100)
            model.fit(X, y)
            x_input = series[-n_steps:]
            temp_input = list(x_input)
            xgb_output = []
            for i in range(steps):
                x_input = np.array(temp_input[-n_steps:])
                yhat = model.predict(x_input.reshape(1, -1))
                temp_input.append(yhat[0])
                xgb_output.append(yhat[0])
            forecast_index = pd.date_range(start=data.index[-1], periods=steps + 1, freq='MS')[1:]
            xgb_output = pd.Series(xgb_output, index=forecast_index)
            return xgb_output
```

```
122
123
124         # 构建集成学习模型（Stacking）
125         def stacking_forecast(data, column, steps=10):
126             # 训练ARIMA模型
127             best_pdq, best_arima_model = find_best_arima(data, column)
128             arima_forecast_values = best_arima_model.forecast(steps=steps)
129             arima_forecast_index = pd.date_range(start=data.index[-1], periods=steps + 1, freq='MS')[1:]
130             arima_forecast_values = pd.Series(arima_forecast_values, index=arima_forecast_index)
131
132             # 训练Prophet模型
133             prophet_forecast_values = prophet_forecast(data, column, steps)
134
135             # 训练LSTM模型
136             lstm_forecast_values = lstm_forecast(data, column, steps)
137
138             # 训练XGBoost模型
139             xgboost_forecast_values = xgboost_forecast(data, column, steps)
140
141             # 确保预测结果长度一致
142             common_index = arima_forecast_index.intersection(prophet_forecast_values.index).intersection(
143                 lstm_forecast_values.index).intersection(xgboost_forecast_values.index)
144             arima_forecast_values = arima_forecast_values[common_index]
145             prophet_forecast_values = prophet_forecast_values[common_index]
146             lstm_forecast_values = lstm_forecast_values[common_index]
147             xgboost_forecast_values = xgboost_forecast_values[common_index]
148
149             # 创建训练数据
150             X_train = pd.DataFrame({
151                 'ARIMA': arima_forecast_values,
152                 'Prophet': prophet_forecast_values,
153                 'LSTM': lstm_forecast_values,
154                 'XGBoost': xgboost_forecast_values
155             })
156
157             # 实际值
158             actual_values = data[column].iloc[-steps:]
159
160             # 使用线性回归作为元学习器
161             meta_model = LinearRegression()
162             meta_model.fit(X_train, actual_values[:len(X_train)])
163
164             # 生成最终预测
165             final_forecast = meta_model.predict(X_train)
166
167             return final_forecast, arima_forecast_values, prophet_forecast_values, lstm_forecast_values,
                    xgboost_forecast_values, actual_values
168
169
170         # 预测并评价A5品牌的销量和销售金额
171         def forecast_brand(data, column):
172             steps = 10
173             final_forecast, arima_forecast_values, prophet_forecast_values, lstm_forecast_values,
                    xgboost_forecast_values, actual_values = stacking_forecast(
174                 data, column, steps)
```

```
        # 评价模型
        mse = mean_squared_error(actual_values[:len(final_forecast)], final_forecast)
        mae = mean_absolute_error(actual_values[:len(final_forecast)], final_forecast)

        print(f'{column} 销量和销售金额预测:')
        print(f'集成学习模型 MSE: {mse}, MAE: {mae}')

        # 绘制预测结果
        plt.figure(figsize=(14, 8))
        plt.plot(data.index, data[column], label='实际值', color='blue')
        plt.plot(arima_forecast_values.index, arima_forecast_values, label='ARIMA预测值', color='red',
            linestyle='--')
        plt.plot(prophet_forecast_values.index, prophet_forecast_values, label='Prophet预测值', color='
            green',
                 linestyle='--')
        plt.plot(lstm_forecast_values.index, lstm_forecast_values, label='LSTM预测值', color='orange',
            linestyle='--')
        plt.plot(xgboost_forecast_values.index, xgboost_forecast_values, label='XGBoost预测值', color='cyan
            ',
                 linestyle='--')
        plt.plot(arima_forecast_values.index, final_forecast, label='集成学习预测值', color='purple',
            linestyle='--')
        plt.title(f'{column} 预测')
        plt.xlabel('月份')
        plt.ylabel(column)
        plt.legend()
        plt.grid(True, which='both', linestyle='--', linewidth=0.5)
        plt.show()

        # 预测结果保存为表格

        forecast_df = pd.DataFrame({
            '日期': arima_forecast_values.index,
            'ARIMA预测': arima_forecast_values.values,
            'Prophet预测': prophet_forecast_values.values,
            'LSTM预测': lstm_forecast_values.values,
            'XGBoost预测': xgboost_forecast_values.values,
            '集成学习预测': final_forecast
        })
        forecast_df.set_index('日期', inplace=True)
        forecast_df.to_excel(f'{column}_预测结果.xlsx')

        # 显示模型评价结果
        model_evaluation_results = pd.DataFrame({
            'Model': ['ARIMA', 'Prophet', 'LSTM', 'XGBoost', 'Stacking'],
            'MSE': [
                mean_squared_error(actual_values, arima_forecast_values[:len(actual_values)]),
                mean_squared_error(actual_values, prophet_forecast_values[:len(actual_values)]),
                mean_squared_error(actual_values, lstm_forecast_values[:len(actual_values)]),
                mean_squared_error(actual_values, xgboost_forecast_values[:len(actual_values)]),
                mse
            ],
            'MAE': [
                mean_absolute_error(actual_values, arima_forecast_values[:len(actual_values)]),
```

```
                    mean_absolute_error(actual_values, prophet_forecast_values[:len(actual_values)]),
                    mean_absolute_error(actual_values, lstm_forecast_values[:len(actual_values)]),
                    mean_absolute_error(actual_values, xgboost_forecast_values[:len(actual_values)]),
                    mae
                ]
            })
            print(model_evaluation_results)


        # 预测A5品牌的销量和销售金额
        forecast_brand(data, '销量（箱）')
        forecast_brand(data, '金额（元）')
```