四、作业四（共计25分）

得分

得分

**题目：**请以中国体彩网（https://www.zhcw.com）为数据源，完成以下任务：

1. 爬取截至2025年7月1日之前100期的大乐透开奖数据和中奖情况（链接：https://www.zhcw.com/kjxx/dlt/），分析大乐透总销售额随开奖日期的变化趋势并预测2025年7月1日之后最近一期的销售额。（2）根据爬取的大乐透数据对前区号码与后区号码进行频率统计与可视化，分析其历史分布规律。根据你发现的规律或者采用其他预测方法推荐一组大乐透号码，作为2025年7月1日之后最近一期的投注号码。（3）根据爬取的大乐透数据，分别统计周一、周三、周六的大乐透开奖号码和总销售额。对比不同开奖日之间的号码分布与销售额特征，分析是否存在显著差异或相似性。（4）爬取任意一个彩种中20位以上专家的公开数据，对专家的基本属性（彩龄、发文量等）和表现（中奖情况）进行统计分析，并通过可视化展示其分布规律、相互关系或对中奖率的影响。

**数据源：**中国体彩网（https://www.zhcw.com）

**4.1爬取2024年胡润百富榜数据、趋势分析与销售额预测**

**4.1.1实现步骤**

**数据爬取部分（T4\_1\_1.py）：**

1. 数据源分析：分析中国体彩网大乐透数据接口，确定请求参数和返回数据结构。
2. 设置请求头：模拟浏览器访问，设置User-Agent、Referer等头信息。
3. 分页爬取：循环处理4页（每页30条，共100期）数据。
4. 反爬处理：使用随机等待时间（1.0-2.5秒）避免被封IP。
5. 数据处理：解析JSON数据，提取开奖日期、期号、销售额、奖池、中奖号码、各奖项详情等字段。
6. 特殊字段处理：处理嵌套的中奖详情结构，格式化金额字段。
7. 数据存储：将数据保存为CSV文件。

**趋势分析和销售额预测部分（T4\_1\_2.py）：**

1. 数据预处理：读取数据，转换日期格式，按星期分类（周一、周三、周六），计算移动平均线。
2. 趋势分析：绘制销售额趋势图，包括原始数据、移动平均线（3期、7期、30期）和线性趋势线，并标注极值点。
3. 统计分析：计算并展示最高、最低、平均销售额和标准差等统计指标。
4. 销售额预测：使用三种方法预测下一期（周三）的销售额：

a. 按星期分组预测：基于历史周三的平均销售额，结合近期趋势调整。

b. SARIMA时间序列预测：利用季节性自回归移动平均模型进行预测。

c. 移动平均预测：结合整体3期移动平均和周三3期移动平均进行预测。

1. 可视化预测结果：将三种预测方法的结果在趋势图上标注展示。

**4.1.2重要库函数**

* requests :请求获取数据
* pandas:数据处理和分析
* matplotlib:数据可视化
* scipy.stats:线性回归分析
* statsmodels: SARIMA时间序列预测
* numpy:数值计算
* random 和 time: 实现随机等待避免反爬
* json：解析JSON数据，提取关键字段

**4.1.3重要代码：数据爬取、趋势分析与销售额预测**

def fetch\_dlt\_data():

    base\_url = "https://jc.zhcw.com/port/client\_json.php"

    headers = {

        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36',

        'Referer': 'https://www.zhcw.com/kjxx/dlt/',

        'Accept': 'application/json, text/javascript, \*/\*; q=0.01',

        'Accept-Language': 'zh-CN,zh;q=0.9,en;q=0.8',

        'Connection': 'keep-alive'

    }

    all\_data = []

    total\_pages = 4

    # 分页获取数据

    for page in range(1, total\_pages + 1):

        # 生成时间戳参数

        timestamp = int(time.time() \* 1000)

        params = {

            'transactionType': '10001001',

            'lotteryId': '281',

            'issueCount': '100',

            'pageNum': str(page),

            'pageSize': '30',

            'type': '0',

            'tt': f'0.{timestamp}',

            '\_': str(timestamp)

        }

        try:

            print(f"正在获取第 {page}/{total\_pages} 页数据...")

            response = requests.get(

                url=base\_url,

                params=params,

                headers=headers,

                timeout=20

            )

            if response.status\_code == 200:

                # 处理JSONP响应

                json\_str = response.text

                if json\_str.startswith('jQuery') and '(' in json\_str:

                    json\_str = json\_str.split('(', 1)[1].rsplit(')', 1)[0]

                try:

                    data = json.loads(json\_str)

                except json.JSONDecodeError:

                    print("  响应不是有效JSON，尝试直接解析...")

                    json\_str = response.text.strip().strip(';').strip()

                    if json\_str.startswith('jQuery') and '(' in json\_str:

                        json\_str = json\_str.split('(', 1)[1].rsplit(')', 1)[0]

                    data = json.loads(json\_str)

                # 确保data字段存在

                if 'data' in data and isinstance(data['data'], list):

                    page\_data\_count = 0

                    for item in data['data']:

                        # 解析中奖详情

                        prize\_details = parse\_winner\_details(item.get('winnerDetails', []))

                        # 创建详情页URL

                        issue\_number = item.get('issue', '')

                        detail\_url = f"https://www.zhcw.com/kjxx/dlt/kjxq/?kjData={issue\_number}" if issue\_number else ""

                        # 创建数据记录

                        record = {

                            '期号': issue\_number,

                            '详情页链接': detail\_url,

                            '开奖日期': item.get('openTime'),

                            '星期': item.get('week'),

                            '中奖号码\_前区': item.get('frontWinningNum', ''),

                            '中奖号码\_后区': item.get('backWinningNum', ''),

                            '出球顺序\_前区': item.get('seqFrontWinningNum', ''),

                            '出球顺序\_后区': item.get('seqBackWinningNum', ''),

                            '总销售额(元)': format\_amount(item.get('saleMoney', '0')),

                            '奖池奖金(元)': format\_amount(item.get('prizePoolMoney', '0')),

                        }

                        # 添加中奖详情

                        record.update(prize\_details)

                        all\_data.append(record)

                        page\_data\_count += 1

                    print(f"  成功解析 {page\_data\_count} 期数据")

                else:

                    print(f"  第{page}页返回数据格式异常")

                    print(f"  响应内容: {response.text[:500]}")

            else:

                print(f"  请求失败，状态码：{response.status\_code}")

                print(f"  响应内容: {response.text[:200]}")

        except Exception as e:

            print(f"  请求异常：{str(e)}")

        # 添加随机请求间隔，避免被封IP

        time.sleep(random.uniform(1.0, 2.5))

return all\_data

def parse\_winner\_details(winner\_details):

    """解析中奖详情数据结构，只为1-2等奖添加派奖和追加派奖字段"""

    prize\_data = {}

    # 初始化所有奖项的基本和追加投注字段

    for level in range(1, 10):

        # 基本投注

        prize\_data[f'{level}等奖\_基本注数'] = 0

        prize\_data[f'{level}等奖\_基本单注奖金(元)'] = 0

        prize\_data[f'{level}等奖\_基本总奖金(元)'] = 0

        # 追加投注

        prize\_data[f'{level}等奖\_追加注数'] = 0

        prize\_data[f'{level}等奖\_追加单注奖金(元)'] = 0

        prize\_data[f'{level}等奖\_追加总奖金(元)'] = 0

    # 只为1-2等奖添加派奖和追加派奖字段

    for level in range(1, 3):

        prize\_data[f'{level}等奖\_派奖注数'] = 0

        prize\_data[f'{level}等奖\_派奖单注金额(元)'] = 0

        prize\_data[f'{level}等奖\_派奖总金额(元)'] = 0

        prize\_data[f'{level}等奖\_追加派奖注数'] = 0

        prize\_data[f'{level}等奖\_追加派奖单注金额(元)'] = 0

        prize\_data[f'{level}等奖\_追加派奖总金额(元)'] = 0

    # 解析每个奖项

    for detail in winner\_details:

        try:

            level = int(detail.get('awardEtc', 0))

            if not (1 <= level <= 9):

                continue

            # 处理基本投注

            base = detail.get('baseBetWinner', {})

            if base and isinstance(base, dict):

                prize\_data[f'{level}等奖\_基本注数'] = base.get('awardNum', 0)

                prize\_data[f'{level}等奖\_基本单注奖金(元)'] = format\_amount(base.get('awardMoney', 0))

                prize\_data[f'{level}等奖\_基本总奖金(元)'] = format\_amount(base.get('totalMoney', 0))

            # 处理追加投注

            add = detail.get('addToBetWinner', {})

            if add and isinstance(add, dict):

                prize\_data[f'{level}等奖\_追加注数'] = add.get('awardNum', 0)

                prize\_data[f'{level}等奖\_追加单注奖金(元)'] = format\_amount(add.get('awardMoney', 0))

                prize\_data[f'{level}等奖\_追加总奖金(元)'] = format\_amount(add.get('totalMoney', 0))

            # 只处理1-2等奖的派奖信息

            if level in (1, 2):

                # 处理派奖

                add2 = detail.get('addToBetWinner2', {})

                if add2 and isinstance(add2, dict):

                    prize\_data[f'{level}等奖\_派奖注数'] = add2.get('awardNum', 0)

                    prize\_data[f'{level}等奖\_派奖单注金额(元)'] = format\_amount(add2.get('awardMoney', 0))

                    prize\_data[f'{level}等奖\_派奖总金额(元)'] = format\_amount(add2.get('totalMoney', 0))

                # 处理追加派奖

                add3 = detail.get('addToBetWinner3', {})

                if add3 and isinstance(add3, dict):

                    prize\_data[f'{level}等奖\_追加派奖注数'] = add3.get('awardNum', 0)

                    prize\_data[f'{level}等奖\_追加派奖单注金额(元)'] = format\_amount(add3.get('awardMoney', 0))

                    prize\_data[f'{level}等奖\_追加派奖总金额(元)'] = format\_amount(add3.get('totalMoney', 0))

        except Exception as e:

            print(f"解析中奖详情出错：{str(e)}")

            print(f"问题数据: {detail}")

    return prize\_data

# 读取数据并处理

df = pd.read\_csv('近100期大乐透开奖数据和中奖情况.csv', usecols=[2, 8])  # 第三列索引2，第九列索引8

df.columns = ['开奖日期', '总销售额']  # 重命名列

# 转换日期格式并排序

df['开奖日期'] = pd.to\_datetime(df['开奖日期'])

df.sort\_values('开奖日期', inplace=True)  # 按日期升序排列

# 添加星期几信息（0=周一，1=周三，2=周六）

df['星期'] = df['开奖日期'].dt.weekday.map({0: '周一', 2: '周三', 5: '周六'})

# 计算移动平均线

df['3期移动平均'] = df['总销售额'].rolling(window=3).mean()

df['7期移动平均'] = df['总销售额'].rolling(window=7).mean()

df['30期移动平均'] = df['总销售额'].rolling(window=30).mean()

# 创建画布

plt.figure(figsize=(14, 8), facecolor='#f8f9fa')

ax = plt.gca()

ax.set\_facecolor('#f8f9fa')

# 绘制销售额趋势

plt.plot(df['开奖日期'], df['总销售额'], label='每日销售额', color='#1f77b4',

         alpha=0.7, marker='o', markersize=4, linewidth=1.5)

# 绘制移动平均线

plt.plot(df['开奖日期'], df['3期移动平均'], label='3期移动平均', color='#9467bd', linewidth=2.5, linestyle='-.')

plt.plot(df['开奖日期'], df['7期移动平均'], label='7期移动平均', color='#ff7f0e', linewidth=2.5, linestyle='--')

plt.plot(df['开奖日期'], df['30期移动平均'], label='30期移动平均', color='#2ca02c', linewidth=2.5)

# 添加趋势线（线性回归）

x\_num = mdates.date2num(df['开奖日期'])

slope, intercept, r\_value, p\_value, std\_err = linregress(x\_num, df['总销售额'])

trendline = slope \* x\_num + intercept

plt.plot(df['开奖日期'], trendline, color='#d62728', linewidth=2.5, linestyle='-.',

         label=f'趋势线 (R²={r\_value\*\*2: .3f})'.replace('²', '^2'))

# 标记极值点

max\_idx = df['总销售额'].idxmax()

min\_idx = df['总销售额'].idxmin()

plt.scatter(df.loc[max\_idx, '开奖日期'], df.loc[max\_idx, '总销售额'], color='#d62728', s=100, zorder=5)

plt.scatter(df.loc[min\_idx, '开奖日期'], df.loc[min\_idx, '总销售额'], color='#2ca02c', s=100, zorder=5)

# 添加文本标注

plt.annotate(f'最高: {df.loc[max\_idx, "总销售额"]/1e6: .2f}百万', xy=(df.loc[max\_idx, '开奖日期'], df.loc[max\_idx, '总销售额']),

             xytext=(10, 20), textcoords='offset points', arrowprops=dict(arrowstyle='->', color='#d62728'))

plt.annotate(f'最低: {df.loc[min\_idx, "总销售额"]/1e6: .2f}百万', xy=(df.loc[min\_idx, '开奖日期'], df.loc[min\_idx, '总销售额']),

             xytext=(10, -30), textcoords='offset points', arrowprops=dict(arrowstyle='->', color='#2ca02c'))

# 设置标题和标签

plt.title('大乐透总销售额趋势分析 (近100期)', fontsize=16, pad=20)

plt.xlabel('开奖日期', fontsize=12)

plt.ylabel('总销售额 (元)', fontsize=12)

plt.grid(True, linestyle='--', alpha=0.7)

# 格式化坐标轴

ax.xaxis.set\_major\_formatter(mdates.DateFormatter('%Y-%m'))

ax.xaxis.set\_major\_locator(mdates.MonthLocator(interval=1))

plt.xticks(rotation=45)

plt.gcf().autofmt\_xdate()

# 添加统计信息框

stats\_text = f"""数据统计:

最高销售额: {df['总销售额'].max()/1e6: .2f} 百万

最低销售额: {df['总销售额'].min()/1e6: .2f} 百万

平均销售额: {df['总销售额'].mean()/1e6: .2f} 百万

标准差: {df['总销售额'].std()/1e6: .2f} 百万

销售额变化率: {(df['总销售额'].iloc[-1]/df['总销售额'].iloc[0]-1)\*100: .1f}%"""

plt.text(0.02, 0.98, stats\_text, transform=ax.transAxes, verticalalignment='top',

         bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))

# 添加图例

plt.legend(loc='upper right', fontsize=10)

plt.tight\_layout()

plt.subplots\_adjust(bottom=0.15)

plt.savefig('大乐透总销售额随开奖日期的变化趋势.png', dpi=300)

plt.show()

# 预测

# 方法1：按星期几分组预测

def method\_weekday\_avg(df):

    # 计算各星期平均销售额

    weekday\_avg = df.groupby('星期')['总销售额'].mean()

    # 计算最近3期周三销售额的加权平均（最近一期权重最高）

    wed\_data = df[df['星期'] == '周三'].tail(3)['总销售额']

    weights = np.array([0.2, 0.3, 0.5])  # 权重分配（由远到近）

    weighted\_avg = np.sum(wed\_data.values \* weights)

    # 综合预测值（基础平均值占40%，近期趋势占60%）

    wed\_pred = 0.4 \* weekday\_avg['周三'] + 0.6 \* weighted\_avg

    return wed\_pred, weekday\_avg

# 方法2：SARIMA时间序列预测（修复索引问题）

def method\_sarima(df):

    # 创建整数索引的时间序列

    ts = df['总销售额'].reset\_index(drop=True)

    # SARIMA参数设置 (p,d,q)(P,D,Q,s)

    # 基于数据特点：小数据量+明显周周期(s=3)

    order = (1, 0, 1)  # 非季节性部分

    seasonal\_order = (1, 0, 1, 3)  # 季节性部分（周期为3）

    # 训练SARIMA模型

    model = SARIMAX(ts, order=order, seasonal\_order=seasonal\_order,

                    enforce\_stationarity=False, enforce\_invertibility=False)

    results = model.fit(disp=False, maxiter=100)

    # 预测下一期

    forecast = results.get\_forecast(steps=1)

    pred\_value = forecast.predicted\_mean.iloc[0]

    conf\_int = forecast.conf\_int().iloc[0].tolist()

    # 获取拟合值（处理可能的缺失值）

    fitted\_values = results.fittedvalues

    if len(fitted\_values) < len(ts):

        # 补齐缺失的拟合值

        n\_missing = len(ts) - len(fitted\_values)

        fitted\_values = pd.concat([pd.Series([np.nan] \* n\_missing), fitted\_values])

    return pred\_value, conf\_int, fitted\_values

# 方法3：移动平均预测（优化）

def method\_moving\_average(df):

    # 计算3期移动平均

    df['3期移动平均'] = df['总销售额'].rolling(window=3, min\_periods=1).mean()

    # 计算周三的移动平均

    wed\_mask = df['星期'] == '周三'

    wed\_ma = df.loc[wed\_mask, '总销售额'].rolling(window=3, min\_periods=1).mean()

    # 获取最近值

    last\_three = df['总销售额'].tail(3).values

    last\_wed\_ma = wed\_ma.iloc[-1] if not wed\_ma.empty else np.mean(last\_three)

    # 综合预测值（整体移动平均占50%，周三移动平均占50%）

    final\_pred = 0.5 \* np.mean(last\_three) + 0.5 \* last\_wed\_ma

    return final\_pred, df

# 执行三种预测方法

pred\_weekday, weekday\_avg = method\_weekday\_avg(df)

pred\_sarima, conf\_int, fitted\_values = method\_sarima(df)

pred\_ma, df = method\_moving\_average(df)

**4.1.4爬取、分析、预测结果**

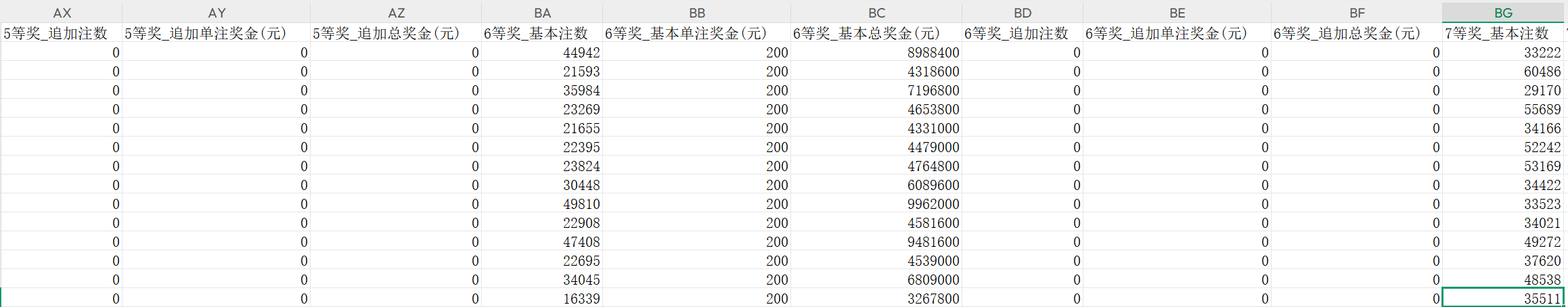






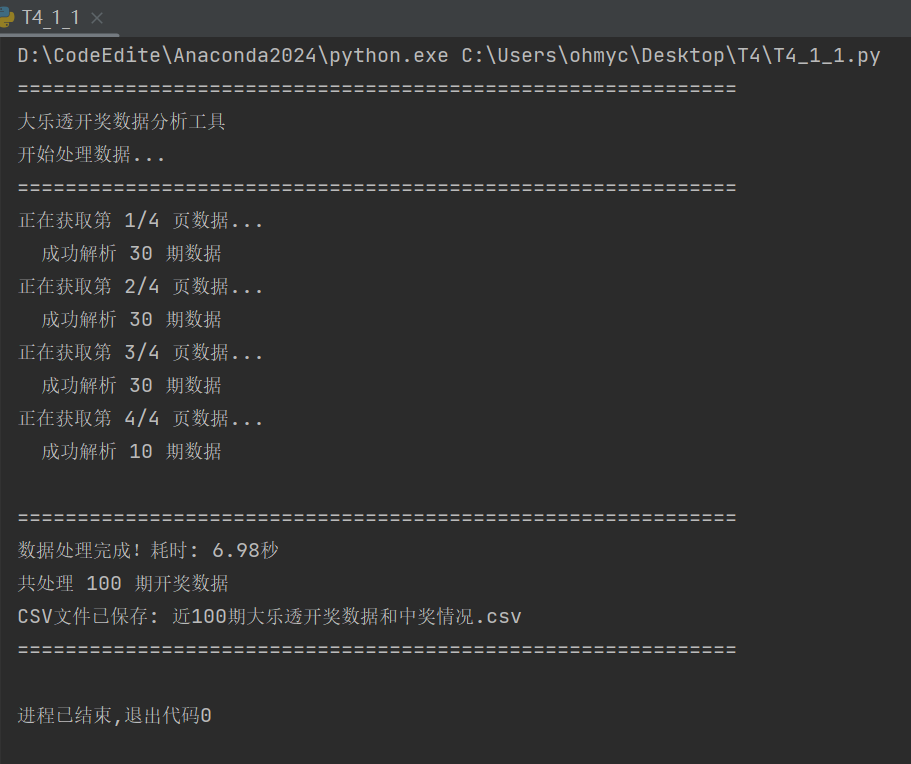


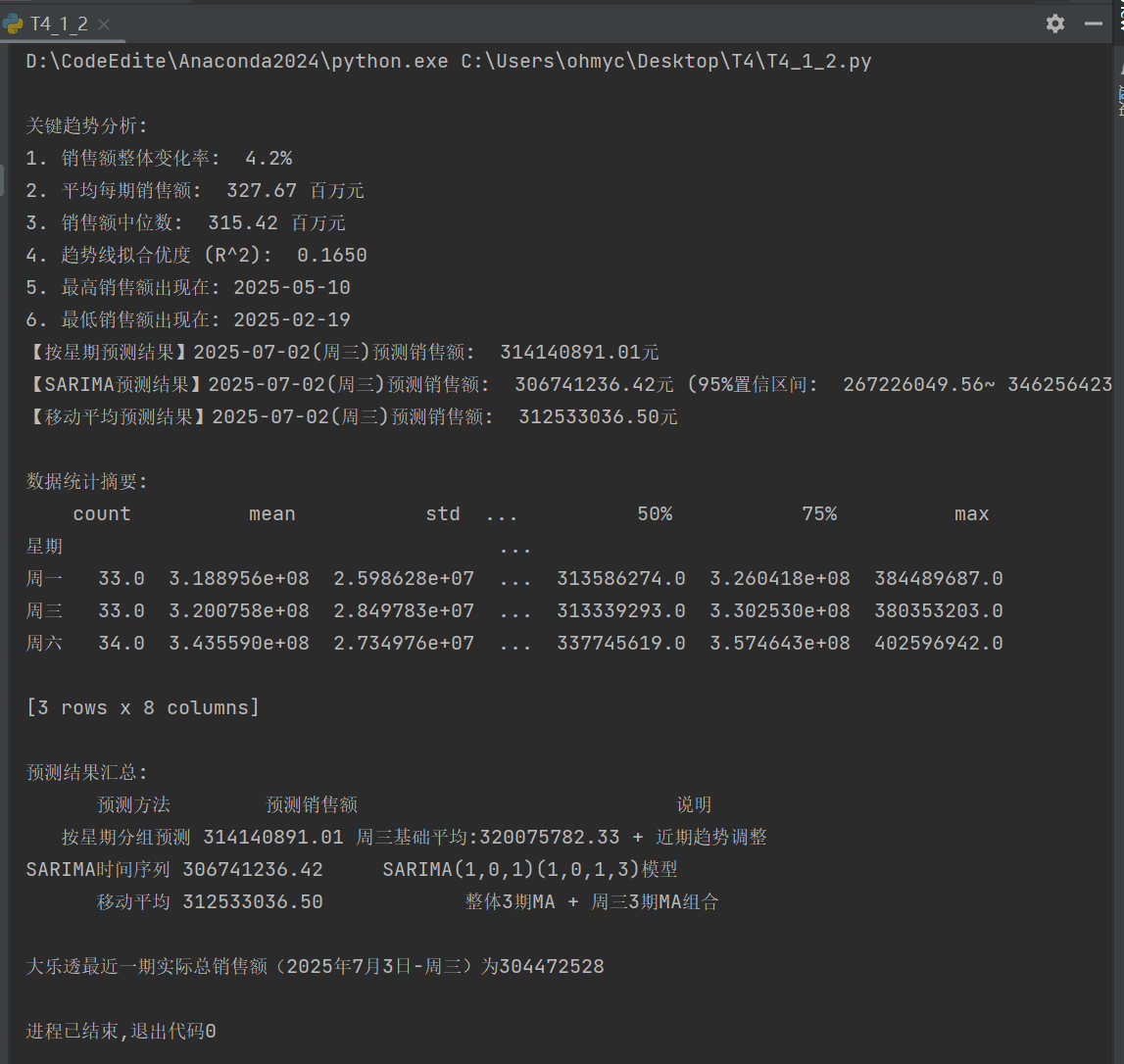


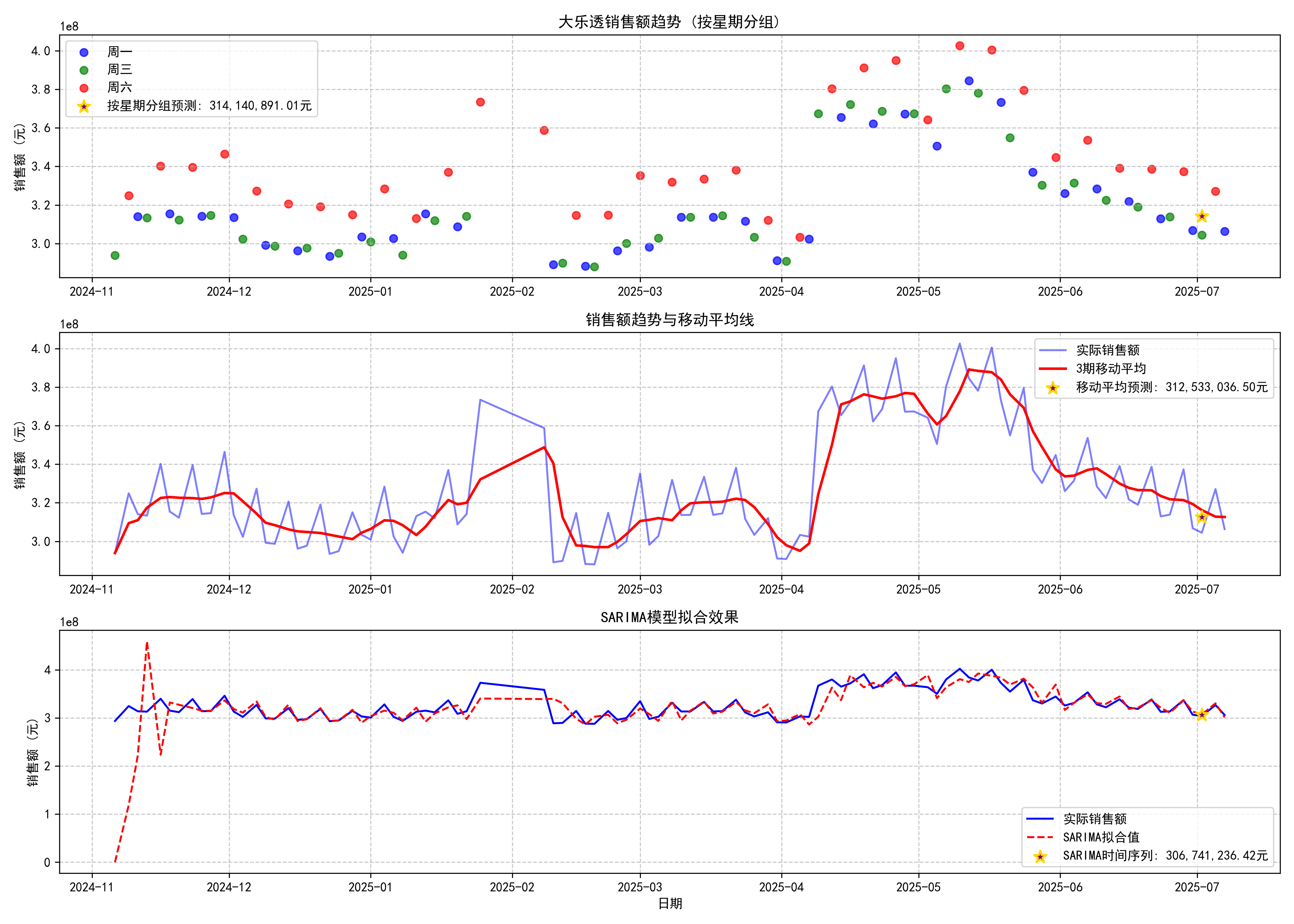
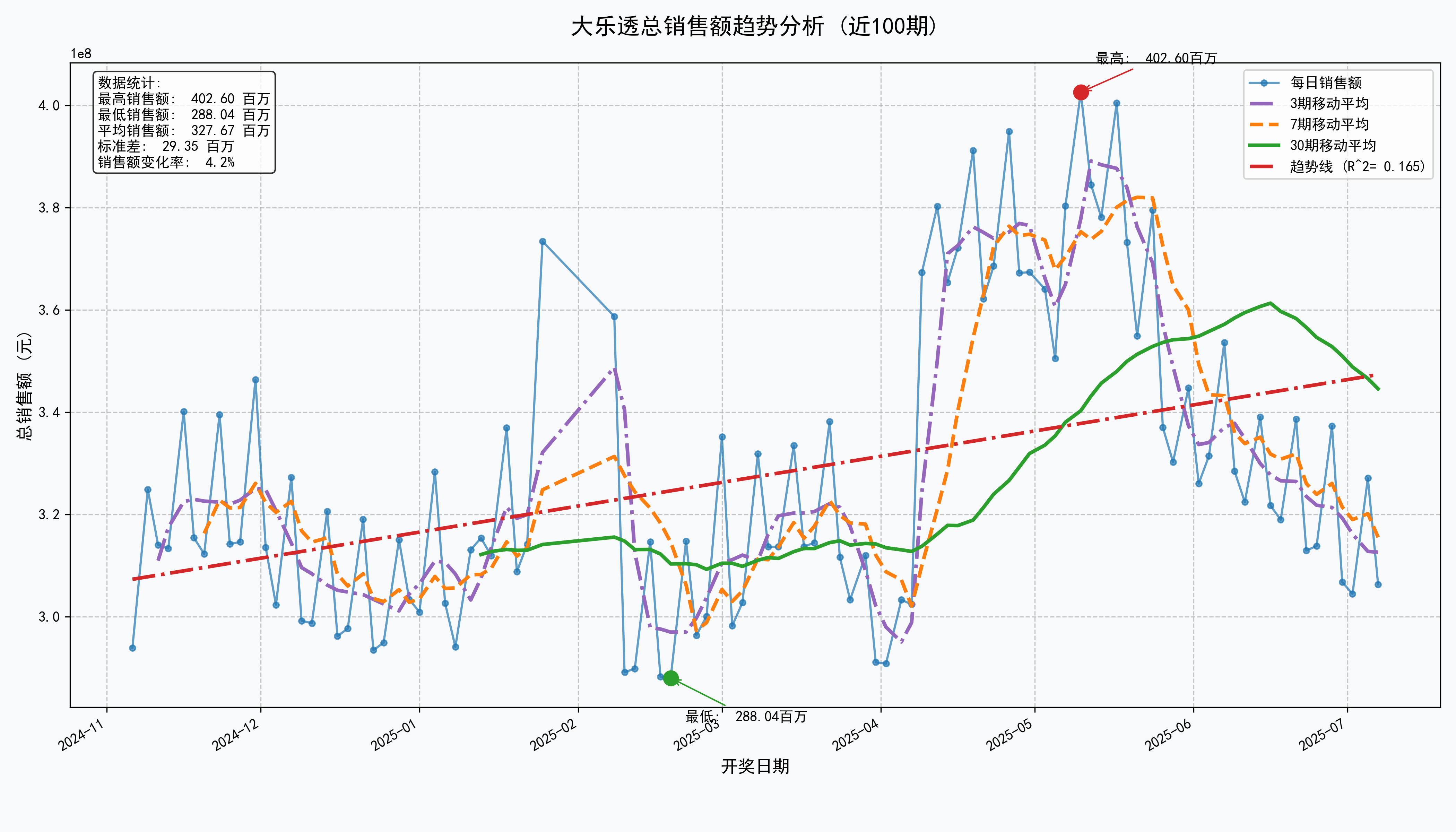










****

**4.2前后区号码频率统计与可视化及大乐透号码预测**

**4.2.1实现步骤（T4\_2.py）**

1. 数据加载与预处理：读取大乐透历史开奖数据Excel文件，将前区和后区号码字符串转换为数值列表；
2. 整体统计分析：计算前区号码(1-35)和后区号码(1-12)的整体出现频率，统计前区-后区号码组合的出现频率；
3. 按星期分组统计：将数据按开奖日期星期分组（周一、周三、周六），分别计算各组的前区/后区频率和组合频率；
4. 数据可视化：生成整体和各星期的前区/后区频率分布柱状图以及组合热力图；
5. 号码预测：基于整体数据和星期三历史数据，采用6种策略预测下一期（周三）的号码组合；
6. 结果输出：保存频率统计CSV文件、可视化图表和预测结果文本文件。

**4.2.2重要库函数**

pandas：数据处理和分析

numpy：数值计算和随机选择

matplotlib：数据可视化

collections.Counter：高效频率统计

os：文件目录管理

**4.2.3重要代码：频率统计、可视化与预测**

# 整体频率统计（不区分星期）

def generate\_full\_stats():

    # 前区号码频率统计

    front\_counts = pd.Series([num for sublist in df['前区号码'] for num in sublist]).value\_counts().sort\_index()

    front\_counts\_df = front\_counts.reset\_index()

    front\_counts\_df.columns = ['前区号码', '出现次数']

    front\_counts\_df.to\_csv('整体前区频率统计.csv', index=False, encoding='utf\_8\_sig')

    # 后区号码频率统计

    back\_counts = pd.Series([num for sublist in df['后区号码'] for num in sublist]).value\_counts().sort\_index()

    back\_counts\_df = back\_counts.reset\_index()

    back\_counts\_df.columns = ['后区号码', '出现次数']

    back\_counts\_df.to\_csv('整体后区频率统计.csv', index=False, encoding='utf\_8\_sig')

    # 组合频率统计

    heatmap\_data = pd.DataFrame(0, index=range(1, 36), columns=range(1, 13))

    for \_, row in df.iterrows():

        for f in row['前区号码']:

            for b in row['后区号码']:

                heatmap\_data.loc[f, b] += 1

    heatmap\_data.to\_csv('整体前后区组合频率统计.csv', index=True, index\_label='前区号码\后区号码', encoding='utf\_8\_sig')

    # 可视化前区频率

    plt.figure(figsize=(15, 6))

    front\_counts.plot(kind='bar', color='skyblue')

    plt.title('大乐透前区号码出现频率 (近100期)', fontsize=15)

    plt.xlabel('号码', fontsize=12)

    plt.ylabel('出现次数', fontsize=12)

    plt.xticks(rotation=0)

    plt.grid(axis='y', linestyle='--', alpha=0.7)

    plt.tight\_layout()

    plt.savefig('整体前区号码频率分布图.png', dpi=300)

    plt.show()

    # 可视化后区频率

    plt.figure(figsize=(10, 6))

    back\_counts.plot(kind='bar', color='lightgreen')

    plt.title('大乐透后区号码出现频率 (近100期)', fontsize=15)

    plt.xlabel('号码', fontsize=12)

    plt.ylabel('出现次数', fontsize=12)

    plt.xticks(rotation=0)

    plt.grid(axis='y', linestyle='--', alpha=0.7)

    plt.tight\_layout()

    plt.savefig('整体后区号码频率分布图.png', dpi=300)

    plt.show()

    # 绘制热力图

    plt.figure(figsize=(12, 8))

    plt.imshow(heatmap\_data, cmap='YlOrRd', aspect='auto')

    plt.colorbar(label='组合出现次数')

    plt.title('前区-后区号码组合热力图', fontsize=15)

    plt.xlabel('后区号码', fontsize=12)

    plt.ylabel('前区号码', fontsize=12)

    plt.xticks(range(12), range(1, 13))

    plt.yticks(range(35), range(1, 36))

    plt.tight\_layout()

    plt.savefig('整体前后区组合热力图.png', dpi=300)

plt.show()

# 按星期生成统计

wednesday\_data = None

for day, group in grouped:

    # 保存星期三的数据用于预测

    if day == '星期三':

        wednesday\_data = group.copy()

    # 前区号码频率统计

    front\_counts = pd.Series([num for sublist in group['前区号码'] for num in sublist]).value\_counts().sort\_index()

    front\_counts\_df = front\_counts.reset\_index()

    front\_counts\_df.columns = ['前区号码', '出现次数']

    front\_counts\_df.to\_csv(f'星期统计/{day}\_前区频率统计.csv', index=False, encoding='utf\_8\_sig')

    # 后区号码频率统计

    back\_counts = pd.Series([num for sublist in group['后区号码'] for num in sublist]).value\_counts().sort\_index()

    back\_counts\_df = back\_counts.reset\_index()

    back\_counts\_df.columns = ['后区号码', '出现次数']

    back\_counts\_df.to\_csv(f'星期统计/{day}\_后区频率统计.csv', index=False, encoding='utf\_8\_sig')

    # 组合频率统计

    heatmap\_data = pd.DataFrame(0, index=range(1, 36), columns=range(1, 13))

    for \_, row in group.iterrows():

        for f in row['前区号码']:

            for b in row['后区号码']:

                heatmap\_data.loc[f, b] += 1

    heatmap\_data.to\_csv(f'星期统计/{day}\_前后区组合频率统计.csv', index=True, index\_label='前区号码\后区号码',

                        encoding='utf\_8\_sig')

    # 生成可视化图表

    plt.figure(figsize=(12, 6))

    front\_counts.plot(kind='bar', color='skyblue')

    plt.title(f'大乐透前区号码出现频率 ({day})', fontsize=15)

    plt.xlabel('号码', fontsize=12)

    plt.ylabel('出现次数', fontsize=12)

    plt.xticks(rotation=0)

    plt.grid(axis='y', linestyle='--', alpha=0.7)

    plt.tight\_layout()

    plt.savefig(f'星期统计/{day}\_前区频率分布图.png', dpi=300)

    plt.show()

    plt.figure(figsize=(8, 6))

    back\_counts.plot(kind='bar', color='lightgreen')

    plt.title(f'大乐透后区号码出现频率 ({day})', fontsize=15)

    plt.xlabel('号码', fontsize=12)

    plt.ylabel('出现次数', fontsize=12)

    plt.xticks(rotation=0)

    plt.grid(axis='y', linestyle='--', alpha=0.7)

    plt.tight\_layout()

    plt.savefig(f'星期统计/{day}\_后区频率分布图.png', dpi=300)

    plt.show()

    plt.figure(figsize=(12, 8))

    plt.imshow(heatmap\_data, cmap='YlOrRd', aspect='auto')

    plt.colorbar(label='组合出现次数')

    plt.title(f'大乐透前区-后区号码组合热力图 ({day})', fontsize=15)

    plt.xlabel('后区号码', fontsize=12)

    plt.ylabel('前区号码', fontsize=12)

    plt.xticks(range(12), range(1, 13))

    plt.yticks(range(35), range(1, 36))

    plt.tight\_layout()

    plt.savefig(f'星期统计/{day}\_前后区组合热力图.png', dpi=300)

    plt.show()

# 预测函数 - 基于整体数据和星期三历史数据

def predict\_numbers(wed\_data, full\_df):

    # 计算整体频率

    full\_front\_counts = Counter([num for sublist in full\_df['前区号码'] for num in sublist])

    full\_back\_counts = Counter([num for sublist in full\_df['后区号码'] for num in sublist])

    # 计算星期三频率

    wed\_front\_counts = Counter([num for sublist in wed\_data['前区号码'] for num in sublist])

    wed\_back\_counts = Counter([num for sublist in wed\_data['后区号码'] for num in sublist])

    # 组合频率统计（星期三）

    wed\_combo\_counts = {}

    for \_, row in wed\_data.iterrows():

        for f in row['前区号码']:

            for b in row['后区号码']:

                wed\_combo\_counts[(f, b)] = wed\_combo\_counts.get((f, b), 0) + 1

    # 1. 保守策略 - 加权热号组合

    # 创建加权频率（50%星期三频率 + 50%整体频率）

    weighted\_front = {}

    for num in range(1, 36):

        wed\_count = wed\_front\_counts.get(num, 0)

        full\_count = full\_front\_counts.get(num, 0)

        weighted\_front[num] = 0.5 \* wed\_count + 0.5 \* full\_count

    weighted\_back = {}

    for num in range(1, 13):

        wed\_count = wed\_back\_counts.get(num, 0)

        full\_count = full\_back\_counts.get(num, 0)

        weighted\_back[num] = 0.6 \* wed\_count + 0.4 \* full\_count

    # 选择加权频率最高的号码

    front\_conservative = sorted(weighted\_front, key=weighted\_front.get, reverse=True)[:5]

    front\_conservative.sort()

    back\_conservative = sorted(weighted\_back, key=weighted\_back.get, reverse=True)[:2]

    back\_conservative.sort()

    # 2. 平衡策略 - 热号+潜力号

    # 前区：3个高频号 + 2个中频号

    # 高频：加权值在前20%的号码

    front\_values = list(weighted\_front.values())

    high\_threshold = np.percentile(front\_values, 80)

    mid\_threshold = np.percentile(front\_values, 50)

    high\_front = [num for num, val in weighted\_front.items() if val >= high\_threshold]

    mid\_front = [num for num, val in weighted\_front.items() if mid\_threshold <= val < high\_threshold]

    if len(high\_front) < 3:

        # 如果高频号码不足，补充整体高频

        high\_front = sorted(full\_front\_counts, key=full\_front\_counts.get, reverse=True)[:10]

    if len(mid\_front) < 5:

        # 如果中频号码不足，补充整体中频

        mid\_front = [num for num in range(1, 36) if 5 <= full\_front\_counts.get(num, 0) <= 7]

    # 随机选择

    front\_balanced = list(np.random.choice(high\_front, min(3, len(high\_front)), replace=False))

    if len(mid\_front) > 0:

        front\_balanced += list(np.random.choice(mid\_front, 5 - len(front\_balanced), replace=False))

    else:

    # 如果没有中频，补充高频

        front\_balanced += list(np.random.choice(high\_front, 5 - len(front\_balanced), replace=False))

    front\_balanced.sort()

    # 后区：1个高频 + 1个中频

    back\_values = list(weighted\_back.values())

    high\_threshold\_b = np.percentile(back\_values, 80)

    mid\_threshold\_b = np.percentile(back\_values, 50)

    high\_back = [num for num, val in weighted\_back.items() if val >= high\_threshold\_b]

    mid\_back = [num for num, val in weighted\_back.items() if mid\_threshold\_b <= val < high\_threshold\_b]

    if len(high\_back) < 1:

        high\_back = sorted(full\_back\_counts, key=full\_back\_counts.get, reverse=True)[:3]

    if len(mid\_back) < 1:

        mid\_back = [num for num in range(1, 13) if 3 <= full\_back\_counts.get(num, 0) <= 4]

    back\_balanced = list(np.random.choice(high\_back, 1, replace=False))

    if len(mid\_back) > 0:

        back\_balanced += list(np.random.choice(mid\_back, 1, replace=False))

    else:

        back\_balanced += list(np.random.choice(high\_back, 1, replace=False))

    back\_balanced.sort()

    # 3. 激进策略 - 热冷结合

    # 前区：2个高频 + 1个中频 + 2个低频

    low\_front = [num for num in range(1, 36) if weighted\_front.get(num, 0) < np.percentile(front\_values, 30)]

    if len(low\_front) < 2:

        low\_front = [num for num in range(1, 36) if full\_front\_counts.get(num, 0) < 4]

    front\_aggressive = list(np.random.choice(high\_front, 2, replace=False))

    if len(mid\_front) > 0:

        front\_aggressive += list(np.random.choice(mid\_front, 1, replace=False))

    else:

        front\_aggressive += list(np.random.choice(high\_front, 1, replace=False))

    if len(low\_front) > 0:

        front\_aggressive += list(np.random.choice(low\_front, 2, replace=False))

    else:

        front\_aggressive += list(np.random.choice(mid\_front, 2, replace=False))

    front\_aggressive.sort()

    # 后区：1个高频 + 1个低频

    low\_back = [num for num in range(1, 13) if weighted\_back.get(num, 0) < np.percentile(back\_values, 30)]

    if len(low\_back) < 1:

        low\_back = [num for num in range(1, 13) if full\_back\_counts.get(num, 0) < 3]

    back\_aggressive = list(np.random.choice(high\_back, 1, replace=False))

    if len(low\_back) > 0:

        back\_aggressive += list(np.random.choice(low\_back, 1, replace=False))

    else:

        back\_aggressive += list(np.random.choice(mid\_back, 1, replace=False))

    back\_aggressive.sort()

    # 4. 组合优化策略 - 基于最佳组合

    # 找出最高频的前区-后区组合（星期三）

    top\_combos = sorted(wed\_combo\_counts.items(), key=lambda x: x[1], reverse=True)[:20]

    # 创建组合权重（考虑整体频率）

    combo\_weights = {}

    for (f, b), count in top\_combos:

    # 组合权重 = 星期三组合频率 \* 0.7 + (整体前区频率 + 整体后区频率) \* 0.3

        combo\_weights[(f, b)] = 0.7 \* count + 0.3 \* (full\_front\_counts[f] + full\_back\_counts[b])

    # 选择最高权重的组合

    top\_combos\_weighted = sorted(combo\_weights.items(), key=lambda x: x[1], reverse=True)[:20]

    front\_combo = []

    back\_combo = []

    selected\_front = set()

    selected\_back = set()

    for (f, b), weight in top\_combos\_weighted:

        if len(front\_combo) < 5 and f not in selected\_front:

            front\_combo.append(f)

            selected\_front.add(f)

        if len(back\_combo) < 2 and b not in selected\_back:

            back\_combo.append(b)

            selected\_back.add(b)

        if len(front\_combo) == 5 and len(back\_combo) == 2:

            break

    # 如果不足5个前区号码，补充加权高频号码

    if len(front\_combo) < 5:

        remaining = sorted(weighted\_front, key=weighted\_front.get, reverse=True)

        remaining = [num for num in remaining if num not in selected\_front]

        front\_combo.extend(remaining[:5 - len(front\_combo)])

    # 如果不足2个后区号码，补充加权高频号码

    if len(back\_combo) < 2:

        remaining = sorted(weighted\_back, key=weighted\_back.get, reverse=True)

        remaining = [num for num in remaining if num not in selected\_back]

        back\_combo.extend(remaining[:2 - len(back\_combo)])

    front\_combo.sort()

    back\_combo.sort()

    # 5. 星期三高频号码

    # 前区：选择星期三频率最高的5个号码

    wed\_high\_front = [num for num, count in wed\_front\_counts.most\_common(5)]

    wed\_high\_front.sort()

    # 后区：选择星期三频率最高的2个号码

    wed\_high\_back = [num for num, count in wed\_back\_counts.most\_common(2)]

    wed\_high\_back.sort()

    # 6. 整体高频号码

    # 前区：选择整体频率最高的5个号码

    full\_high\_front = [num for num, count in full\_front\_counts.most\_common(5)]

    full\_high\_front.sort()

    # 后区：选择整体频率最高的2个号码

    full\_high\_back = [num for num, count in full\_back\_counts.most\_common(2)]

    full\_high\_back.sort()

    return {

        "保守策略(加权)": (front\_conservative, back\_conservative),

        "平衡策略": (front\_balanced, back\_balanced),

        "激进策略": (front\_aggressive, back\_aggressive),

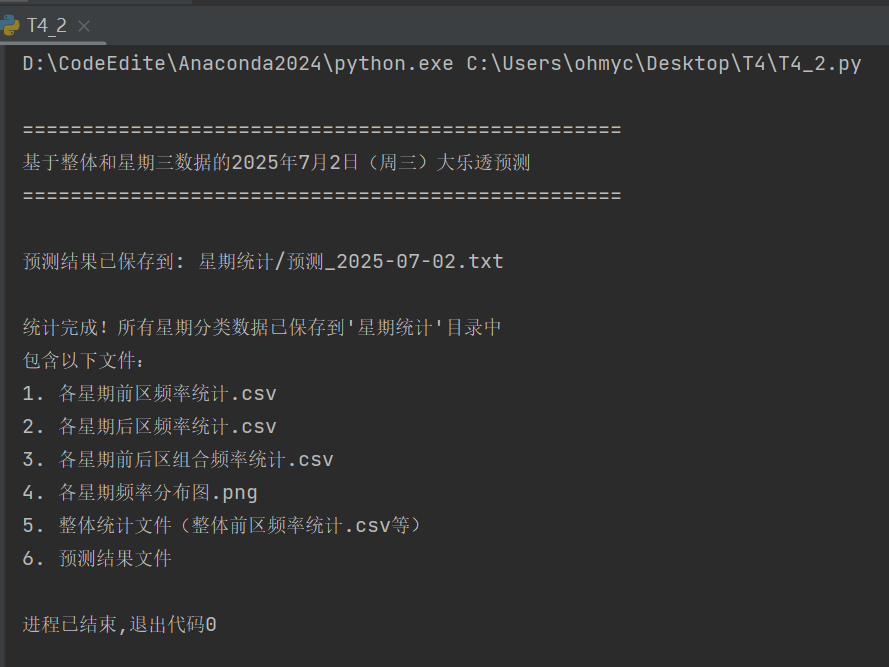
        "组合优化": (front\_combo, back\_combo),

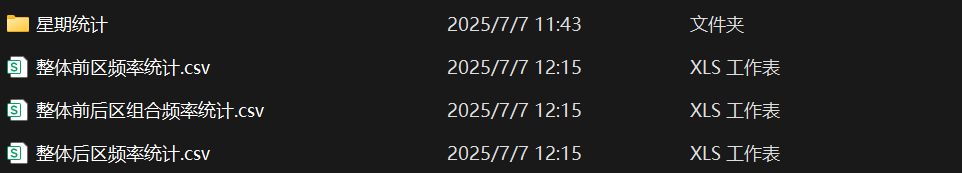
        "星期三高频": (wed\_high\_front, wed\_high\_back),

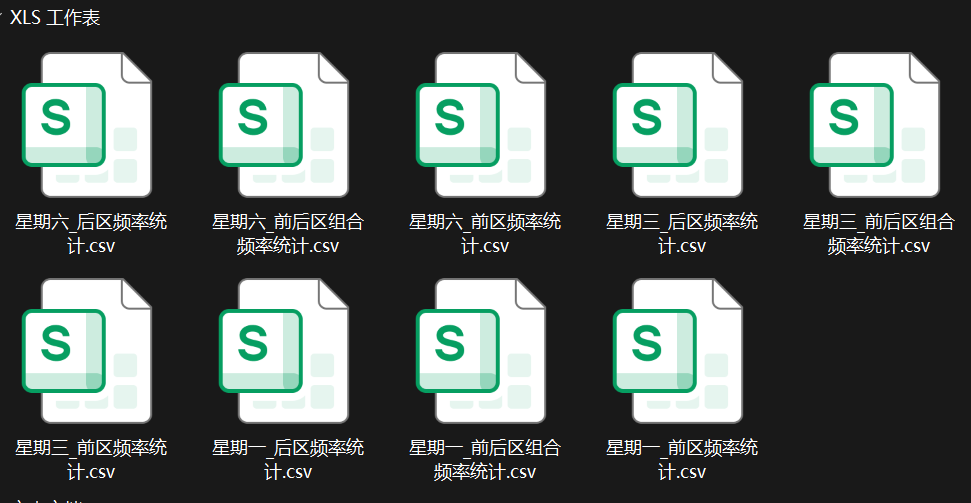
        "整体高频": (full\_high\_front, full\_high\_back)

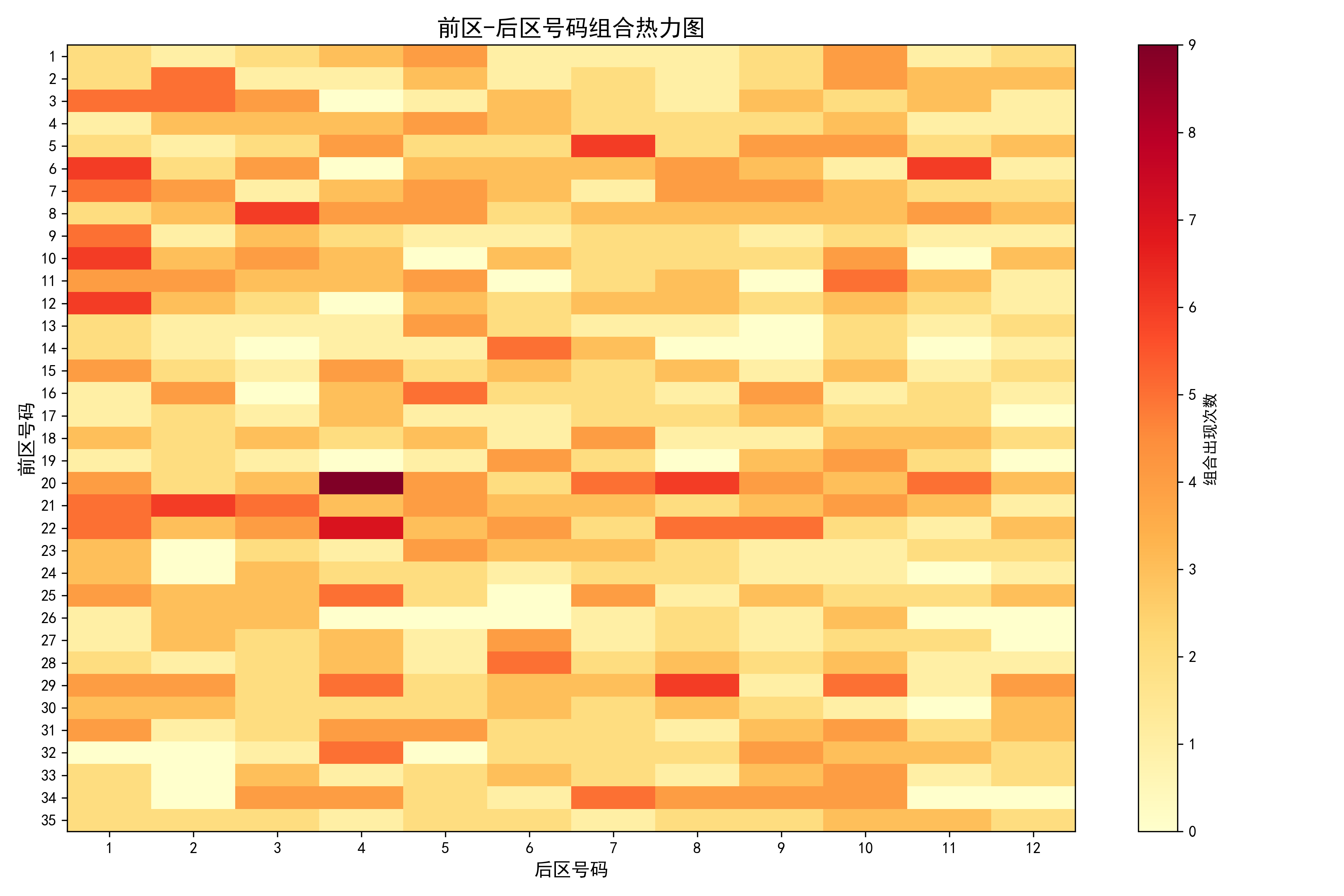
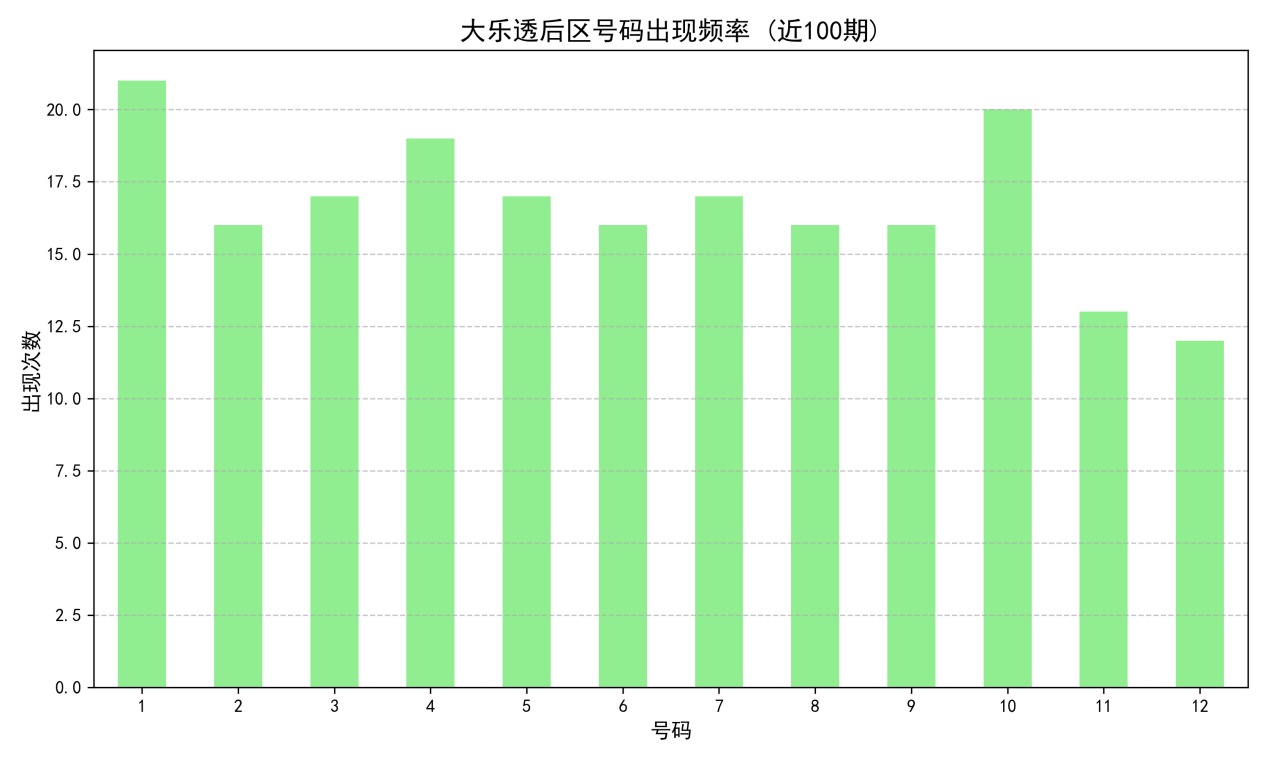
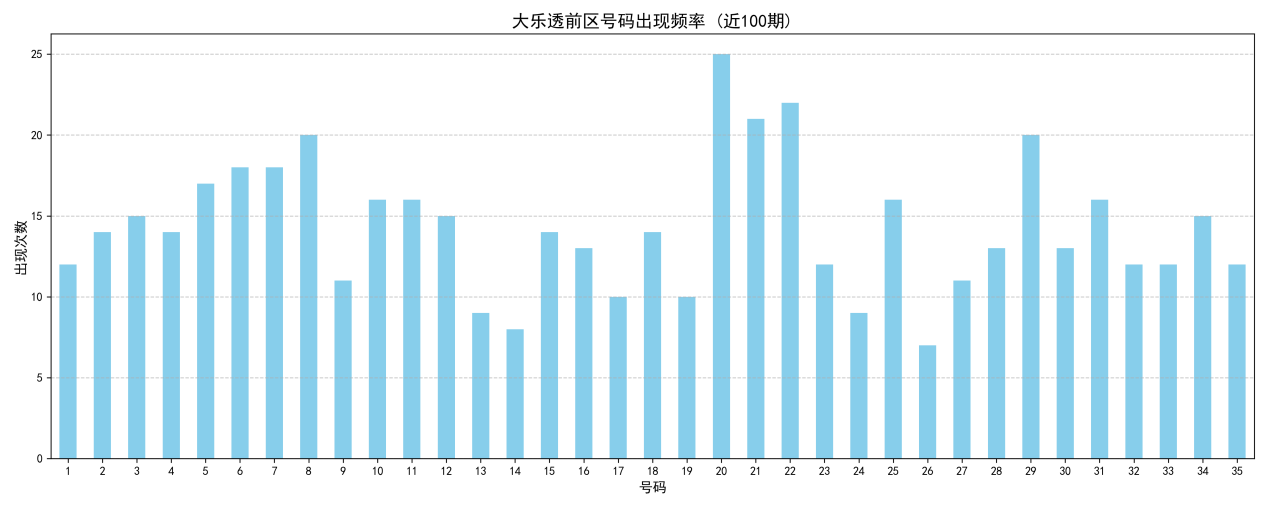
    }

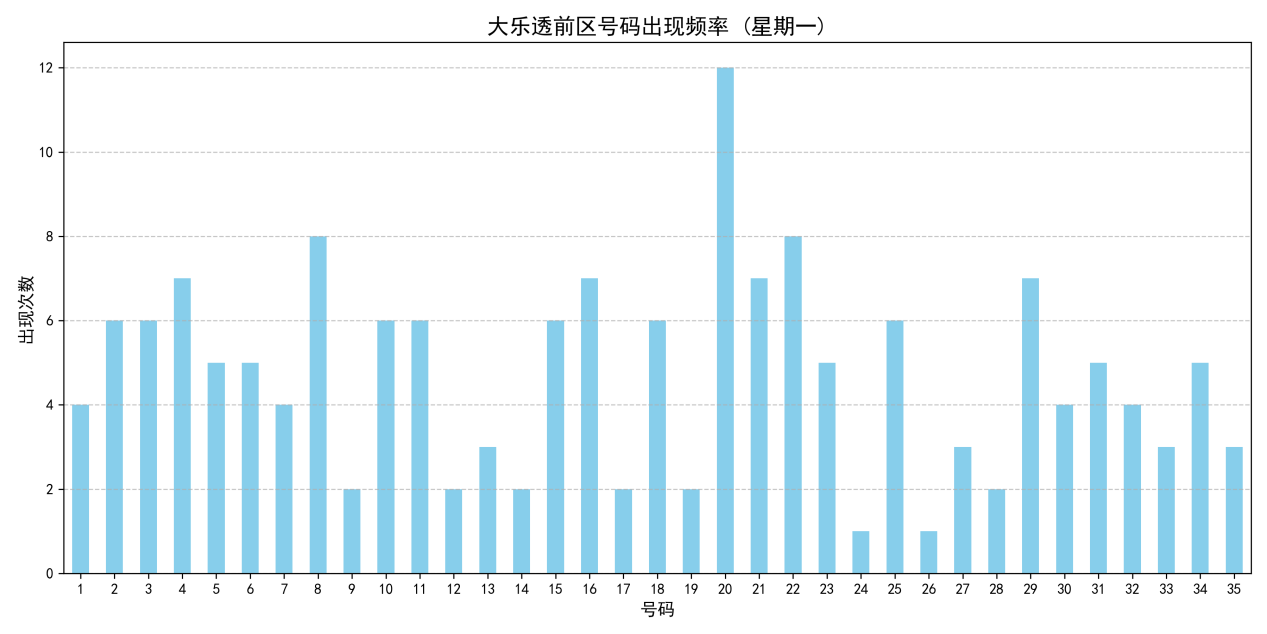
**4.2.4运行与可视化结果**

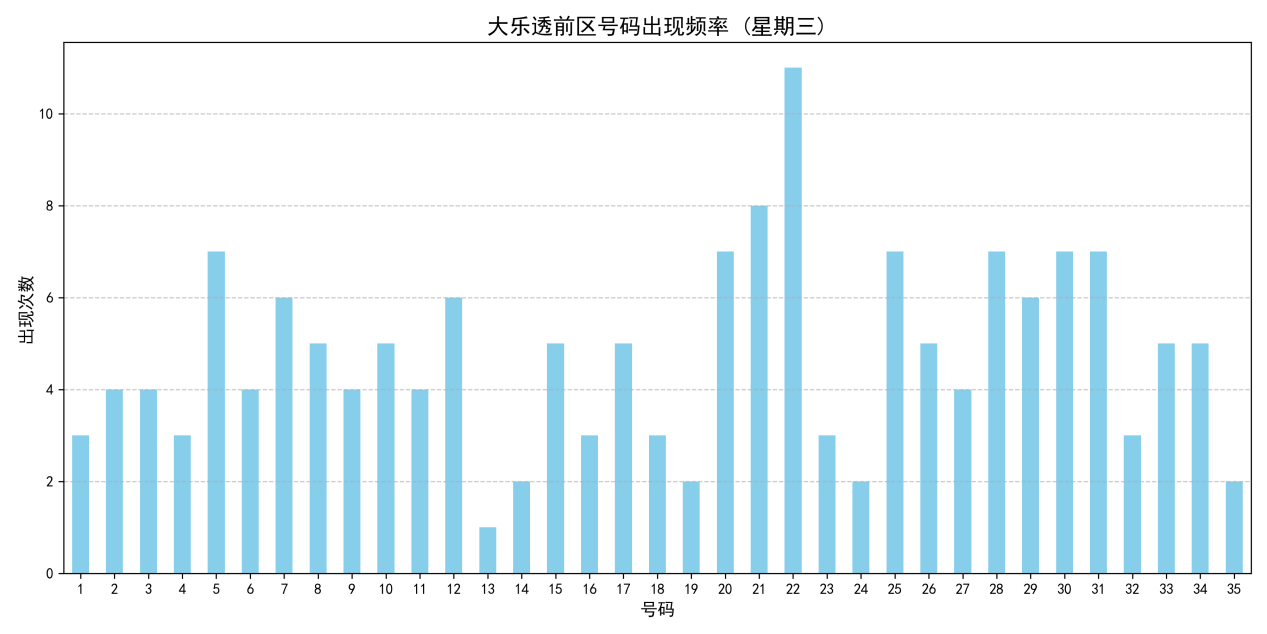
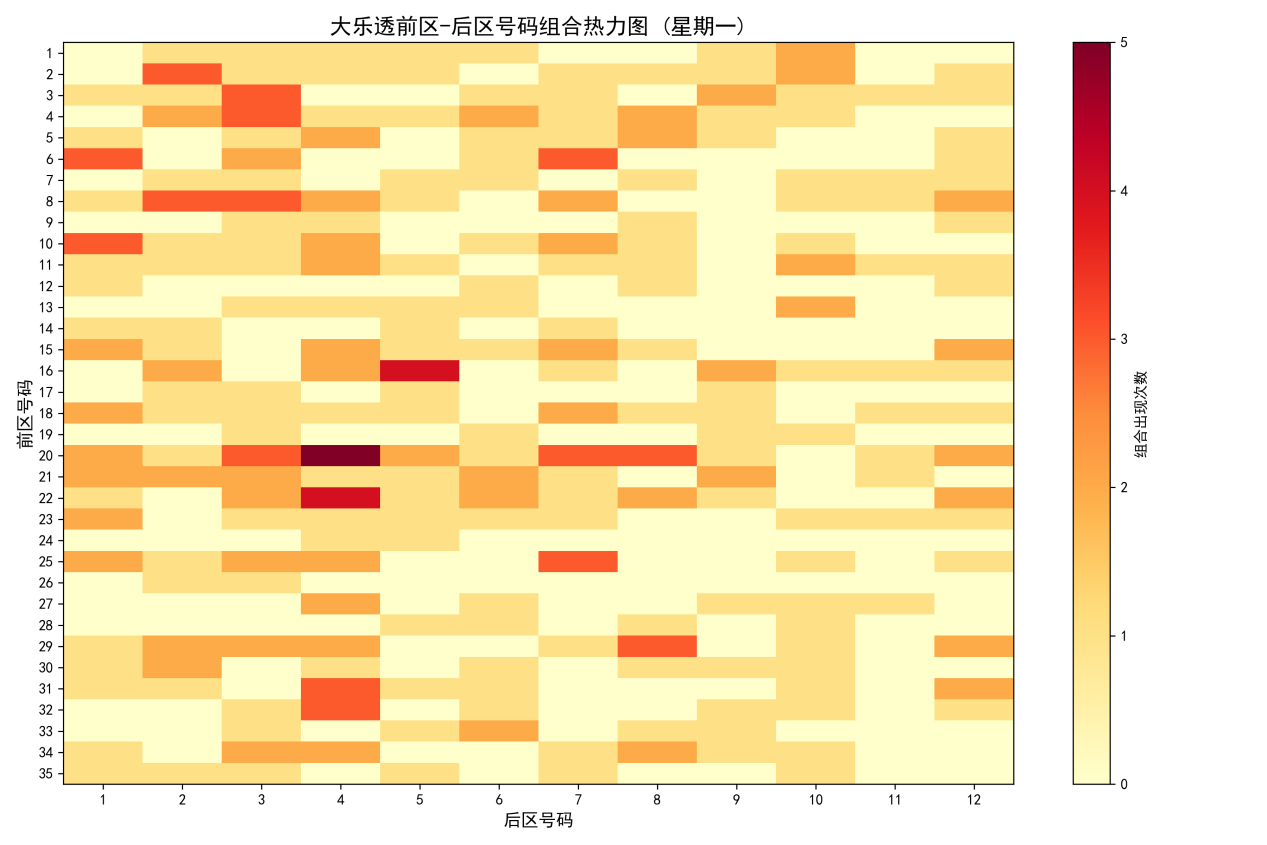
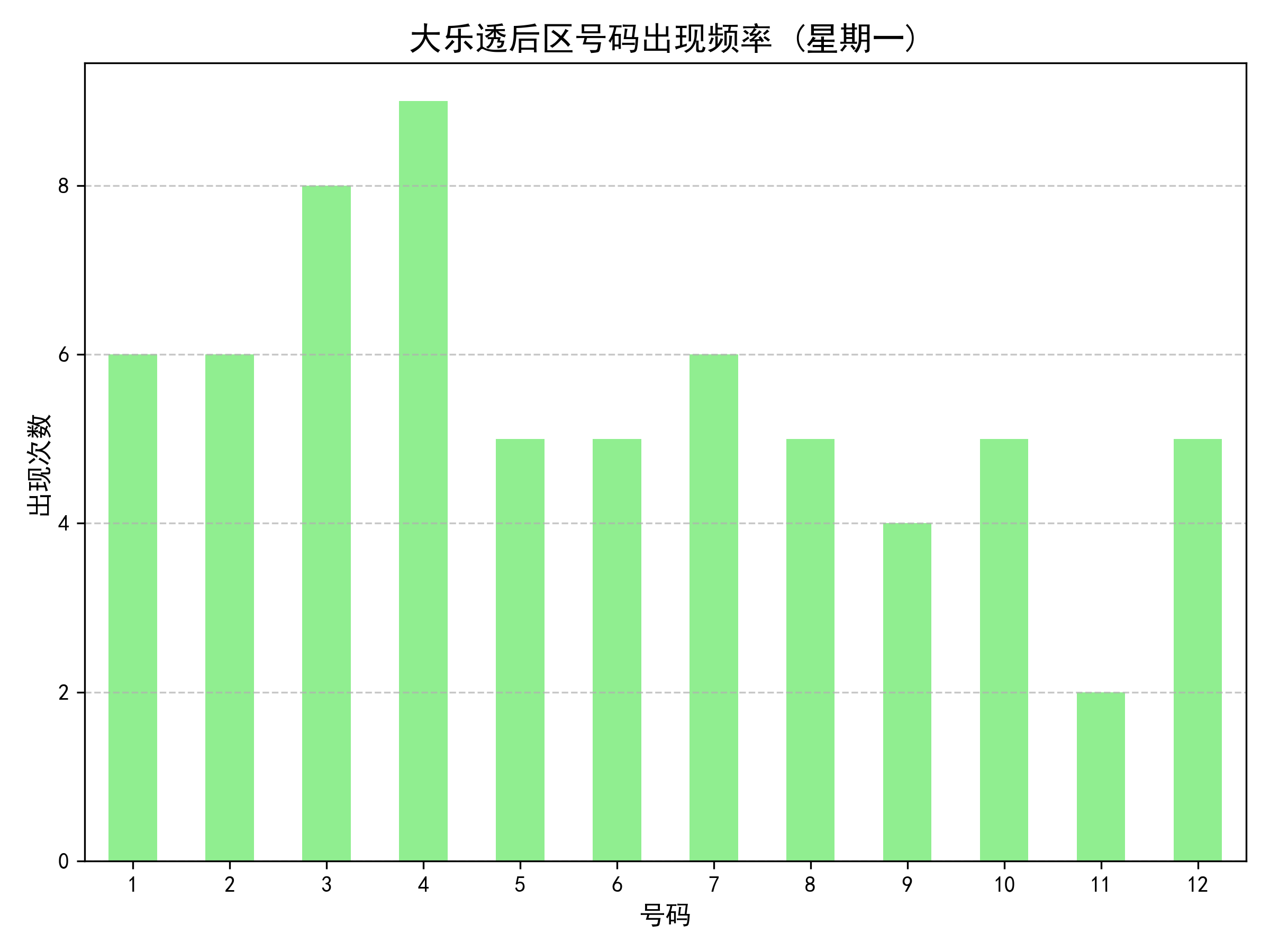


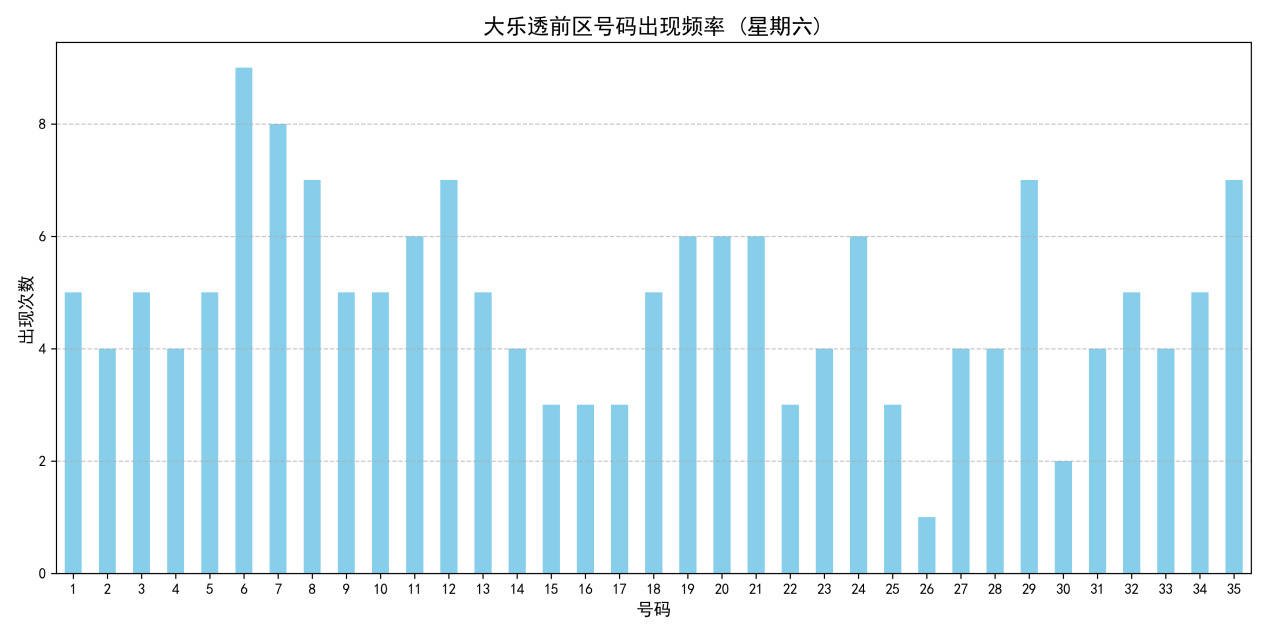
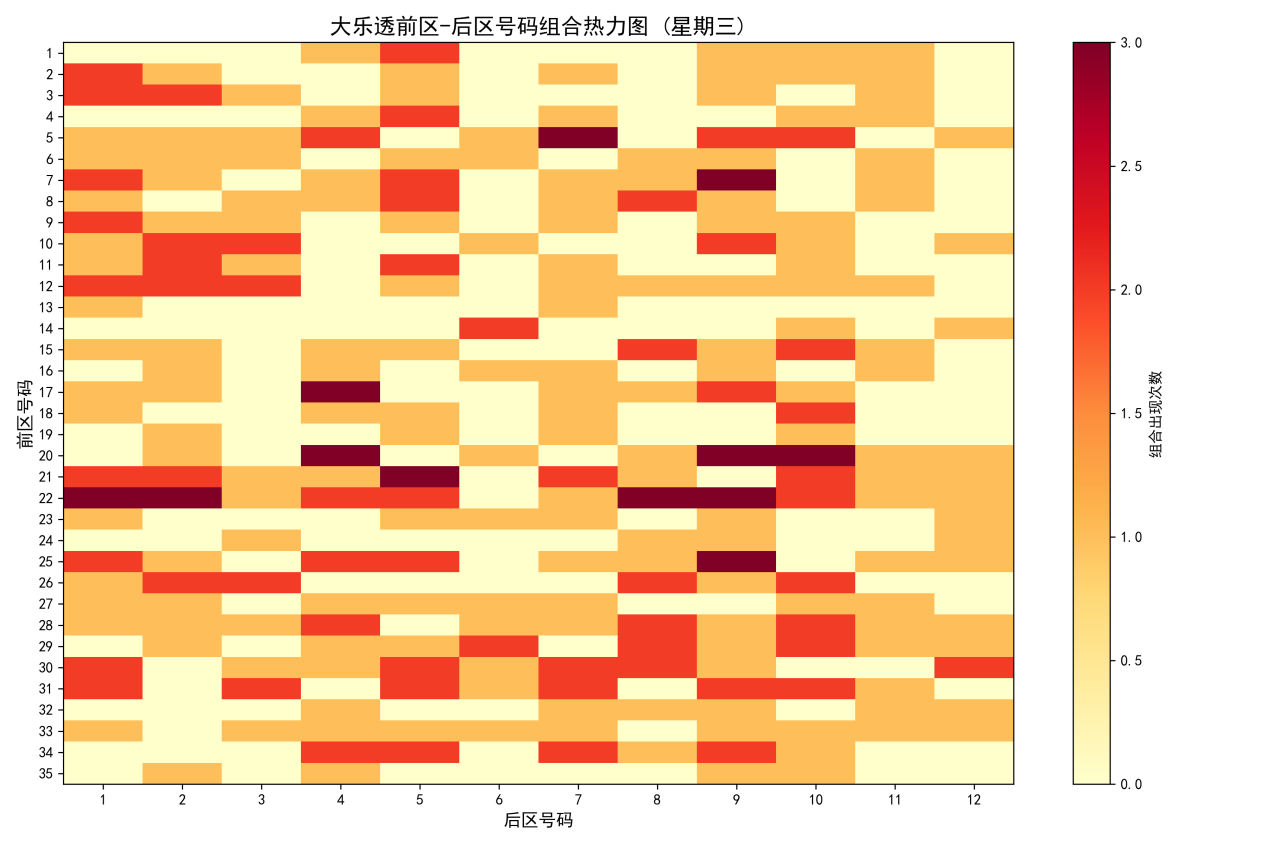
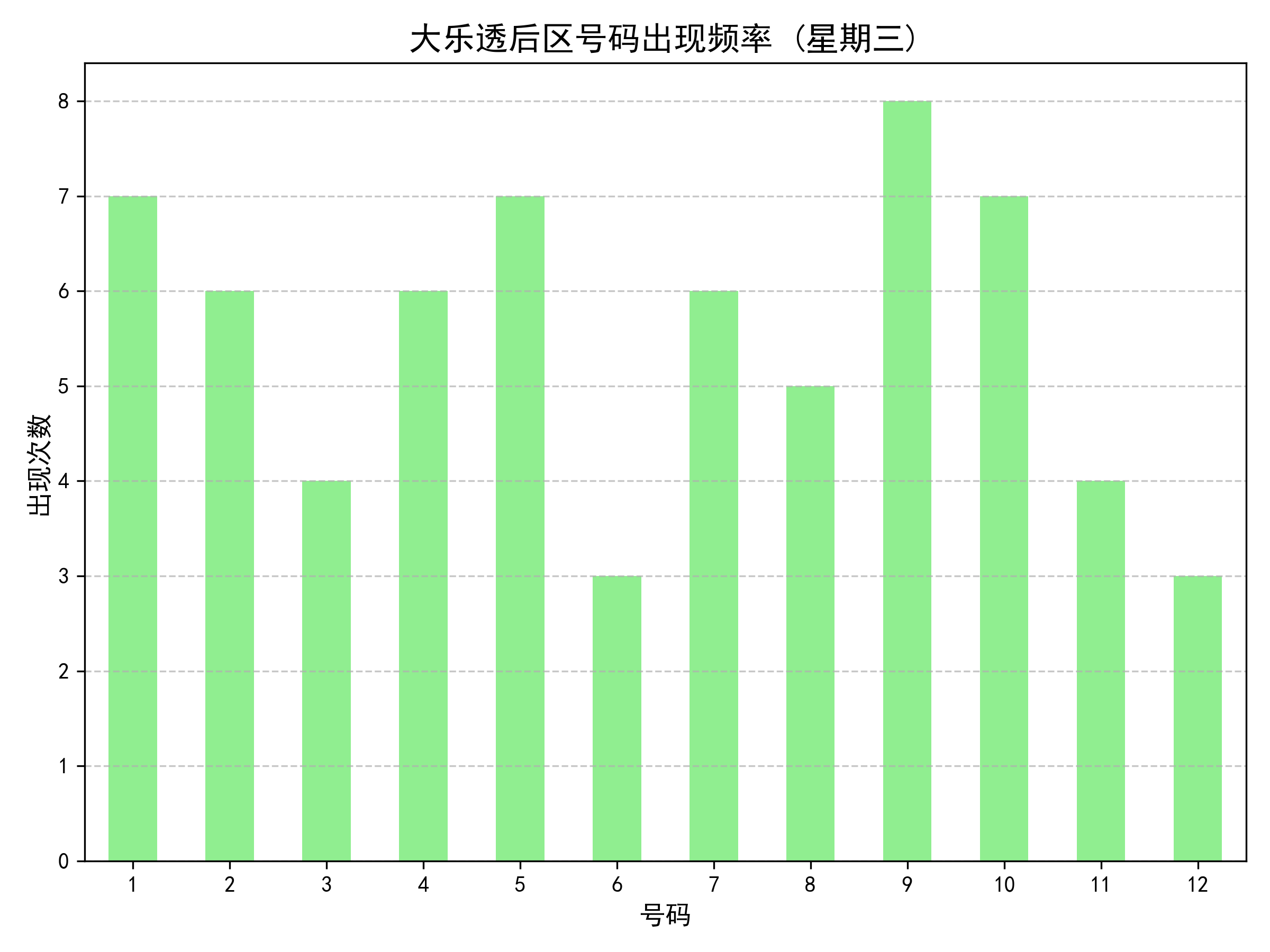


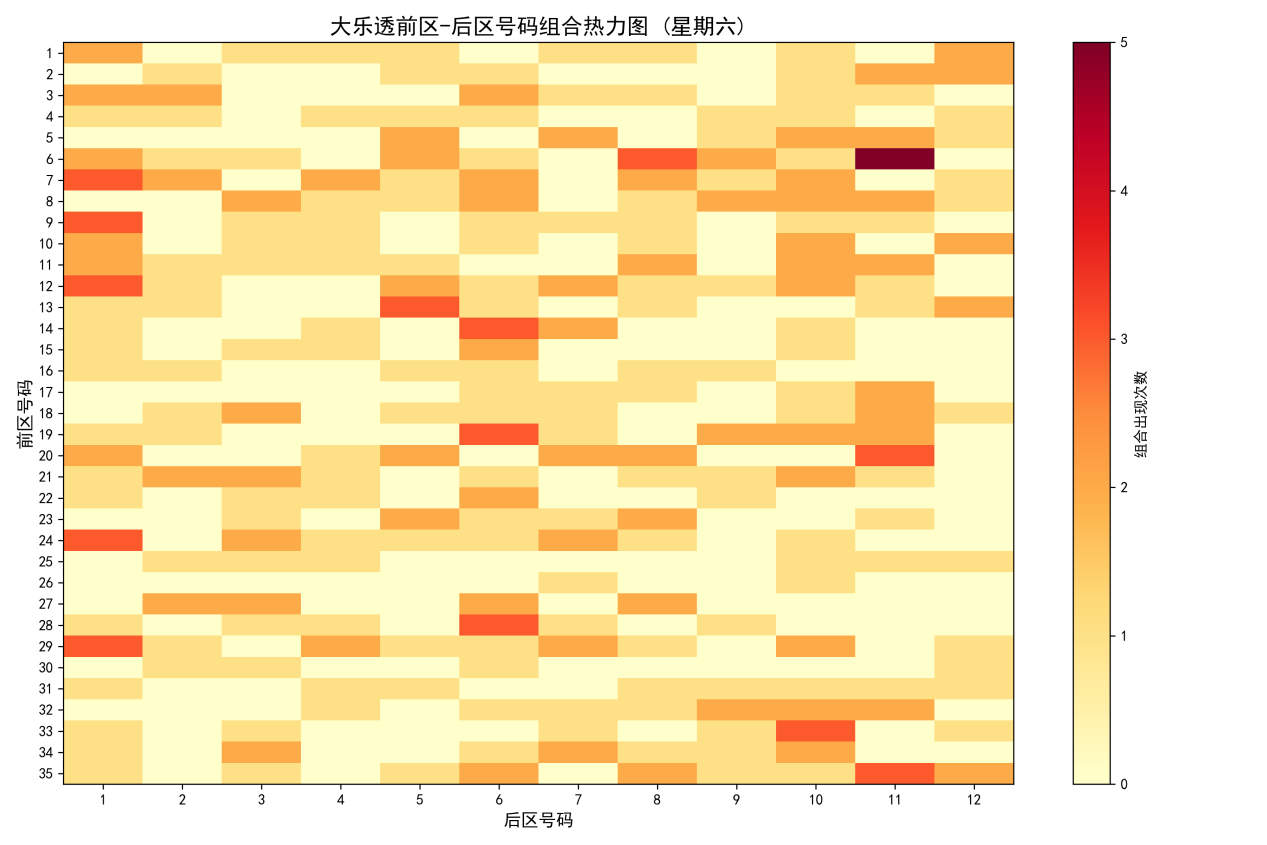
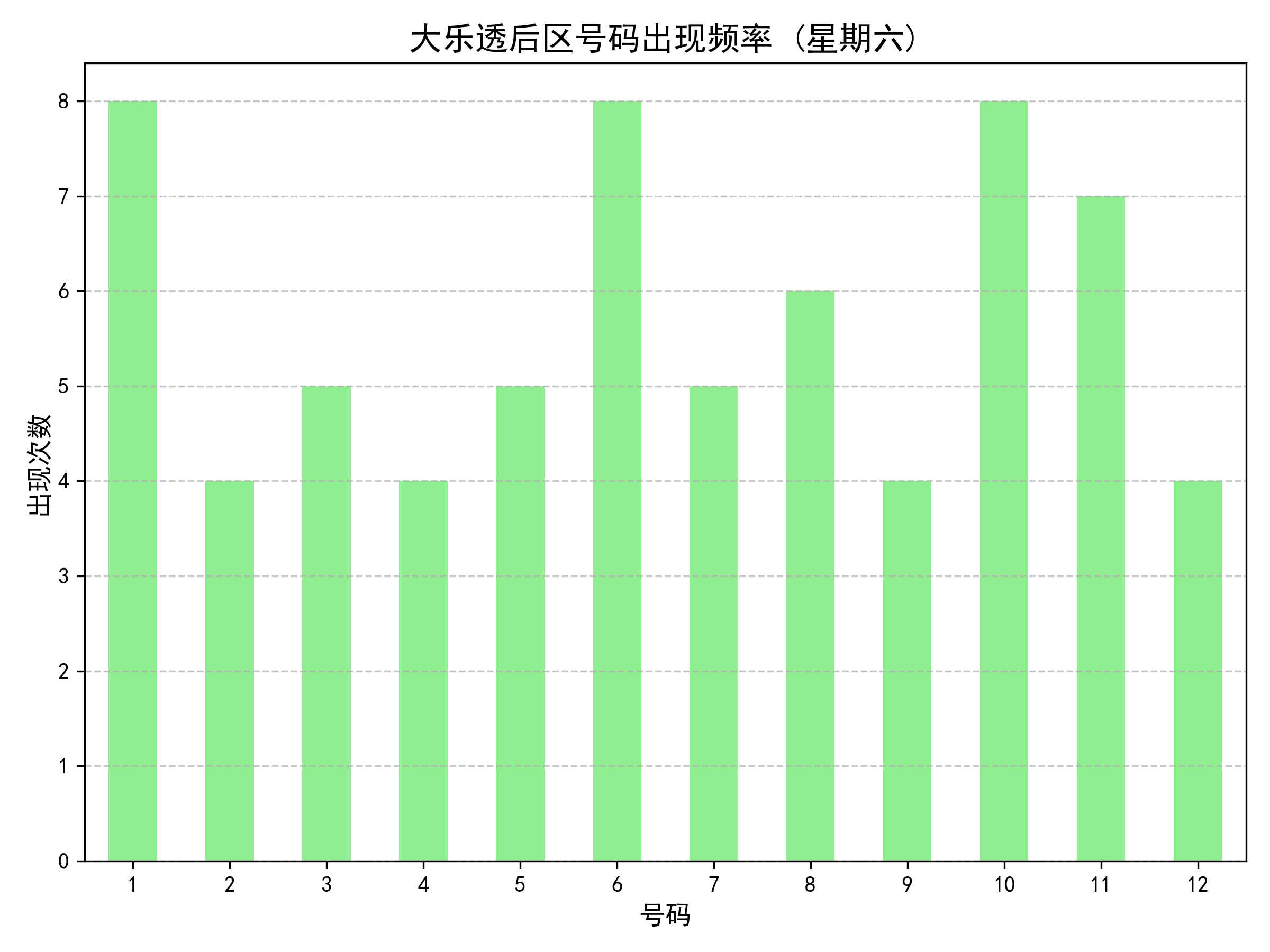




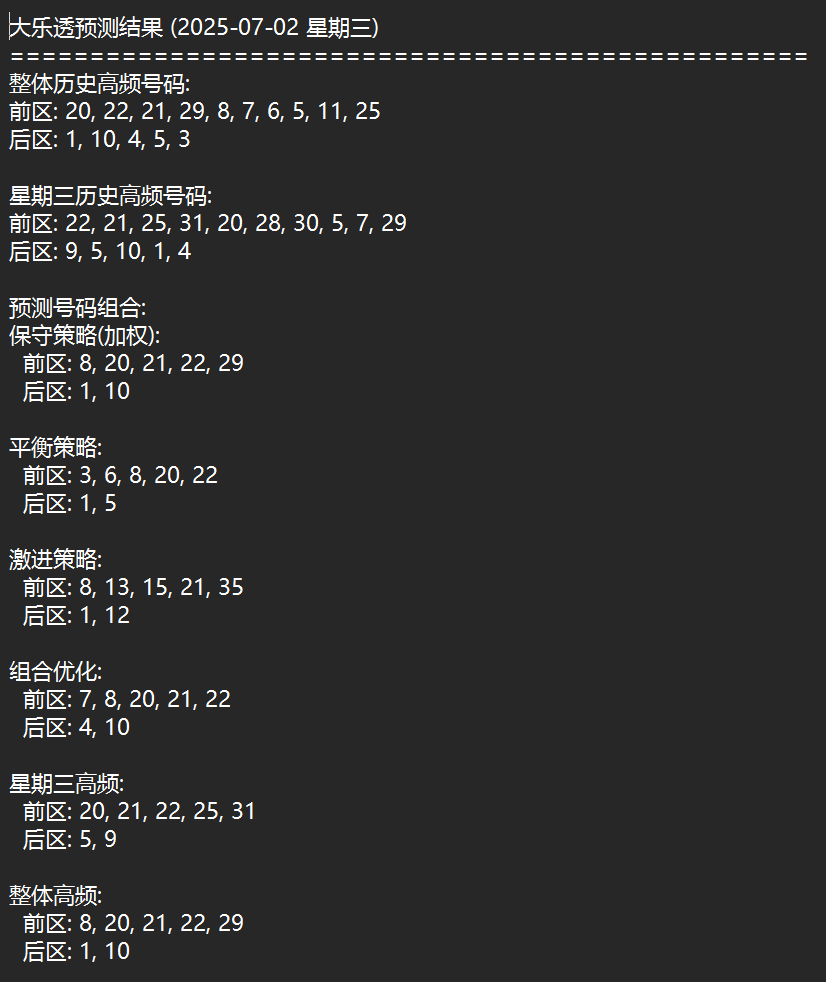








**4.2.5预测结果**



**4.3按星期统计开奖号码和总销售额与特征对比**

**4.3.1实现步骤**

1. 数据加载与预处理：读取CSV文件，处理日期、销售额，拆分前区和后区号码，并验证星期列。
2. 销售额分析：按星期分组，计算销售额的统计量（均值、中位数、标准差、计数），并通过箱线图、条形图、时间序列图和密度图进行可视化。
3. 号码分布分析：分别统计前区（1-35）和后区（1-12）号码在不同开奖日出现的频率，并通过直方图和热力图进行可视化。
4. 统计检验：对销售额进行方差分析（ANOVA）检验，对号码分布进行卡方检验，判断不同开奖日之间的差异是否显著。

**4.3.2重要库函数**

* pandas: 用于数据处理
* numpy: 数值计算
* matplotlib.pyplot: 绘图
* seaborn: 高级统计图表（箱线图、条形图、密度图、直方图、热力图等）
* scipy.stats: 统计检验

**4.3.3重要代码：销售额分析、号码分布分析、统计检验**

def analyze\_sales(df):

    """分析销售额特征"""

    # 按星期分组统计

    grouped = df.groupby('星期')

    sales\_stats = grouped['总销售额(元)'].agg(['mean', 'median', 'std', 'count'])

    # 销售额可视化

    plt.figure(figsize=(14, 10))

    # 箱线图

    plt.subplot(2, 2, 1)

    sns.boxplot(x='星期', y='总销售额(元)', data=df, order=['星期一', '星期三', '星期六'])

    plt.title('不同开奖日销售额分布')

    plt.ylabel('销售额(百万元)')

    # 条形图

    plt.subplot(2, 2, 2)

    sales\_mean = grouped['总销售额(元)'].mean().div(1000000)

    sales\_mean = sales\_mean.reindex(['星期一', '星期三', '星期六'])

    sales\_mean.plot(kind='bar', color=['skyblue', 'lightgreen', 'salmon'])

    plt.title('平均销售额对比')

    plt.ylabel('平均销售额(百万元)')

    plt.xticks(rotation=0)

    # 时间序列图

    plt.subplot(2, 2, 3)

    for day in ['星期一', '星期三', '星期六']:

        subset = df[df['星期'] == day]

        plt.plot(subset['开奖日期'], subset['总销售额(元)'] / 1000000, 'o-', label=day, alpha=0.7)

    plt.title('销售额随时间变化趋势')

    plt.ylabel('销售额(百万元)')

    plt.legend()

    # 密度图

    plt.subplot(2, 2, 4)

    sns.kdeplot(data=df, x='总销售额(元)', hue='星期', hue\_order=['星期一', '星期三', '星期六'],

                fill=True, common\_norm=False, alpha=0.3, palette='viridis')

    plt.title('销售额密度分布')

    plt.xlabel('销售额(元)')

    plt.tight\_layout()

    plt.savefig('销售额特征.png', dpi=300)

    plt.show()

return sales\_stats

def analyze\_numbers(df):

    """分析号码分布特征"""

    # 按星期分组

    grouped = df.groupby('星期')

    # 分别创建前区和后区的数据列表

    front\_data = {'号码': [], '区域': [], '星期': []}

    back\_data = {'号码': [], '区域': [], '星期': []}

    for day in ['星期一', '星期三', '星期六']:

        # 获取该星期的数据

        day\_data = grouped.get\_group(day) if day in grouped.groups else pd.DataFrame()

        if not day\_data.empty:

            # 处理前区号码

            for nums in day\_data['前区号码']:

                for num in nums:

                    front\_data['号码'].append(num)

                    front\_data['区域'].append('前区')

                    front\_data['星期'].append(day)

            # 处理后区号码

            for nums in day\_data['后区号码']:

                for num in nums:

                    back\_data['号码'].append(num)

                    back\_data['区域'].append('后区')

                    back\_data['星期'].append(day)

    # 创建数据框

    front\_df = pd.DataFrame(front\_data)

    back\_df = pd.DataFrame(back\_data)

    numbers\_df = pd.concat([front\_df, back\_df])

    # 号码频率可视化

    plt.figure(figsize=(16, 12))

    # 前区号码分布

    plt.subplot(2, 2, 1)

    if not front\_df.empty:

        sns.histplot(data=front\_df, x='号码', hue='星期', multiple='dodge',

                     binwidth=1, kde=False, discrete=True,

                     hue\_order=['星期一', '星期三', '星期六'],

                     palette='Set2')

        plt.title('前区号码分布对比')

        plt.xticks(range(1, 36))

        plt.xlim(0.5, 35.5)

    # 后区号码分布

    plt.subplot(2, 2, 2)

    if not back\_df.empty:

        sns.histplot(data=back\_df, x='号码', hue='星期', multiple='dodge',

                     binwidth=1, kde=False, discrete=True,

                     hue\_order=['星期一', '星期三', '星期六'],

                     palette='Set2')

        plt.title('后区号码分布对比')

        plt.xticks(range(1, 13))

        plt.xlim(0.5, 12.5)

    # 热力图 - 前区

    plt.subplot(2, 2, 3)

    if not front\_df.empty:

        front\_pivot = front\_df.groupby(['号码', '星期']).size().unstack().fillna(0)

        sns.heatmap(front\_pivot.T, annot=True, fmt='g', cmap='YlGnBu')

        plt.title('前区号码出现频率热力图')

        plt.ylabel('开奖日')

    # 热力图 - 后区

    plt.subplot(2, 2, 4)

    if not back\_df.empty:

        back\_pivot = back\_df.groupby(['号码', '星期']).size().unstack().fillna(0)

        sns.heatmap(back\_pivot.T, annot=True, fmt='g', cmap='YlGnBu')

        plt.title('后区号码出现频率热力图')

        plt.ylabel('开奖日')

    plt.tight\_layout()

    plt.savefig('号码分布.png', dpi=300)

    plt.show()

    return numbers\_df

def statistical\_analysis(df):

    """执行统计检验"""

    # 按星期分组

    grouped = df.groupby('星期')

    # 准备销售额数据

    mon = grouped.get\_group('星期一')['总销售额(元)'] if '星期一' in grouped.groups else pd.Series(dtype=float)

    wed = grouped.get\_group('星期三')['总销售额(元)'] if '星期三' in grouped.groups else pd.Series(dtype=float)

    sat = grouped.get\_group('星期六')['总销售额(元)'] if '星期六' in grouped.groups else pd.Series(dtype=float)

    # 方差分析 (检验销售额差异)

    if not mon.empty and not wed.empty and not sat.empty:

        \_, p\_sales = stats.f\_oneway(mon, wed, sat)

    else:

        p\_sales = 1.0  # 如果缺少数据，设为不显著

    # 号码分布检验

    results = {}

    for num\_type in ['前区', '后区']:

        # 创建号码列表

        all\_numbers = []

        all\_days = []

        for day in ['星期一', '星期三', '星期六']:

            if day in grouped.groups:

                day\_data = grouped.get\_group(day)

                for nums in day\_data[f'{num\_type}号码']:

                    all\_numbers.extend(nums)

                    all\_days.extend([day] \* len(nums))

        # 创建列联表

        if all\_numbers:

            crosstab = pd.crosstab(

                index=pd.Series(all\_numbers, name='号码'),

                columns=pd.Series(all\_days, name='星期')

            )

            # 确保有足够的数据进行卡方检验

            if crosstab.size > 0 and crosstab.sum().sum() > 0:

                # 卡方检验

                \_, p\_num, \_, \_ = stats.chi2\_contingency(crosstab)

                results[f'{num\_type}号码分布'] = p\_num

            else:

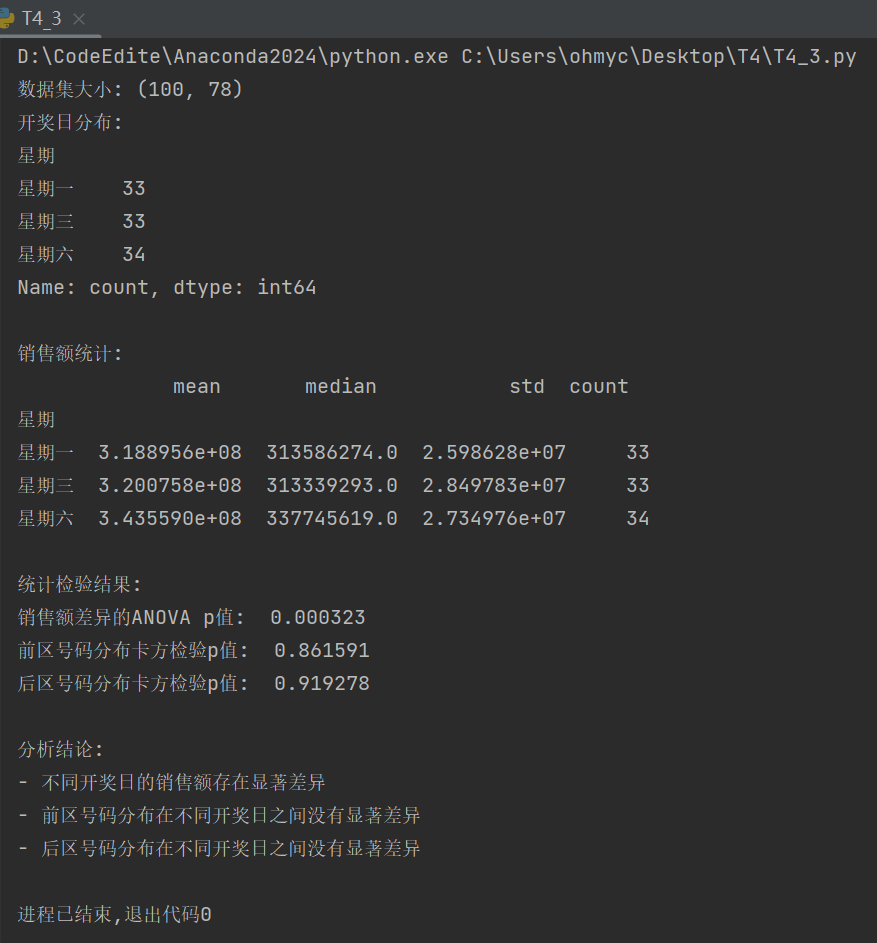
                results[f'{num\_type}号码分布'] = 1.0  # 如果缺少数据，设为不显著

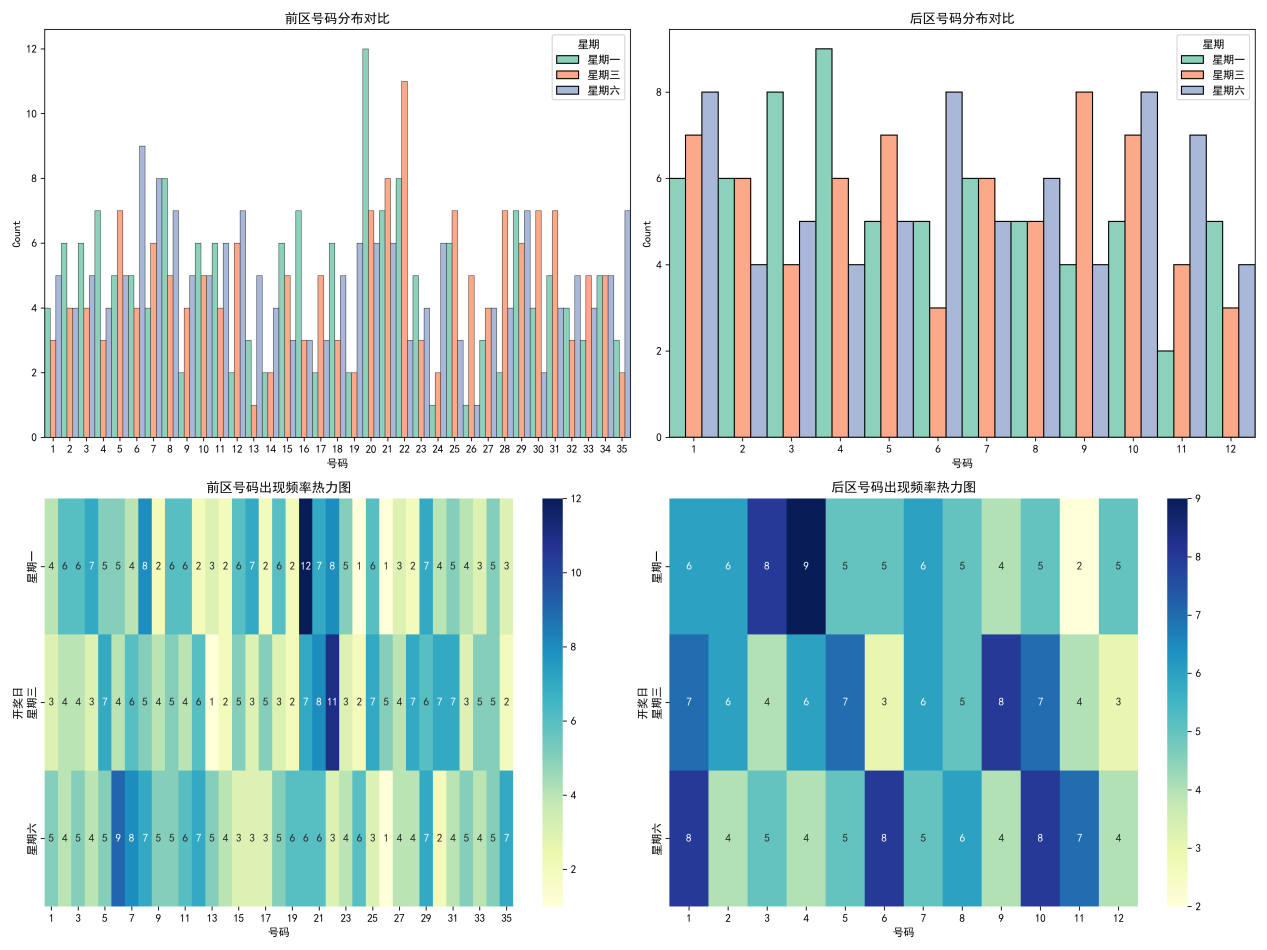
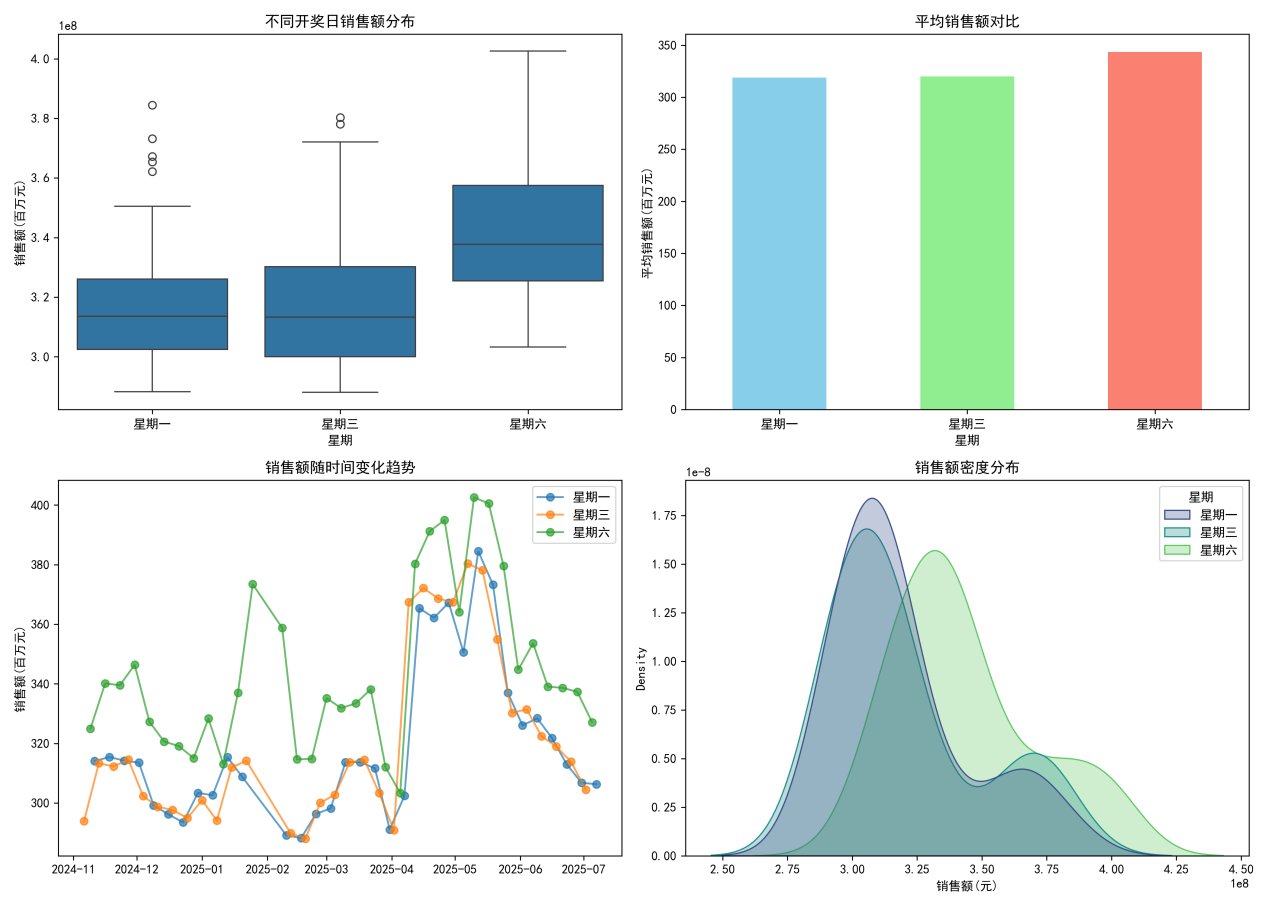
        else:

            results[f'{num\_type}号码分布'] = 1.0

    return p\_sales, results

**4.3.4运行、可视化与分析结果**





**4.4爬取专家数据、统计分析并可视化**

**4.4.1实现步骤**

**数据爬取部分（T4\_4\_1.py）：**

1. 数据源分析：分析“彩民之家”网站（cmzj.net）的专家排行榜和专家详情接口，确定请求参数和返回数据结构。
2. 设置请求头：模拟浏览器访问，设置User-Agent、Referer等头信息（如代码中的headers所示）。
3. 分页爬取专家排行榜：循环处理100页（每页10位专家）数据，获取每位专家的基本信息（如专家ID、排名、名称、成功率等）。
4. 反爬处理：使用随机等待时间。
5. 爬取专家详情：对排行榜中的每位专家，根据其专家ID请求详情接口，获取详细信息（如等级、技能、粉丝数、推荐记录等）。
6. 数据整合：将排行榜中的基本信息与详情信息合并为一个完整记录。
7. 数据存储：将所有专家数据保存为CSV文件。

**趋势分析和销售额预测部分（T4\_4\_2.py）：**

**4.4.2重要库函数**

* requests：网络请求与JSON解析
* pandas：数据结构构建与重组
* tqdm：进度条可视化
* random 和 time: 实现随机等待避免反爬

**4.4.3重要代码：数据爬取、统计分析与可视化展现**

# 配置请求头

headers = {

    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/125.0.0.0 Safari/537.36',

    'Referer': 'https://www.cmzj.net/',

    'Accept': 'application/json, text/plain, \*/\*',

    'Accept-Language': 'zh-CN,zh;q=0.9,en;q=0.8',

    'Connection': 'keep-alive'

}

def fetch\_expert\_list(page):

    """获取单页专家列表数据"""

    url = f"https://i.cmzj.net/expert/rankingDetail?limit=10&page={page}&lottery=4&issueNum=7&target=esm&classPay=2"

    try:

        response = requests.get(url, headers=headers, timeout=10)

        response.raise\_for\_status()

        data = response.json()

        if data['code'] == 0:

            return data['data']

        else:

            print(f"第 {page} 页数据获取失败: {data['msg']}")

            return []

    except Exception as e:

        print(f"获取第 {page} 页数据时出错: {str(e)}")

        return []

def fetch\_expert\_detail(expert\_id):

    """获取专家详情数据"""

    url = f"https://i.cmzj.net/expert/queryExpertById?expertId={expert\_id}"

    try:

        response = requests.get(url, headers=headers, timeout=8)

        response.raise\_for\_status()

        data = response.json()

        if data['code'] == 0 and 'data' in data:

            return data['data']

        else:

            print(f"专家 {expert\_id} 详情获取失败: {data.get('msg', '未知错误')}")

            return {}

    except Exception as e:

        print(f"获取专家 {expert\_id} 详情时出错: {str(e)}")

        return {}

def main():

    all\_experts = []

    # 爬取所有100页的专家基本信息

    print("开始爬取专家排行榜数据...")

    for page in tqdm(range(1, 101)):

        experts = fetch\_expert\_list(page)

        if experts:

            all\_experts.extend(experts)

        # 随机延时防止请求过快

        time.sleep(random.uniform(0.3, 0.8))

    print(f"共获取到 {len(all\_experts)} 位专家的基本信息")

    # 爬取每个专家的详情信息

    print("\n开始爬取专家详情数据...")

    full\_data = []

    # 进度条

    pbar = tqdm(total=len(all\_experts))

    for expert in all\_experts:

        # 基础信息

        expert\_data = {

            # 排行榜信息

            '专家编号': expert['expertId'],

            '榜单排名': expert['rank'],

            '专家名': expert['name'],

            '成功率': expert['scoreRate'],

            '成绩': expert['score'],

            '最新文章': expert['title'],

            'bbxId': expert['bbxId'],

            'lift': expert['lift'],

            # 详情信息 (先初始化为空)

            'gradeName': '',

            'skills': '',

            'explains': '',

            'bestRecord': '',

            'fans': '',

            'age': '',

            'articles': '',

            'dltOne': '',

            'dltTwo': '',

            'dltThree': '',

            'dltScore': ''

        }

        # 获取详情信息

        detail = fetch\_expert\_detail(expert['expertId'])

        if detail:

            # 更新详情字段

            expert\_data.update({

                'gradeName': detail.get('gradeName', ''),

                'skills': detail.get('skills', ''),

                'explains': detail.get('explains', ''),

                'bestRecord': detail.get('bestRecord', ''),

                'fans': detail.get('fans', ''),

                'age': detail.get('age', ''),

                'articles': detail.get('articles', ''),

                'dltOne': detail.get('dltOne', ''),

                'dltTwo': detail.get('dltTwo', ''),

                'dltThree': detail.get('dltThree', ''),

                'dltScore': detail.get('dltScore', '')

            })

        full\_data.append(expert\_data)

        pbar.update(1)

        # 随机延时

        time.sleep(random.uniform(0.2, 0.6))

    pbar.close()

    # 保存到CSV文件

    df = pd.DataFrame(full\_data)

    # 优化列顺序

    columns\_order = [

        'expertId', 'rank', 'name', 'gradeName', 'scoreRate', 'score',

        'bestRecord', 'dltOne', 'dltTwo', 'dltThree', 'dltScore',

        'fans', 'age', 'articles', 'skills', 'explains',

        'title', 'bbxId', 'lift'

    ]

    df = df.reindex(columns=columns\_order)

    timestamp = time.strftime("%Y%m%d\_%H%M%S")

    filename = f'大乐透收费专家20码排行榜各专家信息.csv'

    df.to\_csv(filename, index=False, encoding='utf\_8\_sig')

    print(f"\n数据爬取完成! 共获取 {len(full\_data)} 位专家数据")

    print(f"结果已保存到: {filename}")

**4.4.4运行与可视化结果**