队伍编号	MC2406053
题号	C

基于随机森林和 PulP 的物流预测和人员安排

摘要

摘要内容

关键词: 随机森林算法、LSTM、ARIMA（SARIMA）、PuLP

目录

1 问题重述

1.1 问题背景

随着网购的流行，电商物流显得更加重要。在电商物流网络中，订单配送的过程包括多个环节，其中核心环节之一是分拣。分拣中心负责根据不同的目的地对包裹进行分类，并将它们发送到下一个目的地，最终交付给顾客。因此，提高分拣中心的管理效率对整个网络的订单交付效率和成本控制至关重要。

货量预测在电商物流网络中扮演着至关重要的角色。准确预测分拣中心的货物量是后续管理和决策的基础。通常，货量预测是根据历史货物量、物流网络配置等信息，来预测每个分拣中心每天和每小时的货物量。

分拣中心的货量预测与网络的运输线路密切相关。通过分析各线路的运输货物量，可以确定各分拣中心之间的网络连接关系。当线路关系发生变化时，可以根据调整信息来提高对各分拣中心货量的准确预测。

基于货量预测的人员排班是下一步需要解决的重要问题。分拣中心的人员包括正式员工和临时工两种类型。合理安排人员旨在完成工作的前提下尽可能降低人力成本。根据物流网络的情况，制定了人员安排的班次和小时人效指标。在确定人员安排时，优先考虑使用正式员工，必要时再增加临时工。

1.2 问题描述

1.2.1 问题一

根据所给的前 3 个月每天的货量数据以及前一个月每一个小时的货量数据建立货量预测模型，对 57 个分拣中心未来 30 天每天及每小时的货量进行预测。

1.2.2 问题二

由于分拣中心之间存在货物运送的关联，在已知过去三个月内的货物运输关联情况下，若运输关系发生变化，预测分拣中心的货量变化。

1.3 总体思路分析

2 问题一的建模与求解

2.1 分析

问题一给出了分拣中心在过去的四个月内每天的货量以及过去 30 天内每个小时的货量。在这些数据的基础上要对下一个月（30 天）的货量进行预测，我们先对给出的数据进行观察。我们选取，例如 SC6,SC41 来观察。利用 python/matplotlib 作图

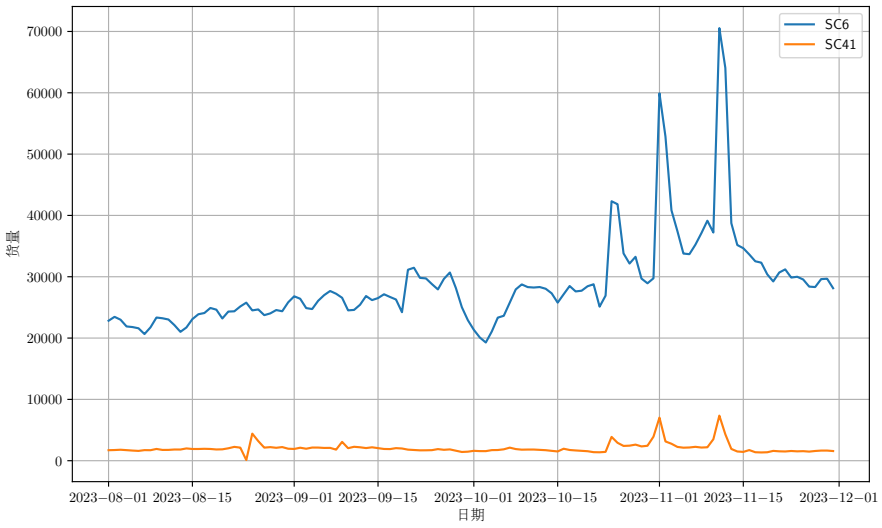

图 1: SC6,SC41 的日期-货量图线

根据观察，我们发现在特定的时间段会出现一些”异常值”，而这些异常值的出现很可能是由于节假日等因素引起的。因此为了预测 30 天后的货物量，我们选取几种深度学习的方法来进行预测。

1. LSTM
2. ARIMA
3. 随机森林算法

2.1.1 LSTM

长短期记忆（英语：Long Short-Term Memory，LSTM）是一种时间循环神经网络（RNN）。由于独特的设计结构，LSTM 适合于处理和预测时间序列中间隔和延迟非常长的重要事件。LSTM 通过引入“门”结构来调节信息的流动，这使得它在长序列数据上表现得更好。LSTM 的核心组件有遗忘门（Forget Gate）、输入门（Input Gate）、细胞状态（Cell State）、输出门（Output Gate）。LSTM 的每个时刻会通过上述四个门控制信息的流动。遗忘门决定丢弃哪些过去的信息，输入门和它的候选值共同决定将哪些新信息加入细胞状态。细胞状态随后更新，最后通过输出门确定应输出什么信息。

通过这样的结构，LSTM 能够在处理序列数据时有效地保留长期依赖信息，解决了传统 RNN 在长序列上的梯度消失或爆炸问题。

2.1.2 ARIMA

ARIMA 模型（自回归积分滑动平均模型）是一种广泛应用于时间序列预测的统计模型。ARIMA 模型结合了自回归（AR）、差分（I）和移动平均（MA）三种主要组成部分，适用于分析和预测具有时间依赖性的数据。其核心组件包括自回归（AR）部分、差分（I）部分、移动平均（MA）部分。将这三部分结合起来，ARIMA 模型的完整形式可以表示为：

$$\phi(B)\nabla^d X_t = \theta(B)a_t$$

其中 $\phi(B)X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} + \cdots + \phi_p X_{t-p}$，$p$ 是自回归项的阶数,ϕ_i 是自回归系数，B 是退后算子。$\theta(B)a_t$ 则类似，a_t 是时间序列的误差项。

从式子中可以看出，通过对原始数据进行差分转换，ARIMA 能够将非平稳时间序列转换为平稳时间序列，因此模型能够适用于预测那些显示出趋势或季节性模式的数据。

2.1.3 随机森林算法

随机森林是一种集成学习方法，特别适合用于分类、回归和其他任务，通过构建多棵决策树在训练时并输出模式的类（分类）或平均预测（回归）。随机森林算法的核心思想是通过合并多个决策树的预测结果来提高整体模型的预测准确性和稳定性。

随机森林算法的主要步骤有：自助采样（Bootstrap sampling），从原始数据集中随机（允许重复的）选择 N 个样本构成了一个训练集。N 次采样产生的 N 个训练集被用于训练 N 棵决策树；构建决策树：对于每棵树的每个节点，随机选择 k 个特征（而不是所有特征），然后使用这些特征中的最佳分割方法来分割节点。这种“特征随机选择”的做法增加了树之间的差异性。决策树的生长：每棵树都尽可能地生长而不进行剪枝。每棵树都完全依赖于自助采样得到的训练数据集来建立。聚合预测：对于分类问题，使用多数投票法；对于回归问题，计算所有树的输出值的平均值。

随机森林算法的成功在于它的简单性和在多种数据集上表现出的高效性与准确性。它既能处理分类问题，也能处理回归问题，同时还能进行特征选择，是一种非常强大且灵活的机器学习算法。虽然随机森林通常不如专门的时间序列或图数据模型那样直接适用于处理分拣中心之间的货物流动问题，但它仍然可以在某些情况下提供有价值的见解和预测，特别是随机森林可以提供关于哪些特征（例如历史货物流量、时间因素、分拣中心之间的距离等）对预测货物流量最有影响的洞察。这对于理解货物流量模式和优化物流网络可能非常有用。

2.2 按天的数据模拟与预测

根据选择，我们对其进行数据模拟。我们通过使用 `python` 的 `sklearn` 中提供的模型来进行拟合。首先我们通过分类不同的仓库，获取所有的信息。在分类完成之后，我们利用几种方法来实现。首先是利用 LSTM 来生成预测的曲线。[1]如图2。

相同的，我们选取随机森林进行预测可以得到图3。另外由于我们发现数据有较大的波动，而 ARIMA 模型适用于平稳时间序列，而从图中看，原始数据不是平稳的。因此直接使用 ARIMA 模型可能不会得到好的预测效果。我们尝试季节性 ARIMA (SARIMA) 模型。

[1]具体代码在附录中。

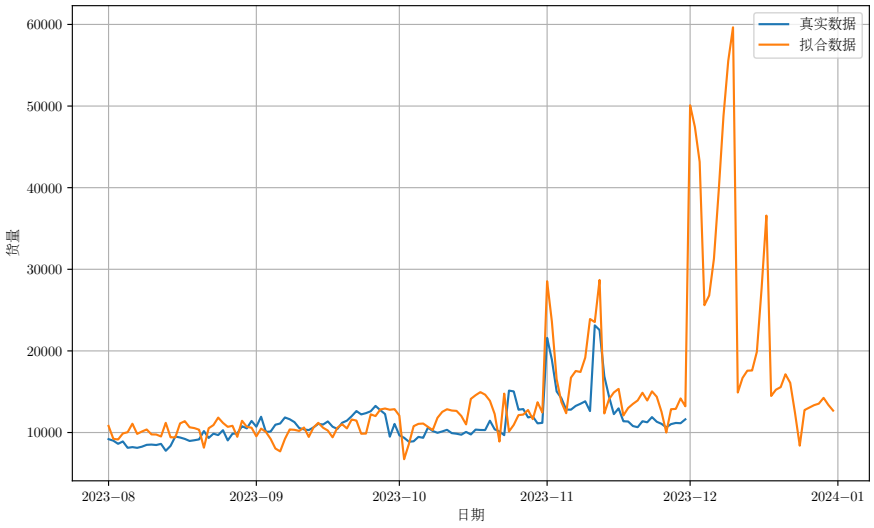

图 2: 利用 LSTM 预测图线示例

经过所输出的平均方差的对比，我们认为使用随机森林算法对于第一题是相对的最优解。因此我们对未来一个月之内每一个小时的货量预测也采取完全相同的方法预测。

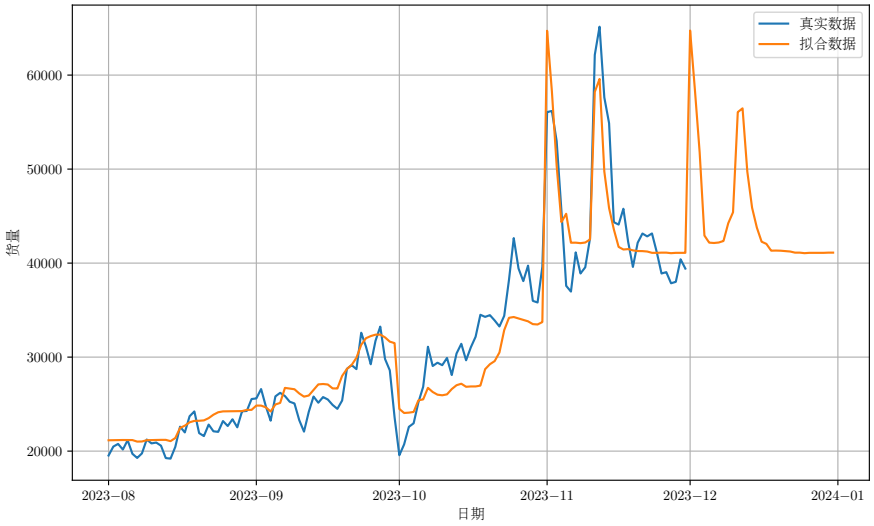

图 3: 利用随机森林预测图线示例（SC10）

2.3 按小时的数据模拟和预测

按照之前的方法，相似的预测之后的数据，如图5。从图中可以看出，拟合效果符合数据波动的周期性。通过计算拟合模型与真实数据之间的平均方差，我们确定使用随机森林算法来拟合数据。

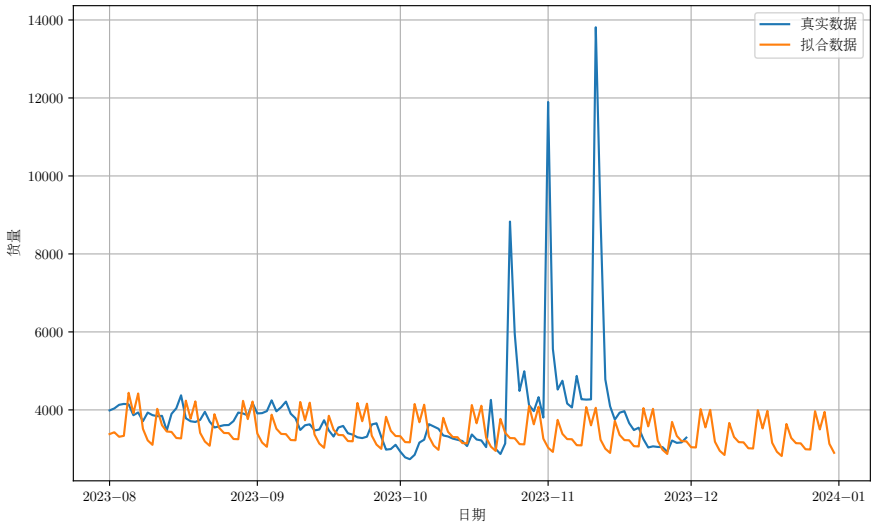

图 4: 利用 SARIMA 预测图线示例（SC55）

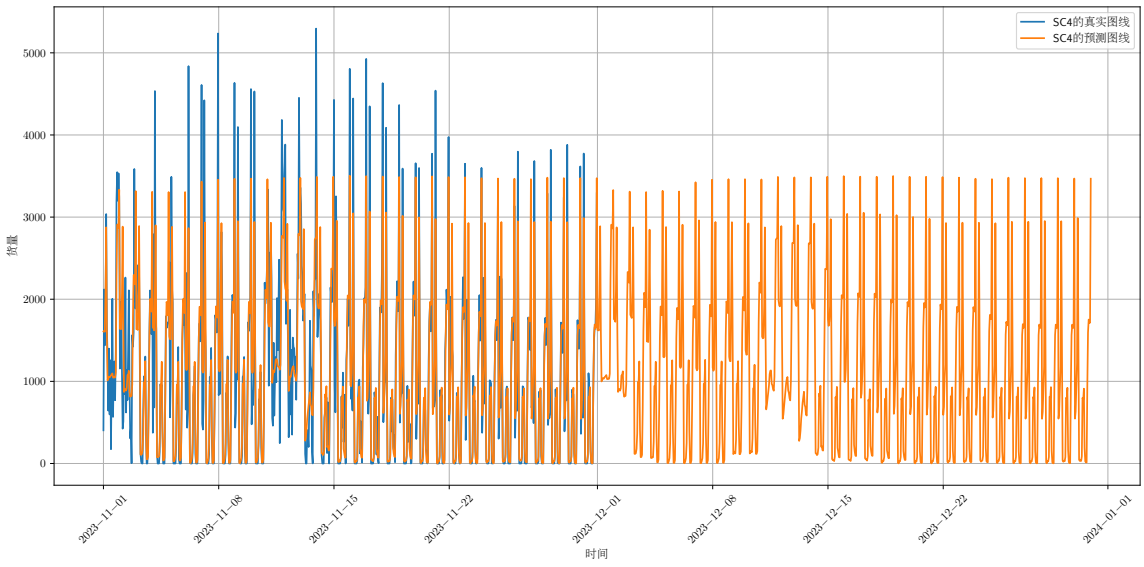

图 5: 利用随机森林模型预测小时图线示例（SC4）

3 问题二的建模与求解

3.1 分析

我们需要对已知的关系进行分析。因此我们首先先将仓库之间的关联进行可视化，我们选用弦图的形式，如图6。在现有的运输关系模型上发生了变动，此时我们需要考虑将数据预处理、特征提取、模型训练和预测。

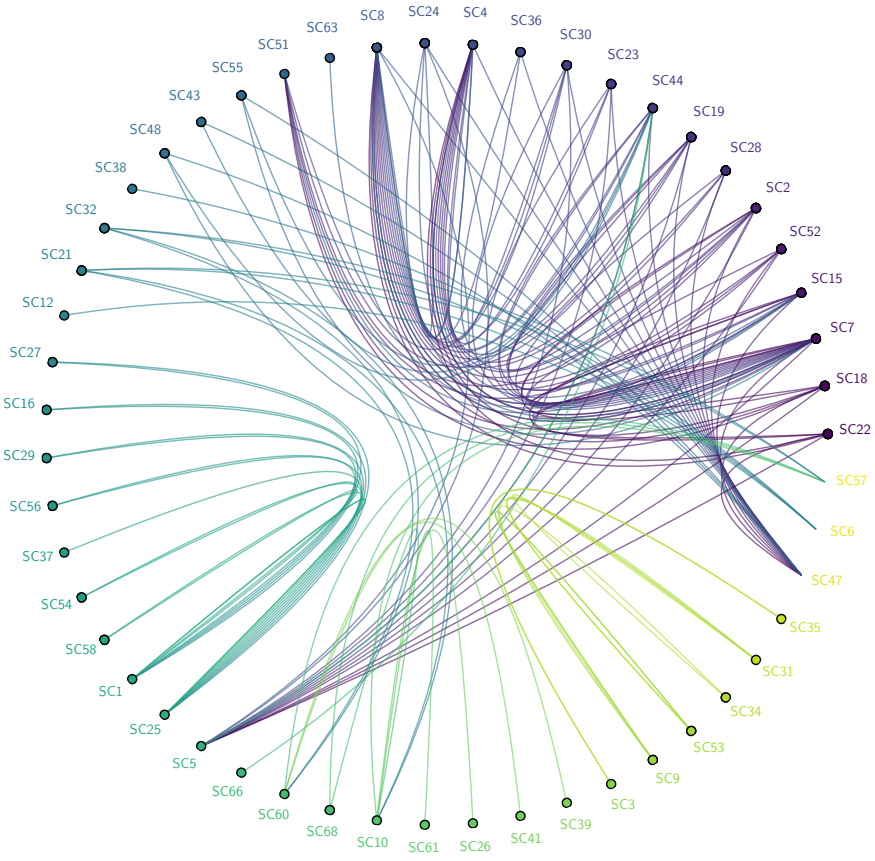

图 6: 分配中心之间的关系弦图

4 问题四的建模与求解

4.1 问题四的分析

这个问题是关于如何调度员工的最优解。这是一个线性规划问题，目标是最小化总人力成本，并同时满足工作班次的需求。下面是关于一些变量的假设。

表 1: 符号说明

符号	说明
$F(i,d,s)$	二元变量，表示固定员工 i 在日期 d 和班次 s 是否工作
$T(d,s)$	整数变量，表示在日期 d 和班次 s 需要的临时工人数
$D(d,s)$	整数变量，表示在日期 d 和班次 s 所需的工作需求

我们的目标函数是

$$\text{Minimize} \quad Z = \sum_{d \in \text{Dates}} \sum_{s \in \text{Shifts}} \left(\sum_{i=1}^{200} F(i,d,s) + T(d,s) \right)$$

即最小化全职和临时工的总人天数。这个最小化函数需要满足以下几个约束条件：

1. 需求满足：

$$\forall d \in \text{Dates}, \forall s \in \text{Shifts} : \sum_{i=1}^{200} F(i,d,s) + 20 \times T(d,s) \geq D(d,s)$$

这个约束确保每个班次的工作需求被满足。这里，25 和 20 是每个工作的产出（或能力）。

2. 正式工出勤率不超过 85%:

$$\forall i : \sum_{d \in \text{Dates}} \sum_{s \in \text{Shifts}} F(i,d,s) \leq 30 \times 0.85$$

此约束限制每个全职员工在一个月内的最大工作天数。

3. 连续工作天数不超过 7 天:

$$\forall i, \forall d \in \text{Dates if } d+6 \leq \text{len(Dates)} : \sum_{k=0}^{6} \sum_{s \in \text{Shifts}} F(i,d+k,s) \leq 7$$

这确保任何全职员工在连续 7 天内的工作天数不超过 7 天。

4.2 建模求解

针对这个问题，我们选择开源的 Python/PuLP 库求解。PuLP 作为一个 Python 库，可以定义问题、变量、目标函数和约束，然后使用 CBC 或其他求解器求解这些问题。这里我们选择在 PuLP 库中调用 CBC（Coin-or branch and cut）解决这个问题。

由于数量较大，我们设置了一个容忍度 0.001%，这是我们在这个容忍度下得到的结果。

```
Result - Optimal solution found (within gap tolerance)

Objective value:                51921.00000000
Lower bound:                    51872.078
Gap:                            0.00
Enumerated nodes:               0
Total iterations:               7555
Time (CPU seconds):             9.44
Time (Wallclock seconds):       9.90

Option for printingOptions changed from normal to all
Total time (CPU seconds):       9.53   (Wallclock seconds):       9.99
```

附录

A 问题一代码

随机森林调参代码

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler

# 假设存在的 SCid 列表
data_for_sc = pd.read_csv("../../附件/附件1.csv", encoding="GB2312")
ALL_SC = list(set(data_for_sc["分拣中心"]))
existing_scs = list(map(lambda SC_: int(SC_[2:]), ALL_SC))
existing_scs.sort()


# 加载数据
def load_data(existing_scs):
    all_data = []
    for sc_id in existing_scs:
        try:
            data = pd.read_csv(f"SC{sc_id}.csv")
            data["center_id"] = sc_id
            all_data.append(data)
        except FileNotFoundError:
            print(f"File for center {sc_id} not found.")
    return pd.concat(all_data, ignore_index=True) if all_data else pd.DataFrame()


# 数据清洗和预处理
def preprocess_data(data):
    data["date"] = pd.to_datetime(data["date"], errors="coerce")
    data.dropna(subset=["date", "value"], inplace=True)

    data["year"] = data["date"].dt.year
    data["month"] = data["date"].dt.month
    data["day"] = data["date"].dt.day
    data["weekday"] = data["date"].dt.weekday

    scaler = StandardScaler()
    data[["year", "month", "day", "weekday"]] = scaler.fit_transform(
        data[["year", "month", "day", "weekday"]]
    )

```

```
    return data


# 模型训练和参数调整
def train_and_optimize_model(X_train, y_train):
    param_grid = {
        "n_estimators": [100 * i for i in range(1, 10)],
        "max_depth": [10 * i for i in range(1, 10)],
        "min_samples_split": [10 * i for i in range(1, 10)],
    }
    model = RandomForestRegressor(random_state=42)
    grid_search = GridSearchCV(
        model, param_grid, cv=5, scoring="neg_mean_squared_error", verbose=2
    )
    grid_search.fit(X_train, y_train)

    print("Best parameters:", grid_search.best_params_)
    return grid_search.best_estimator_


if __name__ == "__main__":
    data = load_data(existing_scs)
    if not data.empty:
        data = preprocess_data(data)
        features = data[["center_id", "year", "month", "day", "weekday"]]
        target = data["value"]

        X_train, X_test, y_train, y_test = train_test_split(
            features, target, test_size=0.2, random_state=42
        )

        best_model = train_and_optimize_model(X_train, y_train)

        y_pred = best_model.predict(X_test)
        mse = mean_squared_error(y_test, y_pred)
        print(f"Test MSE: {mse}")
    else:
        print("No data loaded, please check the data files.")
```

随机森林训练代码

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# 假设数据已经按照分拣中心编号和日期排列好
# 每个文件名格式为'SCi.csv', 其中 i 是分拣中心编号
```

```python
# 假设存在的 SCid 列表
data_for_sc = pd.read_csv("../../附件/附件1.csv", encoding="GB2312")
ALL_SC = list(set(data_for_sc["分拣中心"]))
existing_scs = list(map(lambda SC_: int(SC_[2:]), ALL_SC))
existing_scs.sort()

# 加载数据
def load_data(existing_scs):
    all_data = []
    for i in existing_scs:
        data = pd.read_csv(f"SC{i}.csv")
        data["SC_id"] = i
        all_data.append(data)
    return pd.concat(all_data, ignore_index=True)


# 数据预处理
def preprocess(data):
    # 假设数据包含日期和货量两列，日期列名为'date'，货量列名为'value'
    data["date"] = pd.to_datetime(data["date"])
    data["year"] = (pd.to_datetime(data["date"])).dt.year
    data["month"] = (pd.to_datetime(data["date"])).dt.month
    data["day"] = (pd.to_datetime(data["date"])).dt.day
    data["weekday"] = (pd.to_datetime(data["date"])).dt.weekday
    return data


# 训练模型
def train_model(data):
    features = data[["SC_id", "year", "month", "day", "weekday"]]
    target = data["value"]
    X_train, X_test, y_train, y_test = train_test_split(
        features,
        target,
        test_size=0.2,
        random_state=50,

    )

    model = RandomForestRegressor(n_estimators=300, random_state=42, min_samples_split=20)
    model.fit(X_train, y_train)

    # 预测和评估
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    print(f"Mean Squared Error: {mse}")
    return model

```

```python

# 预测未来的货量
def predict_future(model, start_date, num_days, existing_scs):
    future_dates = pd.date_range(start_date, periods=num_days)
    future_data = pd.DataFrame(
        {
            "date": np.repeat(future_dates, len(existing_scs)),
            "SC_id": np.tile(existing_scs, num_days),
        }
    )
    future_data = preprocess(future_data)
    features = future_data[["SC_id", "year", "month", "day", "weekday"]]
    predictions = model.predict(features)
    future_data["predicted_volume"] = predictions
    return future_data


# 保存结果到 CSV
def save_predictions_to_csv(predictions, file_name):
    predictions["date"] = predictions["date"].dt.strftime("%Y/%m/%d")
    predictions.to_csv(file_name, index=False)
    print(f"Saved predictions to {file_name}")


# 主函数
def main():
    data = load_data(existing_scs)
    data = preprocess(data)
    model = train_model(data)
    future_predictions = predict_future(model, "2023-08-01", 153, existing_scs)
    save_predictions_to_csv(future_predictions, "predicted_volumes.csv")


if __name__ == "__main__":
    main()
```

ARIMA 和 LSTM 的代码未被采用，因此仅放在支撑材料中。

B 问题二代码

按小时预测

```python
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error


```

```python
def load_data():
    data_sc = pd.read_csv("附件2.csv", encoding="gb2312")  # Updated file name
    data_routes = pd.read_csv("附件3.csv", encoding="gb2312")
    data_changes = pd.read_csv("附件4.csv", encoding="gb2312")
    return data_sc, data_routes, data_changes


def preprocess_data(data_sc, data_routes, data_changes):
    data_sc["日期"] = pd.to_datetime(data_sc["日期"], format="%Y/%m/%d")
    le = LabelEncoder()
    data_sc["分拣中心"] = le.fit_transform(data_sc["分拣中心"])

    # New: Extract hour from the '小时' column
    data_sc["小时"] = data_sc["小时"].astype(int)

    if "小时" in data_sc.columns:
        # Calculate the average departure and arrival for each hour
        avg_departure_hourly = (
            data_sc.groupby(["分拣中心", "小时"])["货量"]
            .mean()
            .rename("平均发出货量_hourly")
            .reset_index()
        )
        avg_arrival_hourly = (
            data_sc.groupby(["分拣中心", "小时"])["货量"]
            .mean()
            .rename("平均到达货量_hourly")
            .reset_index()
        )

        # Merge the adjusted volume information
        data_sc = pd.merge(
            data_sc, avg_departure_hourly, on=["分拣中心", "小时"], how="left"
        )
        data_sc = pd.merge(
            data_sc, avg_arrival_hourly, on=["分拣中心", "小时"], how="left"
        )
        data_sc.fillna(
            {"平均发出货量_hourly": 0, "平均到达货量_hourly": 0}, inplace=True
        )

        return data_sc, le
    else:
        print("'小时' 列不存在于数据框 'data_sc' 中! ")
        return None, None


def create_features_labels(data):
```

```
    data["year"] = data["日期"].dt.year
    data["month"] = data["日期"].dt.month
    data["day"] = data["日期"].dt.day
    data["weekday"] = data["日期"].dt.weekday
    features = data[
        [
            "分拣中心",
            "year",
            "month",
            "day",
            "weekday",
            "小时",
            "平均发出货量_hourly",
            "平均到达货量_hourly",
        ]
    ]
    labels = data["货量"]
    return features, labels


def train_and_predict(features, labels):
    X_train, X_test, y_train, y_test = train_test_split(
        features, labels, test_size=0.2, random_state=42
    )
    model = RandomForestRegressor(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    print(f"Mean Squared Error: {mse}")
    return model


def predict_future(model, last_date, le, data_sc, existing_csc):
    # Generate future dates and hours for existing sorting centers
    future_dates = pd.date_range(start=last_date + pd.Timedelta(days=1), periods=30)
    hours = range(24)

    future_df = pd.DataFrame(
        {
            "日期": [
                date for date in future_dates for _ in hours for _ in existing_csc
            ],
            "小时": [
                hour for _ in future_dates for hour in hours for _ in existing_csc
            ],
            "分拣中心": existing_csc * len(future_dates) * len(hours),
        }
    )
```

```
104
105     future_df["year"] = future_df["日期"].dt.year
106     future_df["month"] = future_df["日期"].dt.month
107     future_df["day"] = future_df["日期"].dt.day
108     future_df["weekday"] = future_df["日期"].dt.weekday
109
110     # Estimate average departure and arrival volumes
111     avg_departure = data_sc["平均发出货量_hourly"].mean()
112     avg_arrival = data_sc["平均到达货量_hourly"].mean()
113
114     future_df["平均发出货量_hourly"] = avg_departure
115     future_df["平均到达货量_hourly"] = avg_arrival
116
117     # Extract features
118     features = future_df[
119         [
120             "分拣中心",
121             "year",
122             "month",
123             "day",
124             "weekday",
125             "小时",
126             "平均发出货量_hourly",
127             "平均到达货量_hourly",
128         ]
129     ]
130
131     # Use the model to make predictions
132     predicted_values = model.predict(features)
133     future_df["value"] = predicted_values
134     future_df["日期"] = future_df["日期"].dt.strftime("%Y/%m/%d")  # Format the date
135
136     results_df = future_df[["分拣中心", "日期", "小时", "value"]]
137     results_df.rename(columns={"分拣中心": "SC_ID", "小时": "Hour"}, inplace=True)
138
139     # Save to CSV
140     results_df.to_csv("future_predictions_hourly.csv", index=False)
141     print("未来预测结果已保存到 'future_predictions_hourly.csv'")
142
143
144 def main():
145     data_sc, data_routes, data_changes = load_data()
146     data_sc, le = preprocess_data(data_sc, data_routes, data_changes)
147
148     if data_sc is not None:
149         data_for_sc = pd.read_csv("../../附件/附件1.csv", encoding="GB2312")
150         ALL_SC = list(set(data_for_sc["分拣中心"]))
151         existing_csc = list(map(lambda SC_: int(SC_[2:]), ALL_SC))
```

```
        existing_csc.sort()
        features, labels = create_features_labels(data_sc)
        model = train_and_predict(features, labels)
        last_date = data_sc["日期"].max()  # Get the last known date
        predict_future(model, last_date, le, data_sc, existing_csc)


if __name__ == "__main__":
    main()
```

按天预测

```
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error

def load_data():
    data_sc = pd.read_csv('附件1.csv', encoding='gb2312')
    data_routes = pd.read_csv('附件3.csv', encoding='gb2312')
    data_changes = pd.read_csv('附件4.csv', encoding='gb2312')
    return data_sc, data_routes, data_changes

def preprocess_data(data_sc, data_routes, data_changes, existing_csc):
    data_sc['日期'] = pd.to_datetime(data_sc['日期'], format='%Y/%m/%d')
    le = LabelEncoder()
    data_sc['分拣中心'] = le.fit_transform(data_sc['分拣中心'])

    # Calculate the average departure and arrival for each sorting center
    avg_departure = data_routes.groupby('始发分拣中心')['货量'].mean().rename('平均发出货量').
  reset_index()
    avg_arrival = data_routes.groupby('到达分拣中心')['货量'].mean().rename('平均到达货量').
  reset_index()

    # Convert sorting center codes
    avg_departure['始发分拣中心'] = le.transform(avg_departure['始发分拣中心'])
    avg_arrival['到达分拣中心'] = le.transform(avg_arrival['到达分拣中心'])

    # Merge the adjusted volume information
    data_sc = pd.merge(data_sc, avg_departure, left_on='分拣中心', right_on='始发分拣中心', how='
  left')
    data_sc = pd.merge(data_sc, avg_arrival, left_on='分拣中心', right_on='到达分拣中心', how='left'
  )
    data_sc.fillna({'平均发出货量': 0, '平均到达货量': 0}, inplace=True)

    return data_sc, le

def create_features_labels(data):
```

```
    data['year'] = data['日期'].dt.year
    data['month'] = data['日期'].dt.month
    data['day'] = data['日期'].dt.day
    data['weekday'] = data['日期'].dt.weekday
    features = data[['分拣中心', 'year', 'month', 'day', 'weekday', '平均发出货量', '平均到达货量']]
    labels = data['货量']
    return features, labels

def train_and_predict(features, labels):
    X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2,
  random_state=42)
    model = RandomForestRegressor(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    print(f"Mean Squared Error: {mse}")
    return model

def predict_future(model, last_date, le, data_sc, existing_csc):
    # Generate future dates for the next 30 days
    future_dates = pd.date_range(start=last_date + pd.Timedelta(days=1), periods=30)
    future_df = pd.DataFrame({
        'date': pd.to_datetime(list(future_dates) * len(existing_csc)),
        '分拣中心': existing_csc * len(future_dates)
    })

    future_df['year'] = future_df['date'].dt.year
    future_df['month'] = future_df['date'].dt.month
    future_df['day'] = future_df['date'].dt.day
    future_df['weekday'] = future_df['date'].dt.weekday

    # Estimate average departure and arrival volumes
    avg_departure = data_sc['平均发出货量'].mean()
    avg_arrival = data_sc['平均到达货量'].mean()

    future_df['平均发出货量'] = avg_departure
    future_df['平均到达货量'] = avg_arrival

    future_df = future_df[future_df['分拣中心'].isin(existing_csc)]

    # Extract features
    features = future_df[['分拣中心', 'year', 'month', 'day', 'weekday', '平均发出货量', '平均到达货
  量']]

    # Use the model to make predictions
    predicted_values = model.predict(features)
    future_df['value'] = predicted_values
    future_df['date'] = future_df['date'].dt.strftime('%Y/%m/%d')  # Format the date
```

```
    results_df = future_df[['分拣中心', 'date', 'value']]
    results_df.rename(columns={'分拣中心': 'SC_ID'}, inplace=True)

    # Save to CSV
    results_df.to_csv('future_predictions.csv', index=False)
    print("未来预测结果已保存到 'future_predictions.csv'")

def main():
    data_sc, data_routes, data_changes = load_data()
    existing_csc = [
        1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 15, 16, 17, 18, 19, 20,
        21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 34, 35, 36, 37,
        38, 39, 40, 41, 43, 44, 46, 47, 48, 49, 51, 52, 53, 54, 55, 56,
        57, 58, 60, 61, 63, 66, 68
    ]
    data_sc, le = preprocess_data(data_sc, data_routes, data_changes, existing_csc)
    features, labels = create_features_labels(data_sc)
    model = train_and_predict(features, labels)
    last_date = data_sc['日期'].max()  # Get the last known date
    predict_future(model, last_date, le, data_sc, existing_csc)

if __name__ == "__main__":
    main()
```

C 问题四代码

```
import pandas as pd
from pulp import LpProblem, LpMinimize, LpVariable, lpSum, LpBinary, PULP_CBC_CMD

# 读取数据
df = pd.read_csv("read.csv")
df["date"] = pd.to_datetime(df["date"]).dt.date

# 班次及时间段
shifts = [(0, 8), (5, 13), (8, 16), (12, 20), (14, 22), (16, 24)]
shift_labels = ["Shift1", "Shift2", "Shift3", "Shift4", "Shift5", "Shift6"]
dates = sorted(df["date"].unique())

# 计算每个班次的需求
demand_per_shift = {date: {} for date in dates}
for date in dates:
    daily_data = df[df["date"] == date]
    for label, (start, end) in zip(shift_labels, shifts):
        demand_per_shift[date][label] = daily_data[
            (daily_data["hour"] >= start) & (daily_data["hour"] < end)
        ]["value"].sum()
```

```python

# 建立优化模型
model = LpProblem("Personnel_Scheduling", LpMinimize)

# 定义变量
full_time = LpVariable.dicts(
    "FullTime", (dates, shift_labels, range(200)), 0, 1, LpBinary
)
temp_workers = LpVariable.dicts(
    "TempWorkers", (dates, shift_labels), 0, None, cat="Integer"
)

# 目标函数：最小化总人天数
model += lpSum(
    full_time[date][shift][i]
    for date in dates
    for shift in shift_labels
    for i in range(200)
) + lpSum(temp_workers[date][shift] for date in dates for shift in shift_labels)

# 每个班次的需求必须被满足的约束
for date in dates:
    for shift in shift_labels:
        model += (
            25 * lpSum(full_time[date][shift][i] for i in range(200))
            + 20 * temp_workers[date][shift]
            >= demand_per_shift[date][shift]
        )

# 正式工的出勤率不超过 85for i in range(200):
    model += (
        lpSum(full_time[date][shift][i] for date in dates for shift in shift_labels)
        <= 30 * 0.85
    )

# 正式工连续出勤天数不超过 7 天
for i in range(200):
    for d in range(len(dates) - 6):
        model += (
            lpSum(
                full_time[dates[d + k]][shift][i]
                for k in range(7)
                for shift in shift_labels
            )
            <= 7
        )
# 求解问题
model.solve(PULP_CBC_CMD(msg=1, threads=8,gapRel=0.001))
```

```python
print("soolved")
# 收集结果并输出为 CSV
results = []
for date in dates:
    for shift in shift_labels:
        for i in range(200):
            if full_time[date][shift][i].varValue > 0:
                results.append(
                    {
                        "Sorting_Center": "SC60",
                        "Date": date,
                        "Shift": shift,
                        "Employee": f"FullTime({i})",
                    }
                )
        temp_workers_count = int(temp_workers[date][shift].varValue)
        for j in range(temp_workers_count):
            results.append(
                {
                    "Sorting_Center": "SC60",
                    "Date": date,
                    "Shift": shift,
                    "Employee": f"Temp({j})",
                }
            )

# 创建 DataFrame 并保存到 CSV
results_df = pd.DataFrame(results)
results_df.to_csv("scheduling_results.csv", index=False)
print("CSV file has been saved with the scheduling results.")
```