```python
# -*- coding: utf-8 -*-
"""
Temperature Analysis for AOS Research
Pranav Addepalli

The full project folder can be found at https://github.com/pranavaddepalli/AOSResearch2019
"""
#%% SETUP and LOAD RAW DATA

import numpy as np
import os
import matplotlib.pyplot as plt
import pandas as pd
import scipy.stats as stats


np.set_printoptions(precision=3, suppress=True)

base_dir = os.getcwd()
data_dir = base_dir + "/Data/"

raw_10_percent = np.genfromtxt(data_dir + '10pLines', delimiter=',')
print("10% infill data has {} columns and {} rows.".format(np.size(raw_10_percent, axis=1), np.size(raw_10_percent, axis=0)))
raw_10_percent = raw_10_percent

raw_20_percent = np.genfromtxt(data_dir + '20pLines (1)', delimiter=',')[:, :8]
print("20% infill data has {} columns and {} rows.".format(np.size(raw_20_percent, axis=1), np.size(raw_20_percent, axis=0)))
raw_20_percent = raw_20_percent

raw_30_percent = np.genfromtxt(data_dir + '30pLines', delimiter=',')
print("30% infill data has {} columns and {} rows.".format(np.size(raw_30_percent, axis=1), np.size(raw_30_percent, axis=0)))
raw_30_percent = raw_30_percent

#%% PROCESS RAW DATA

def point(raw, row, col):
    n = col
    temperature = raw[row, col]
    if col == 0:
         x = 70
         y = -70
    elif col == 1:
        x = 55
        y = 56
    elif col == 2:
        x = 32.4
        y = -41.8
    elif col ==3:
        x = 24
        y = 15
    elif col == 4:
        x = 0
        y = -12
    elif col == 5:
        x = -22.5
        y = 33
    elif col == 6:
        x = -45
        y = 48
    else:
        x = -72.5
        y = 56
    return n, x, y, temperature
```

```


system = np.zeros(shape = (3, len(raw_30_percent) + len(raw_20_percent) + len(raw_10_percent), 8), dtype='O')

hm_x = [[] for _ in range(3)]
hm_y = [[] for _ in range(3)]
hm_t = [[] for _ in range(3)]
hm_temp = [[] for _ in range(3)]

for i in range(0,30, 10):
    infill = globals()['raw_' + str(i + 10) + '_percent']
    print("Creating points for {}% infill...".format(i + 10), end="", flush=True)
    for col in range(0, np.size(infill, axis=1)):
        for row in range(0, np.size(infill, axis=0)):
            n, x, y, temperature = point(infill, row, col)
            hm_x[int(i / 10)].append(x)
            hm_y[int(i / 10)].append(y)
            hm_t[int(i / 10)].append(row)
            hm_temp[int(i / 10)].append(temperature)
            system[int(i / 10), row, n] = (x, y, temperature)
    print("Done!")


#%% CENTER CALCULATIONS

x_graph_list = [[] for i in range(3)]
y_graph_list = [[] for i in range(3)]
avgTemp_graph_list = [[] for i in range(3)]
equilibrium = (5.175, 10.525, 0)
def center(points):
    global equilibrium
    global x_graph_list
    global y_graph_list
    weighted_mean_temp = sum((((((p[0])**2) +
                                ((p[1])**2))**0.5)
                                * p[2]) for p in points)
    tmp = (sum(((((point[0])**2) + ((point[1])**2))**0.5) for point in points))
    weighted_mean_temp = weighted_mean_temp / tmp

    x = sum([point[0] * point[2] for point in points]) / sum([point[2] for point in points])
    y = sum([point[1] * point[2] for point in points]) / sum([point[2] for point in points])
    return (x, y, weighted_mean_temp)



centers = [[] for _ in range(3)]
for infill in range(0, 3):
    print("Calculating centers for {}% infill...".format((infill + 1)*10), end="", flush=True)
    for time in system[infill]:
        if type(time[0]) is tuple:
            x, y, temperature_mean = center(time)
            x_graph_list[infill].append(x)
            y_graph_list[infill].append(y)
            avgTemp_graph_list[infill].append(temperature_mean)
            centers[infill].append((x, y, temperature_mean))
        else: break
    print("Done!")

#%% VECTOR CALCULATIONS

gradients = [[] for _ in range(3)]
for infill in range(0, 3):
    print("Calculating gradients for {}% infill...".format((infill + 1)*10), end="",
    flush=True)
    for c in centers[infill]:
        dx = c[0] - equilibrium[0]
        dy = c[1] - equilibrium[1]
        dt = c[2] - equilibrium[2]
        gradT = np.sqrt( ((dt / dx)**2) + ((dt / dy)**2) )
        direction = np.degrees(np.arctan(dy / dx))
        gradients[infill].append((dx, dy, dt, gradT, direction))
    print("Done!")




#%% VISUALIZATION
for GRAPHING_INFILL in range(0, 3):

    #CENTERS

    fig = (plt.figure())
    plt.subplot(111)
    ax = plt.gca()
    ax.scatter(x_graph_list[GRAPHING_INFILL], y_graph_list[GRAPHING_INFILL], s=1)
    ax.plot(equilibrium[0], equilibrium[1], "or")
    plt.ylabel("Y position (mm)")
    plt.xlabel("X position (mm)")
    plt.xlim(0,12.5)
    plt.ylim(0,15)
    plt.title("Temperature Centers for {}% infill".format(format((GRAPHING_INFILL + 1)*10)))

    #RAW DATA

    raw_df = pd.DataFrame(data= globals()['raw_' + str(10*GRAPHING_INFILL + 10) + '_percent'])
    raw_df.columns = ['Thermistor 1', 'Thermistor 2', 'Thermistor 3', 'Thermistor 4', 'Thermistor 5', 'Thermistor 6', 'Thermistor 7', 'Thermistor 8']
    ax0 = raw_df.plot(title="Temperature over Time for {}% Infill".format(format((GRAPHING_INFILL + 1)*10)), xlim=(0, 10000))
    ax0.set_xlabel("Time (s)")
    ax0.set_ylabel("Temperature (C)")

plt.show()





#%% STATISTICS

#STANDARD DEVIATION OF CENTERS

print("Standard Deviation of Temperature Centers:")

x_std_ten = np.std(x_graph_list[0])
y_std_ten = np.std(y_graph_list[0])
print("\nTen percent: \n")
print("STD in X: {} \nSTD in Y: {}".format(x_std_ten,y_std_ten))

x_std_twenty = np.std(x_graph_list[1])
y_std_twenty = np.std(y_graph_list[1])
print("\nTwenty percent:\n")
print("STD in X: {} \nSTD in Y: {}".format(x_std_twenty,y_std_twenty))

x_std_thirty = np.std(x_graph_list[2])
y_std_thirty = np.std(y_graph_list[2])
print("\nThirty percent: \n")
print("STD in X: {} \nSTD in Y: {}".format(x_std_thirty,y_std_thirty))
```

```python


#GRADIENTS
ten_gradients = [value[3] for value in gradients[0] ]
twenty_gradients = [value[3] for value in gradients[1] ]
thirty_gradients = [value[3] for value in gradients[2] ]
anova_statistic, anova_pvalue = stats.f_oneway(ten_gradients, twenty_gradients, thirty_gradients)
print("\nANOVA test:\n-------------------- \nStatistic: {} \np-value: {}\n".format(anova_statistic, anova_pvalue))
print("Mean of 10%: {} \nMean of 20%: {} \nMean of 30%: {}".format(np.mean(ten_gradients), np.mean(twenty_gradients), np.mean(thirty_gradients)))

kw_statistic, kw_pvalue = stats.kruskal(ten_gradients, twenty_gradients, thirty_gradients)
print("\nKruskal-Wallis test:\n-------------------- \nStatistic: {} \np-value: {}\n".format(kw_statistic, kw_pvalue))

statistic_10_20, pvalue_10_20 = stats.ttest_ind(ten_gradients, twenty_gradients, equal_var=False)
statistic_20_30, pvalue_20_30 = stats.ttest_ind(twenty_gradients, thirty_gradients, equal_var=False)
print("Two-Sample T Test for Independence with unequal variances:\n--------------------")
print("10% to 20%:\n--------------------\nStatistic: {} \np-value: {}".format(statistic_10_20, pvalue_10_20))
print("20% to 30%:\n--------------------\nStatistic: {} \np-value: {}".format((statistic_20_30), pvalue_20_30 ))
```