第1题

```
from scipy.integrate import odeint  # 导入 scipy.integrate 模块
import numpy as np
import pandas as pd


def dySEIR(y, t, lamda, theta, mu):  # SEIR 模型，导数函数
    s, e, i = y
    ds_dt = -lamda*i

    de_dt = lamda*i - theta*e
    di_dt = theta*e - mu*i
    return np.array([ds_dt, de_dt, di_dt])


class LambdaSolver:
    def __init__(self) -> None:
        self.actually_injected = pd.read_csv(
            'Data\Day_Population.csv')['合计'].to_list()
        # self.actually_injected = pd.read_excel(
        #     'Data\Day_Shanghai.xlsx')['合计'].to_list()

    def loss(self, lamda, pre_value, after_value):
        """计算当前lambda造成的损失值
        lamda:当前的lambda值,
        pre_value:前一天的确诊数,
        after_value:后一天的确诊数"""
        population = 9066906  # 总人数
        theta = 0.15  # 日发病率，每天密接转阳的比率
        mu = 0.75  # 日治愈率，每天治愈的患病者人数占患病者总数的比例
        tEnd = 2  # 预测日期长度
        i0 = pre_value/population  # 患病者比例的初值
        e0 = i0*8  # 潜伏者比例的初值
        s0 = 1-i0-e0  # 易感者比例的初值
        Y0 = (s0, e0, i0)  # 微分方程组的初值
        # odeint 数值解，求解微分方程初值问题
        t = np.arange(0.0, tEnd, 1)  # (start,stop,step)
        ySEIR = odeint(dySEIR, Y0, t, args=(lamda, theta, mu))  # SEIR 模型
        pre_injected = ySEIR[:, 2] * population
        pre_injected = pre_injected[1] - pre_value
        ture_injected = after_value
        return (pre_injected-ture_injected)**2

    def found_best_lambda(self):
        """
        使用简单的for循环来优化每天的lambda
        """
        actually_injected = self.actually_injected
        lambda_result = list()
        for i in range(len(actually_injected) - 1):
            pre_value = actually_injected[i]  # 前一天的数据
            after_value = actually_injected[i+1]  # 后一天的数据
```

```
            loss_list = list()
            for j in range(100):
                loss_list.append(self.loss(j, pre_value, after_value))
            lambda_result.append(np.argmin(loss_list))
        return lambda_result
```

第2题前半部分:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import networkx as nx
from sklearn.cluster import KMeans
from sklearn.neighbors import KNeighborsClassifier
plt.style.use('fivethirtyeight')


class Solver:
    def __init__(self) -> None:
        self.community = pd.read_excel('Data/community.xlsx')
        self.intersection_node = pd.read_excel('Data\intersection_node.xlsx')
        self.intersection_route = pd.read_excel('Data\intersection_route.xlsx')

    def draw_map(self):
        intersection_route = self.intersection_route
        intersection_node = self.intersection_node
        community = self.community
        routes = intersection_route[['路线起点', '路线终点']]
        x_index = intersection_node['路口横坐标'].tolist()
        y_index = intersection_node['路口纵坐标'].tolist()
        area = list(community['所属区域'].value_counts().index)
        colors = ['red', 'yellow', 'black', 'green',
                  'orange', 'purple', 'blueviolet', 'cyan', 'tan']
        area_color_dict = dict(zip(area, colors))
        plt.figure(figsize=(10, 10))
        for i in range(routes.shape[0]):
            try:
                start = routes.iloc[i, 0]-1
                end = routes.iloc[i, 1]-1
                plt.plot([x_index[start], x_index[end]], [y_index[start],
                                                          y_index[end]],
color='b', alpha=0.6, lw=0.2, zorder=0.2)
            except:
                print(start, end)
        for i in range(community.shape[0]):
            x_index = community.iloc[i, 4]
            y_index = community.iloc[i, 5]
            area = community.iloc[i, 7]
            plt.scatter(x_index, y_index, color=area_color_dict[area], s=2)
        plt.show()

    def community_connect_intersection(self):
        community = self.community
        intersection_node = self.intersection_node
```

```
        community_intersection = list()
        for i in range(community.shape[0]):
            community_node_x_index = community.iloc[i, 4]
            community_node_y_index = community.iloc[i, 5]
            distance = list()
            for j in range(intersection_node.shape[0]):
                intersection_node_x = intersection_node.iloc[j, 1]
                intersection_node_y = intersection_node.iloc[j, 2]
                di = np.sqrt((community_node_x_index - intersection_node_x)
                             ** 2 + (community_node_y_index -
intersection_node_y)**2)
                distance.append(di)
            community_intersection.append(np.argmin(distance) + 1)
        community['最近路口节点'] = community_intersection
        community.to_csv('TempResult\社区路口连接数据.csv', index=False)

    def intersection_area(self):
        """本函数用于计算有小区连着的路口节点所属的行政区"""
        community_intersection = pd.read_csv('TempResult\社区路口连接数据.csv')
        a = community_intersection.groupby(
            ['最近路口节点', '所属区域'], as_index=False).count()
        groups = a.groupby('最近路口节点')
        intersection_node = list()
        intersection_areas = list()
        intersection_communities = list()
        for i in groups:
            node, node_data = i
            areas = node_data['所属区域'].tolist()
            counts = node_data['小区编号'].tolist()
            intersection_node.append(node)
            intersection_areas.append(areas[np.argmax(counts)])
            intersection_communities.append(np.sum(counts))
        result = pd.DataFrame({
            '路口节点': intersection_node,
            '路口节点所属行政区': intersection_areas,
            '路口节点连接社区数': intersection_communities})
        result.to_csv('TempResult\路口连接行政区数据.csv', index=False)

    def intersection_area_update(self):
        """本函数通过从上一个函数得出的结果，给所有路口节点赋予行政区的标签"""
        label_data = pd.read_csv('TempResult\路口连接行政区数据.csv')
        unlabel_data = pd.read_excel('Data\intersection_node.xlsx')
        train_data = pd.merge(left=unlabel_data,
                              right=label_data, how="right", left_on='节点编号',
right_on='路口节点')[['路口节点', '路口节点所属行政区', '路口节点连接社区数', '路口横坐标',
'路口纵坐标']]
        # train_data.to_csv('TempResult\路口训练集.csv', index=False)
        areas = list(set(train_data['路口节点所属行政区'].tolist()))
        areas_number = list(range(9))
        areas2number = dict(zip(areas, areas_number))
        number2areas = dict(zip(areas_number, areas))
        train_data['路口节点所属行政区'] = train_data['路口节点所属行政区'].apply(
            lambda x: areas2number[x])
        x_train = train_data[['路口横坐标', '路口纵坐标']]
        y_train = train_data['路口节点所属行政区']
```

```
        test_data = pd.merge(left=unlabel_data,
                             right=label_data, how="left", left_on='节点编号',
right_on='路口节点')[['节点编号', '路口横坐标', '路口纵坐标', '路口节点连接社区数']]
        test_data['路口节点连接社区数'] = test_data['路口节点连接社区数'].fillna(0)
        test_data = test_data[test_data['路口节点连接社区数'] == 0]
        knn = KNeighborsClassifier().fit(x_train, y_train)
        x_test = test_data[['路口横坐标', '路口纵坐标']]
        y_test_pre = knn.predict(x_test)
        test_data['路口节点所属行政区'] = y_test_pre
        train_data['节点编号'] = train_data['路口节点']
        train_data = train_data[['节点编号', '路口横坐标',
                                 '路口纵坐标', '路口节点连接社区数', '路口节点所属行政
区']]
        res = pd.concat([train_data, test_data])
        res = res.sort_values(by='节点编号', ascending=True)
        res['路口节点所属行政区'] = res['路口节点所属行政区'].apply(
            lambda x: number2areas[x])
        res.to_csv('TempResult\路口行政区小区数据.csv', index=False)

    def route_area(self):
        """本函数用于计算路线的行政区划,同时计算公路密度"""
        route_data = pd.read_excel('Data\intersection_route.xlsx')
        intersection_data = pd.read_csv('TempResult\路口行政区小区数据.csv')
        start_area = list()
        end_area = list()
        for i in range(route_data.shape[0]):
            start = route_data.iloc[i, 1]
            end = route_data.iloc[i, 2]
            start_area.append(intersection_data.iloc[start-1, 4])
            end_area.append(intersection_data.iloc[end-1, 4])
        route_data['起点行政区'] = start_area
        route_data['终点行政区'] = end_area
        # route_data.to_csv('TempResult\带行政区的公路线.csv', index=False)
        route_data = route_data[['路线距离(m)', '起点行政区']]
        area_route_length = route_data.groupby(
            ['起点行政区'], as_index=False)['路线距离(m)'].sum()
        area_route_length.to_csv('TempResult\各行政区公路线总长度.csv', index=False)
        area_route_length.to_excel('TempResult\各行政区公路线总长度.xlsx',
index=False)

    def draw_map_with_area(self):
        intersection_route = pd.read_csv('TempResult\带行政区的公路线.csv')
        intersection_node = self.intersection_node
        community = self.community
        routes = intersection_route[['路线起点', '路线终点', '起点行政区']]
        x_index = intersection_node['路口横坐标'].tolist()
        y_index = intersection_node['路口纵坐标'].tolist()
        area = list(community['所属区域'].value_counts().index)
        colors = ['red', 'yellow', 'black', 'green',
                  'orange', 'purple', 'blueviolet', 'cyan', 'tan']
        area_color_dict = dict(zip(area, colors))
        plt.figure(figsize=(10, 10))
        for i in range(routes.shape[0]):
            try:
                start = routes.iloc[i, 0]-1
```

```
                end = routes.iloc[i, 1]-1
                area = routes.iloc[i, 2]
                color = area_color_dict[area]
                plt.plot([x_index[start], x_index[end]], [y_index[start],
                                                          y_index[end]],
color=color, alpha=0.6, lw=0.2, zorder=0.2)
            except:
                print(start, end)
        for i in range(community.shape[0]):
            x_index = community.iloc[i, 4]
            y_index = community.iloc[i, 5]
            area = community.iloc[i, 7]
            plt.scatter(x_index, y_index, color=area_color_dict[area], s=2)
        plt.show()

    def cluster(self):
        """计算该路口所连接的路线数"""
        route_data = pd.read_excel('Data\intersection_route.xlsx')
        G = nx.DiGraph()
        for i in range(route_data.shape[0]):
            start_idx = route_data.iloc[i, 1]
            end_idx = route_data.iloc[i, 2]
            weight = route_data.iloc[i, 3]
            G.add_edge(start_idx, end_idx, weight=weight)
        a = nx.pagerank(G)
        node_list = list(a.keys())
        pr_list = list()
        for i in node_list:
            pr_list.append(a[i])
        pr_list = np.array(pr_list)
        pr_list = (pr_list-np.min(pr_list))/(np.max(pr_list)-np.min(pr_list))
        aa = pd.DataFrame({
            '节点编号': node_list,
            'pr值': pr_list
        })
        b = dict(G.degree())
        node_list = list(b.keys())
        degree_list = list()
        for i in node_list:
            degree_list.append(b[i])
        degree_list = np.array(degree_list)
        degree_list = (degree_list-np.min(degree_list)) / \
            (np.max(degree_list)-np.min(degree_list))
        bb = pd.DataFrame({
            '节点编号': node_list,
            '度值': degree_list
        })
        res = pd.merge(aa, bb, on='节点编号', how="inner")
        left = pd.read_csv('TempResult\路口行政区小区数据.csv')
        cluster_data = pd.merge(left, res, on='节点编号', how='left')
        right = pd.read_csv('TempResult\社区路口连接数据.csv')
        right = right.groupby(['最近路口节点'], as_index=False)['小区人口数
（人）'].sum()
        right['节点编号'] = right['最近路口节点']
        right = right.drop(columns=['最近路口节点'])[['节点编号', '小区人口数（人）']]
```

```
        cluster_data = pd.merge(cluster_data, right, on='节点编号', how='left')
        cluster_data['小区人口数（人）'] = cluster_data['小区人口数（人）'].fillna(0)
        temp = np.array(cluster_data['小区人口数（人）'])
        cluster_data['小区人口数（人）'] = (
            temp-np.min(temp))/(np.max(temp)-np.min(temp))
        temp = np.array(cluster_data['路口节点连接社区数'])
        cluster_data['路口节点连接社区数'] = (
            temp-np.min(temp))/(np.max(temp)-np.min(temp))
        train_data = cluster_data[['路口节点连接社区数', 'pr值', '度值', '小区人口数
（人）']]
        # k = list(range(2,10))
        # pinggu =list()
        # for ki in k:
        #     kmeans = KMeans(n_clusters=ki, random_state=42)
        #     y_pred = kmeans.fit_predict(train_data)
        #     pinggu.append(kmeans.inertia_)
        # plt.plot(k,pinggu)
        kmeans = KMeans(n_clusters=5, random_state=42)
        y_pred = kmeans.fit_predict(train_data)
        cluster_data['节点类别'] = y_pred
        label_translate = dict(
            zip([0, 1, 2, 3, 4], ['一般', '冷清', '较冷清', '繁忙', '较繁忙']))
        cluster_data['节点类别'] = cluster_data['节点类别'].apply(
            lambda x: label_translate[x])
        cluster_data.to_csv('TempResult\行政区统计数据.csv', index=False)

    def compute_area_info(self):
        data = pd.read_csv('TempResult\行政区统计数据.csv')
        groups = data.groupby(['路口节点所属行政区'])
        area_list = list()
        mean_degree = list()
        busy_percent = list()
        for i in groups:
            area, group_data = i
            area_list.append(area)
            mean_degree.append(group_data['度值'].mean())
            length = group_data.shape[0]
            busy_number = group_data[group_data['节点类别'] == '繁忙'].shape[0] + \
                group_data[group_data['节点类别'] == '较繁忙'].shape[0]
            busy_percent.append(busy_number/length)
        res = pd.DataFrame({
            '区域名称': area_list,
            '平均度': mean_degree,
            '繁忙百分比': busy_percent
        })
        data = pd.read_excel('TempResult\行政区指标.xlsx')
        result = pd.merge(res, data, on='区域名称', how='inner')
        result['指标'] = result['生活物资投放点数量'] / \
            result['隔离人口数（万人）'] * result['公路距离'] / \
            result['行政面积'] * result['平均度']*result['繁忙百分比']
        result.to_csv('Result/first_result.csv', index=False)
        result.to_excel('Result/first_result.xlsx', index=False)
```

第2题后半部分:

```python
from cnames import cnames
import pandas as pd
import gurobipy as gb
from numpy import linalg as LA
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
import matplotlib.pyplot as plt
import numpy as np
import os
plt.style.use('fivethirtyeight')
colors = list(cnames.keys())


class ProblemSolver:
    def __init__(self) -> None:
        self.XiaoQu = pd.read_excel('Data\XiaoQu.xlsx')  # 读取小区的坐标和需求数据

    def run(self):
        sortcenter_number = self.get_number_of_SortingCenter()
        # self.draw_cluster_xiaoqu_data()
        return sortcenter_number

    def get_number_of_SortingCenter(self):
        """本函数用于对小区进行聚类从而确定配送中心的数量"""
        XiaoQu = pd.read_excel('Data\XiaoQu.xlsx')
        x_index = np.array(XiaoQu['横坐标'])
        y_index = np.array(XiaoQu['纵坐标'])
        train_data = pd.DataFrame({
            '横坐标': x_index,
            '纵坐标': y_index
        })
        if os.path.exists('Temp\XiaoQu_Cluster.csv'):
            XiaoQu = pd.read_csv('Temp\XiaoQu_Cluster.csv')
        else:
            for i in range(2, 100):
                clust = AgglomerativeClustering(
                    n_clusters=i, linkage='complete')
                y_pred = clust.fit_predict(train_data)
                train_data['节点类别'] = y_pred
                train_data['人口数'] = XiaoQu['小区人口数']
                cluster_population = np.array(
                    train_data.groupby(['节点类别'])['人口数'].sum().tolist())
                decide = all(cluster_population <= 100000)
                if decide:
                    break
            XiaoQu['节点类别'] = y_pred
            XiaoQu.to_csv('Temp\XiaoQu_Cluster.csv', index=False)
        return len(list(set(XiaoQu['节点类别'])))

    def draw_cluster_xiaoqu_data(self):
        """在确定配送中心数量后为地图上色"""
        XiaoQu = pd.read_csv('Temp\XiaoQu_Cluster.csv')
        sortcenter = list(set(XiaoQu['节点类别']))
```

```
        color_dict = dict(zip(sortcenter, colors[:len(sortcenter)]))
        x_index = np.array(XiaoQu['横坐标'])
        y_index = np.array(XiaoQu['纵坐标'])
        for i in range(XiaoQu.shape[0]):
            x_i = x_index[i]
            y_i = y_index[i]
            color_i = XiaoQu.iloc[i, 6]
            color_i = color_dict[color_i]
            plt.scatter(x_i, y_i, color=color_i, s=2)
        plt.show()

    def draw_result(self):
        sort_intersection = pd.read_csv('Result\物资分发中心坐标.csv')
        intersection_route = pd.read_excel('Data\intersection_route.xlsx')
        intersection_node = pd.read_excel('Data\intersection_node.xlsx')
        community = pd.read_excel('Data\community.xlsx')
        routes = intersection_route[['路线起点', '路线终点']]
        x_index = intersection_node['路口横坐标'].tolist()
        y_index = intersection_node['路口纵坐标'].tolist()
        area = list(community['所属区域'].value_counts().index)
        colors = ['red', 'yellow', 'black', 'green',
                  'orange', 'purple', 'blueviolet', 'cyan', 'tan']
        area_color_dict = dict(zip(area, colors))
        plt.figure(figsize=(10, 10))
        for i in range(routes.shape[0]):
            try:
                start = routes.iloc[i, 0]-1
                end = routes.iloc[i, 1]-1
                plt.plot([x_index[start], x_index[end]], [y_index[start],
                                                          y_index[end]],
color='b', alpha=0.6, lw=0.2, zorder=0.2)
            except:
                print(start, end)
        for i in range(sort_intersection.shape[0]):
            xi = sort_intersection.iloc[i, 0]
            yi = sort_intersection.iloc[i, 1]
            plt.scatter(xi, yi, marker='*', color='red', s=100)
        for i in range(community.shape[0]):
            x_index = community.iloc[i, 4]
            y_index = community.iloc[i, 5]
            area = community.iloc[i, 7]
            plt.scatter(x_index, y_index, color=area_color_dict[area], s=2)

    def optimize(self):
        number_sort = 71
        sorting_index = 100 * \
            np.random.RandomState(152).random(size=(number_sort, 2))  # 分拣中心的
坐标
        """计算路径并且确认"""
        chengben = list()
        for xun in range(20):
            xiaoqu = pd.read_csv('Temp\XiaoQu_Cluster.csv')
            p = xiaoqu['需求'].tolist()
            q = 0.2
            v = 100000 * 0.4
```

```
            xiaoqu_index = np.array(xiaoqu[['横坐标', '纵坐标']])
            distance = list()
            for i in range(sorting_index.shape[0]):
                sorting_i = np.array([sorting_index[i]])
                res = LA.norm(np.array(
                    [sorting_i[:, 0] - xiaoqu_index[:, 0], sorting_i[:, 1] -
xiaoqu_index[:, 1]]), axis=0)
                distance.append(res.tolist())
            distance = np.array(distance)*0.32  # 比例尺问题
            model = gb.Model()
            w = model.addVars(number_sort, 1409, lb=0.0, ub=40000,
                              vtype=gb.GRB.CONTINUOUS, name='w')  # 添加变量
            model.setObjective(gb.quicksum(w[i, j] * distance[i, j] * q for i in
range(number_sort)
                                           for j in range(1409)),
gb.GRB.MINIMIZE)  # 添加目标函数
            model.addConstrs(w.sum('*', j) == p[j] for j in range(1409))
            model.addConstrs(w.sum(i, '*') <= v for i in range(number_sort))
            model.Params.LogToConsole = True  # 显示求解过程
            model.Params.TimeLimit = 100  # 限制求解时间为 100s
            model.optimize()
            cost = model.objVal
            """更新坐标"""
            for i in range(number_sort):
                sorting_index[i, 0] = (sum(w[i, j].X * q * xiaoqu_index[j,
0]/distance[i, j] for j in range(
                    1409))+0.0000001) / (sum(w[i, j].X * q / distance[i, j] for
j in range(1409))+0.0000001) + 0.1*np.random.RandomState(152).random()
                sorting_index[i, 1] = (sum(w[i, j].X * q * xiaoqu_index[j,
1]/distance[i, j] for j in range(
                    1409))+0.0000001) / (sum(w[i, j].X * q / distance[i, j] for
j in range(1409))+0.000001) + 0.1*np.random.RandomState(152).random()
            chengben.append(cost)
        print("总花费为", cost)
        all_renkou = pd.read_csv('Temp\XiaoQu_Cluster.csv')['小区人口数'].tolist()
        all_xuqiu = pd.read_csv('Temp\XiaoQu_Cluster.csv')['需求'].tolist()
        all_quyu = pd.read_csv('Temp\XiaoQu_Cluster.csv')['所属区域'].tolist()
        give_plan = list()
        row_index = list()
        column_index = list()
        distance_plan = list()
        Population = list()
        area = list()
        for i in range(number_sort):
            give = sum(w[i, j].X for j in range(1409))
            if give != 0:
                row_index.append(sorting_index[i, 0])
                column_index.append(sorting_index[i, 1])
                give_plan.append([w[i, j].X for j in range(1409)])
                distance_plan.append([distance[i, j] for j in range(1409)])
        give_plan = np.array(give_plan)
        distance_plan = np.array(distance_plan)
        xiaogugeshu = list()
        xuanzhibanjing = list()
        for i in range(39):
```

```
            count = 0
            banjing = list()
            renkou = list()
            quyu = list()
            for j in range(1409):
                give = give_plan[i, j]
                if give != 0:
                    count += 1
                    banjing.append(distance_plan[i, j])
                    renkou.append(all_renkou[j]*give/all_xuqiu[j])
                    quyu.append(all_quyu[j])
            xiaoqugeshu.append(count)
            xuanzhibanjing.append(max(banjing))
            Population.append(int(sum(renkou)))
            area.append(max(set(quyu), key=quyu.count))
        result = pd.DataFrame({
            '横坐标': row_index,
            '纵坐标': column_index,
            '选址半径(km)': xuanzhibanjing,
            '辖区小区数': xiaoqugeshu,
            '辖区内人口数': Population,
            '所在区域': area
        })
        result.to_csv('Result/物资分发中心坐标.csv', index=False)
        result.to_excel('Result/物资分发中心坐标.xlsx', index=False)
```

第3题:

```
import pandas as pd
xiaoqu = pd.read_excel('Data\小区.xlsx')
import matplotlib.pyplot as plt
plt.rcParams["font.sans-serif"] = ["SimHei"]  # 设置字体
plt.rcParams["axes.unicode_minus"] = False  # 正常显示负号

april_10 = pd.read_excel('Data/每天蔬菜包数据/4月10日.xlsx')
april_11 = pd.read_excel('Data/每天蔬菜包数据/4月11日.xlsx')
april_12 = pd.read_excel('Data/每天蔬菜包数据/4月12日.xlsx')
april_13 = pd.read_excel('Data/每天蔬菜包数据/4月13日.xlsx')
april_14 = pd.read_excel('Data/每天蔬菜包数据/4月14日.xlsx')
april_15 = pd.read_excel('Data/每天蔬菜包数据/4月15日.xlsx')

area_population = xiaoqu.groupby(['所属区域'], as_index=False)[[
    '小区人口数（人）', '小区户数（户）']].sum()
area_population.iloc[-1, 0] = '长春新区'
area_population['每天需求'] = 0.4*area_population['小区人口数（人）']/1000
area_population = pd.merge(area_population, april_10,
                           left_on='所属区域', right_on='行政区', how='inner')
demand = area_population['每天需求'].tolist()
buy = area_population['蔬菜包供给重量'].tolist()
give = area_population['蔬菜包发放重量'].tolist()
areas = area_population['所属区域'].tolist()
area_population['供给需求差'] = area_population['蔬菜包供给重量'] -
area_population['每天需求']
```

```
area_population['发放需求差'] = give = area_population['蔬菜包发放重量'] -
area_population['每天需求']
area_population.to_excel('Result/'+str('april_10')+'.xlsx',index=False)
# plt.figure(figsize=(14, 8))

# #设置刻度及刻度标签
# x_demand = list(range(len(areas)))
# x_buy = [i+0.2 for i in x_demand]
# x_give = [i + 0.4 for i in x_demand]

# plt.bar(x_demand, demand, width=0.2, label="当日需求量")
# plt.bar(x_buy, buy, width=0.2, label="当日采购量")
# plt.bar(x_give, give, width=0.2, label="当日分发量")
# plt.legend()
# plt.xticks(x_buy, areas, rotation=45)
# plt.show()
```

第4题:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import networkx as nx
from numpy import linalg as LA
import os
from sklearn.cluster import KMeans
from sklearn.neighbors import KNeighborsClassifier
plt.style.use('fivethirtyeight')

class Solution:
    def __init__(self) -> None:
        self.community = pd.read_excel('Data/community.xlsx')
        self.intersection_node = pd.read_excel('Data/intersection_node.xlsx')
        self.intersection_route = pd.read_excel('Data/intersection_route.xlsx')

    def run(self):
        # self.draw_map()  # 画出全市地图
        pass



    def build_graph(self):
        """本函数用于构建无向图"""
        community_route = self.community_neraest_intersection()
        intersection_route = self.intersection_route
        G = nx.Graph()
        intersection_edges = list()
        community_edges = list()
        for i in range(intersection_route.shape[0]):
            start = intersection_route.iloc[i,1]  # 路线起点
            end = intersection_route.iloc[i,2]
            weight = intersection_route.iloc[i,3]
            intersection_edges.append(('i'+str(start),'i'+str(end),weight))
        for i in range(community_route.shape[0]):
```

```
            start = community_route.iloc[i,0]  # 路线起点
            end = community_route.iloc[i,8]
            weight = community_route.iloc[i,9]
            community_edges.append(('c'+str(start),'i'+str(end),weight))
        G.add_weighted_edges_from(intersection_edges)
        G.add_weighted_edges_from(community_edges)
        return G




    def community_neraest_intersection(self):
        """本函数用于计算每个小区最近的路口"""
        if os.path.exists('TempResult/社区路口连接数据.csv'):
            return pd.read_csv('TempResult/社区路口连接数据.csv')
        else:
            intersection = self.intersection_node
            community = self.community
            intersection_index = np.array(intersection[['路口横坐标','路口纵坐标']])
            community_index = np.array(community[['小区横坐标','小区纵坐标']])
            nearest_intersection = list()
            distance = list()
            for i in range(community_index.shape[0]):
                community_index_i = np.array([community_index[i]])
                res = LA.norm(np.array([community_index_i[:, 0] -
intersection_index[:, 0],
                                        community_index_i[:, 1] -
intersection_index[:, 1]]), axis=0)
                nearest_intersection.append(np.argmin(res)+1)
                distance.append(np.min(res)* 320)
            community['最近路口节点'] = nearest_intersection
            community['路口路线距离'] = distance
            community.to_csv("TempResult/社区路口连接数据.csv",index=False)
            return community



    def draw_map(self):
        intersection_route = self.intersection_route
        intersection_node = self.intersection_node
        community = self.community
        routes = intersection_route[['路线起点', '路线终点']]
        x_index = intersection_node['路口横坐标'].tolist()
        y_index = intersection_node['路口纵坐标'].tolist()
        area = list(community['所属区域'].value_counts().index)
        colors = ['red', 'yellow', 'black', 'green',
                  'orange', 'purple', 'blueviolet', 'cyan', 'tan']
        area_color_dict = dict(zip(area, colors))
        plt.figure(figsize=(10, 10))
        for i in range(routes.shape[0]):
            try:
                start = routes.iloc[i, 0]-1
                end = routes.iloc[i, 1]-1
```

```
                plt.plot([x_index[start], x_index[end]], [y_index[start],
                                                          y_index[end]],
color='b', alpha=0.6, lw=0.2, zorder=0.2)
            except:
                print(start, end)
        for i in range(community.shape[0]):
            x_index = community.iloc[i, 4]
            y_index = community.iloc[i, 5]
            area = community.iloc[i, 7]
            plt.scatter(x_index, y_index, color=area_color_dict[area], s=2)
        plt.show()

    def draw_solution(self):
        sort_intersection = pd.read_csv('无货车结果(集散中心).csv')['节点编
号'].tolist()
        intersection_route = self.intersection_route
        intersection_node = self.intersection_node
        community = self.community
        routes = intersection_route[['路线起点', '路线终点']]
        x_index = intersection_node['路口横坐标'].tolist()
        y_index = intersection_node['路口纵坐标'].tolist()
        area = list(community['所属区域'].value_counts().index)
        colors = ['red', 'yellow', 'black', 'green',
                  'orange', 'purple', 'blueviolet', 'cyan', 'tan']
        area_color_dict = dict(zip(area, colors))
        plt.figure(figsize=(10, 10))
        for i in range(routes.shape[0]):
            try:
                start = routes.iloc[i, 0]-1
                end = routes.iloc[i, 1]-1
                plt.plot([x_index[start], x_index[end]], [y_index[start],
                                                          y_index[end]],
color='b', alpha=0.6, lw=0.2, zorder=0.2)
            except:
                print(start, end)
        for i in sort_intersection:
            xi = x_index[i-1]
            yi = y_index[i-1]
            plt.scatter(xi,yi,marker='*', color='red', s=100)
        for i in range(community.shape[0]):
            x_index = community.iloc[i, 4]
            y_index = community.iloc[i, 5]
            area = community.iloc[i, 7]
            plt.scatter(x_index, y_index, color=area_color_dict[area], s=2)
        cangku = pd.read_csv('无货车结果(仓库).csv')
        for i in range(cangku.shape[0]):
            x_index = cangku.iloc[i, 2]
            y_index = cangku.iloc[i, 3]
            plt.scatter(x_index, y_index, color='blue',marker='*', s=200)
        plt.show()

    def draw_solution_car(self):
        sort_intersection = pd.read_csv('有货车结果(集散中心).csv')['节点编
号'].tolist()
        intersection_route = self.intersection_route
```

```
        intersection_node = self.intersection_node
        community = self.community
        routes = intersection_route[['路线起点', '路线终点']]
        x_index = intersection_node['路口横坐标'].tolist()
        y_index = intersection_node['路口纵坐标'].tolist()
        area = list(community['所属区域'].value_counts().index)
        colors = ['red', 'yellow', 'black', 'green',
                  'orange', 'purple', 'blueviolet', 'cyan', 'tan']
        area_color_dict = dict(zip(area, colors))
        plt.figure(figsize=(10, 10))
        for i in range(routes.shape[0]):
            try:
                start = routes.iloc[i, 0]-1
                end = routes.iloc[i, 1]-1
                plt.plot([x_index[start], x_index[end]], [y_index[start],
                                                          y_index[end]],
color='b', alpha=0.6, lw=0.2, zorder=0.2)
            except:
                print(start, end)
        for i in sort_intersection:
            xi = x_index[i-1]
            yi = y_index[i-1]
            plt.scatter(xi,yi,marker='*', color='red', s=100)
        for i in range(community.shape[0]):
            x_index = community.iloc[i, 4]
            y_index = community.iloc[i, 5]
            area = community.iloc[i, 7]
            plt.scatter(x_index, y_index, color=area_color_dict[area], s=2)
        cangku = pd.read_csv('有货车结果(仓库).csv')
        for i in range(cangku.shape[0]):
            x_index = cangku.iloc[i, 2]
            y_index = cangku.iloc[i, 3]
            plt.scatter(x_index, y_index, color='blue',marker='*', s=200)
        plt.show()
```

有货车:

```
from Solution import *
solve = Solution()
G = solve.build_graph()
import pandas as pd
data = pd.read_csv('TempResult\社区路口连接数据.csv')
intersection = list(set(data['最近路口节点']))
community = pd.read_excel('Data\community.xlsx')
demand = (community['小区人口数（人）']*0.4).tolist()
distance = list()
for i in range(len(intersection)):
    start = intersection[i]
    shortest_distance_dict = nx.shortest_path_length(G,
source='i'+str(start),weight='weight')
    distance.append([shortest_distance_dict['c'+ str(i)] for i in
range(1,1410)])
distance = np.array(distance)
import gurobipy as gb
intersection_length=len(intersection)
```

```
community_length=1409
model=gb.Model()
Q =
model.addVars(intersection_length,community_length,lb=0.0,ub=40000,vtype=gb.GRB.
CONTINUOUS, name='Q')  # 添加变量
R = model.addVars(intersection_length,vtype=gb.GRB.BINARY,name='R')  # 添加变量
B = model.addVars(intersection_length,vtype=gb.GRB.INTEGER,name='B',lb=0,ub=4)
L = model.addVars(intersection_length,vtype=gb.GRB.INTEGER,name='L',lb=0,ub=10)
print("变量添加完毕")
model.addConstrs(
    gb.quicksum(Q[k,l]*R[k] for k in range(intersection_length)) == demand[l]
for l in range(community_length)
)
print('约束1添加完毕')
model.addConstrs(
    gb.quicksum(Q[k,l] for l in range(community_length)) <= 40000 for k in
range(intersection_length)
)
print("约束2添加完毕")
model.addConstr(
    gb.quicksum(R[k] for k in range(intersection_length)) <= 71
)
print("约束3添加完毕")
model.addConstrs(
    B[k]*10000 + L[k]*4000 - gb.quicksum(Q[k,l] for l in
range(community_length)) >= 0 for k in range(intersection_length)
)
print("约束4添加完毕")

model.setObjective(
    gb.quicksum(
        Q[k,l] * distance[k,l] * R[k] for l in range(community_length) for k in
range(intersection_length)
    ) + 2*gb.quicksum(
        R[k]*B[k] for k in range(intersection_length)
        ) + gb.quicksum(R[k]*L[k] for k in range(intersection_length)),
gb.GRB.MINIMIZE
)  # 添加目标函数
print("目标函数设置完毕")
model.Params.TimeLimit=3600
model.optimize()
intersection_node = list()
intersection_x = list()
intersection_y = list()
for i in range(len(intersection)):
    if round(R[i].X) != 0:
        intersection_node.append(intersection[i])

res = pd.DataFrame({'intersection_node':intersection_node})
res.to_csv('有货车.csv',index=False)
old_data = pd.read_excel('Data\intersection_node.xlsx')
intersection_node = pd.read_csv('有货车.csv')['intersection_node']
intersection_node =
pd.merge(intersection_node,old_data,left_on='intersection_node',right_on='节点编
号',
```

```
                                how='left')[['节点编号','路口横坐标','路口纵坐标']]
areas = pd.read_csv('TempResult\社区路口连接数据.csv')
intersection_node = pd.merge(intersection_node,areas,left_on='节点编
号',right_on='最近路口节点',how='left')
intersection_node = intersection_node[['节点编号','路口横坐标','路口纵坐标','所属区
域']]
intersection_node.drop_duplicates(subset=['节点编号'], keep='first', inplace=True)
intersection_node.reset_index(drop=True, inplace=True)
intersection_node.to_csv('有货车结果(集散中心).csv',index=False)
intersection_node.to_excel('有货车结果(集散中心).xlsx',index=False)
groups = intersection_node.groupby(['所属区域'])
areas=list()
center_x = list()
center_y = list()
for i in groups:
    ar,group_data = i
    areas.append(ar)
    center_x.append(group_data['路口横坐标'].mean())
    center_y.append(group_data['路口纵坐标'].mean())
cangku = pd.DataFrame({
    '节点编号':[i+1 for i in range(len(areas))],
    '区域':areas,
    '仓库中心横坐标':center_x,
    '仓库中心纵坐标':center_y
})
cangku.to_csv('有货车结果(仓库).csv',index=False)
cangku.to_excel('有货车结果(仓库).xlsx',index=False)
from Solution import *
solve = Solution()
solve.draw_solution_car()
```

没有货车:

```
from Solution import *
solve = Solution()
G = solve.build_graph()
import pandas as pd
data = pd.read_csv('TempResult\社区路口连接数据.csv')
intersection = list(set(data['最近路口节点']))
community = pd.read_excel('Data\community.xlsx')
demand = (community['小区人口数（人）']*0.4).tolist()
distance = list()
for i in range(len(intersection)):
    start = intersection[i]
    shortest_distance_dict = nx.shortest_path_length(G,
source='i'+str(start),weight='weight')
    distance.append([shortest_distance_dict['c'+ str(i)] for i in
range(1,1410)])
distance = np.array(distance)
import gurobipy as gb
intersection_length=len(intersection)
community_length=1409
model=gb.Model()
```

```
Q =
model.addVars(intersection_length,community_length,lb=0.0,ub=40000,vtype=gb.GRB.
CONTINUOUS, name='Q')  # 添加变量
R = model.addVars(intersection_length,vtype=gb.GRB.BINARY,name='R')
print("变量添加完毕")
model.addConstrs(
    gb.quicksum(Q[k,l]*R[k] for k in range(intersection_length)) == demand[l]
for l in range(community_length)
)
print('约束1添加完毕')
model.addConstrs(
    gb.quicksum(Q[k,l] for l in range(community_length)) <= 40000 for k in
range(intersection_length)
)
print("约束2添加完毕")
model.addConstr(
    gb.quicksum(R[k] for k in range(intersection_length)) <= 71
)

model.setObjective(
    gb.quicksum(
        Q[k,l] * distance[k,l] * R[k] for l in range(community_length) for k in
range(intersection_length)
    ), gb.GRB.MINIMIZE
)  # 添加目标函数
print("目标函数设置完毕")
model.Params.TimeLimit=3600
model.optimize()
intersection_node = list()
intersection_x = list()
intersection_y = list()
for i in range(len(intersection)):
    if round(R[i].X) != 0:
        intersection_node.append(intersection[i])
res = pd.DataFrame({'intersection_node':intersection_node})
res.to_csv('无货车.csv',index=False)
import pandas as pd
old_data = pd.read_excel('Data\intersection_node.xlsx')
intersection_node = pd.read_csv('无货车.csv')['intersection_node']
intersection_node =
pd.merge(intersection_node,old_data,left_on='intersection_node',right_on='节点编
号',
                              how='inner')[['节点编号','路口横坐标','路口纵坐标']]
areas = pd.read_csv('TempResult\社区路口连接数据.csv')
intersection_node = pd.merge(intersection_node,areas,left_on='节点编
号',right_on='最近路口节点',how='inner')
intersection_node = intersection_node[['节点编号','路口横坐标','路口纵坐标','所属区
域']]
intersection_node.drop_duplicates(subset=['节点编号'], keep='first', inplace=True)
intersection_node.reset_index(drop=True, inplace=True)
intersection_node.to_csv('无货车结果(集散中心).csv',index=False)
intersection_node.to_excel('无货车结果(集散中心).xlsx',index=False)
groups = intersection_node.groupby(['所属区域'])
areas=list()
center_x = list()
```

```
center_y = list()
for i in groups:
    ar,group_data = i
    areas.append(ar)
    center_x.append(group_data['路口横坐标'].mean())
    center_y.append(group_data['路口纵坐标'].mean())
cangku = pd.DataFrame({
    '节点编号':[i+1 for i in range(len(areas))],
    '区域':areas,
    '仓库中心横坐标':center_x,
    '仓库中心纵坐标':center_y
})
cangku.to_csv('无货车结果(仓库).csv',index=False)
cangku.to_excel('无货车结果(仓库).xlsx',index=False)
from Solution import *
solve = Solution()
solve.draw_solution()
loss =
[17281,16928,66,12,10.5,10.3,9.43,8.83,8.58,8.21,7.72,5.82,5.04,3.06,1.14]
plt.plot(loss)
```