TRIOS Software and JSON Export: Basic Interaction with the Dataframe in Python

Keywords: TRIOS, JSON, Python

TB106

ABSTRACT

This technical brief discusses how to interact with dataframe information in Python® within the context of a thermal analysis data file. It covers the contents of the data file, how to extract specific details, and introduces basic plotting of that information. This note is not a comprehensive guide to Python and its libraries; readers are encouraged to refer to the official documentation for more details.

Note: The exported JSON file and the final Python script (in Jupyter® code format) used in this technical brief can be downloaded here.

INTRODUCTION

The TA Instruments™ TRIOS™ Software instrument control and data analysis package offers numerous features for both routine and in-depth analysis of scientific data. However, you may sometimes need to generate uniform plot formats, overlay data from multiple sources, or apply advanced functions not available within TRIOS.

TRIOS Software has always supported data export in various formats, including plain text (ASCII), Comma Separated Values (CSV), and Excel™ workbooks. TRIOS Software version 5.8 introduced the ability to export data to a JavaScript™ Object Notation (JSON) file. The import of this JSON file into Python was discussed in a previous technical brief TB105. This note will explore how to interact with the resulting dataframe.

THE PYTHON SCRIPT AND WORKING WITH THE DATAFRAME

In a previous technical brief, importing the JSON file and creating the dataframe were discussed. When executing the Python script, the interaction with the dataframe occurs in the background; you typically see only the final results. However, it is often useful for you to examine the structure or contents of the dataframe. The Python code block below demonstrates how to import the JSON file into the dataframe, this can be downloaded as Script 1.

```
# Import the required libraries
from tadatakit.classes import Experiment
import pandas as pd

# Load the JSON experiment data
experiment = Experiment.from_json("files/PLA Bar.json")

# Get the dataframe of processed data with
# indexing for each step from the JSON file
step_name, pla_df = experiment.get_dataframes_by_step("processed")
```

This script creates five dataframes (indexed 0 to 4), each representing a step in the measurement:

- pla_df[0]: Equilibrate 0.00 °C step
- pla_df[1]: Ramp 10.00 °C/min to 200.00 °C step
- pla_df[2]: Isothermal 2.0 min step
- pla_df[3]: Ramp 10.00 °C/min to 0.00 °C step
- pla_df[4]: Ramp 10.00 °C/min to 200.00 °C step

To view the dataframe for the first heating cycle you can use the Python code line:

```
print(pla_df[1])
```

This displays a snapshot of the complete dataframe, showing the signal labels and the first and last five rows of data as shown in Figure 1.

	Cell Purge / mL/min	Heater Temp / °C	Reference Junction Temperature / °C	Heat Capacity Polystyrene / J/(g·°C)	Heat Capacity (Normalized) / J/(g·°C)	1/Temperature / 1/K	Delta T / µV	Time / min	Heat Capacity / J/°C	Step time / min	Delta Tzero / µV	Results Step Id	Heat Capacity Sapphire / J/(g·°C)	Temperature / °C	Tzero Temperature / °C	Procedure Step Id	Flange Temperature / °C	Power Delivered / W	Heat Flow / mW	Heat Flow (Normalized) / W/g
0	50.012260	1.735767	45.980679	1.113678	0.000000	0.003662	0.861053	0.000000	0.000000	0.000000	-0.760607	ad3d51c9-248e-48b5-b1be-57552b9ccbb1	0.717366	-0.044276	-0.086054	0002060a-0000-0000-0000-000000000000	-86.010788	30.799707	0.189013	0.023538
1	50.018291	1.732518	45.980682	1.113677	0.000000	0.003662	0.850797	0.001667	0.000000	0.001667	-0.742704	ad3d51c9-248e-48b5-b1be-57552b9ccbb1	0.717366	-0.044528	-0.085863	0002060a-0000-0000-0000-000000000000	-86.011696	30.973228	0.186555	0.023232
2	50.018291	1.732642	45.980686	1.113679	0.000000	0.003662	0.833462	0.003333	0.000000	0.003333	-0.729604	ad3d51c9-248e-48b5-b1be-57552b9ccbb1	0.717367	-0.044006	-0.085863	0002060a-0000-0000-0000-000000000000	-86.014404	29.794399	0.182383	0.022713
3	50.020446	1.733408	45.980694	1.113677	0.000000	0.003662	0.829738	0.005000	0.000000	0.005000	-0.710298	ad3d51c9-248e-48b5-b1be-57552b9ccbb1	0.717366	-0.044390	-0.085470	0002060a-0000-0000-0000-000000000000	-86.016068	31.840631	0.180871	0.022524
4	50.020446	1.734927	45.980690	1.113677	0.000000	0.003662	0.817896	0.006667	0.000000	0.006667	-0.695212	ad3d51c9-248e-48b5-b1be-57552b9ccbb1	0.717366	-0.044411	-0.085275	0002060a-0000-0000-0000-000000000000	-86.019325	31.456158	0.178959	0.022286
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
11999	50.009113	197.170654	45.977222	NaN	2.206398	0.002120	-8.333546	19.998334	0.017717	19.998334	52.596995	ad3d51c9-248e-48b5-b1be-57552b9ccbb1	1.017040	198.442307	199.278732	0002060a-0000-0000-0000-000000000000	-63.637012	107.066666	-2.496391	-0.310883
12000	50.009113	197.187729	45.977222	NaN	2.211683	0.002120	-8.333538	20.000000	0.017760	20.000000	52.596941	ad3d51c9-248e-48b5-b1be-57552b9ccbb1	1.017054	198.458755	199.295242	0002060a-0000-0000-0000-000000000000	-63.632233	104.592041	-2.496419	-0.310887
12001	50.009113	197.202835	45.977226	NaN	2.217309	0.002120	-8.333543	20.001666	0.017805	20.001666	52.596879	ad3d51c9-248e-48b5-b1be-57552b9ccbb1	1.017069	198.475937	199.312424	0002060a-0000-0000-0000-000000000000	-63.630039	104.748550	-2.496438	-0.310889
12002	50.011292	197.219940	45.977230	NaN	2.223186	0.002120	-8.333544	20.003333	0.017852	20.003333	52.596821	ad3d51c9-248e-48b5-b1be-57552b9ccbb1	1.017084	198.493073	199.329514	0002060a-0000-0000-0000-000000000000	-63.627640	103.412392	-2.496436	-0.310889
12003	50.011292	197.237305	45.977226	NaN	2.229371	0.002120	-8.333562	20.005001	0.017902	20.005001	52.596737	ad3d51c9-248e-48b5-b1be-57552b9ccbb1	1.017098	198.509567	199.345871	0002060a-0000-0000-0000-000000000000	-63.622841	106.750694	-2.496419	-0.310887

12004 rows × 20 columns

Figure 1. Snapshot of the complete dataframe

It is also possible to extract a single row of data using an index locator. The code below shows an example of this which will extract the fifth row of data on first heat dataframe.

```
1. print(pla_df[1].iloc[4])
```

The result of executing this code is shown in Figure 2.

```
Cell Purge / mL/min                                    50.020446
Heater Temp / °C                                        1.734927
Reference Junction Temperature / °C                    45.98069
Heat Capacity Polystyrene / J/(g.°C)                    1.113677
Heat Capacity (Normalized) / J/(g.°C)                      0.0
1/temperature / 1/K                                     0.003662
Delta T / µV                                            0.817896
Time / min                                              0.006667
Heat Capacity / J/°C                                       0.0
Step time / min                                         0.006667
Delta Tzero / µV                                       -0.695212
Results Step Id                  ad3d51c9-248e-48b5-b1be-57552b9cccb1
Heat Capacity Sapphire / J/(g.°C)                       0.717366
Temperature / °C                                       -0.044411
Tzero Temperature / °C                                 -0.085275
Procedure Step Id                0002060a-0000-0000-0000-000000000000
Flange Temperature / °C                                -86.019325
Power Delivered / W                                    31.456158
Heat Flow / mW                                          0.178959
Heat Flow (Normalized) / W/g                            0.022286
Name: 4, dtype: object
```

Figure 2. Data content of the fifth row of the first heat dataframe

It is also possible to extract a single column of data using the code below.

```
1. print(pla_df[1]["Heat Flow (Normalized) / W/g"])
```

As with the display of the full dataframe, only a snapshot of the first and last five data points will be shown. This is shown in Figure 3.

```
0        0.023538
1        0.023232
2        0.022713
3        0.022524
4        0.022286
           ...
11999   -0.310883
12000   -0.310887
12001   -0.310889
12002   -0.310889
12003   -0.310887
Name: Heat Flow (Normalized) / W/g, Length: 12004, dtype: float64
```

Figure 3. Snapshot display of the selected data column (Heat Flow (Normalized) / W/g in this case)

Finally, it is also possible extract a single value from the dataframe by combining both the column name and the row index as in the following code line.

```
1. print(pla_df[1]["Heat Flow (Normalized) /
   W/g"].iloc[4])
```

This example shows where the heat flow normalized value at the fifth row of the first heat dataframe is displayed with the resultant output in Figure 4.

```
0.0222863025665283
```

Figure 4. Resultant display from the Python code for the single value extraction

DATA VISUALIZATION

The Matplotlib library is a comprehensive visualization library for Python. Refer to the Matplotlib documentation [1] for more details. A basic plot can be created by first importing the library. Then the signals from the dataframe to be plotted can be defined, and labels added to the axes. The code block for this is shown below:

```
 1. import matplotlib.pyplot as plt
 2.
 3. # Set the size of the figure
 4. plt.figure(figsize=(10, 6))
 5.
 6. # Define the x and y signals
 7. plt.plot(pla_df[1]['Temperature / °C'],
    pla_df[1]['Heat Flow (Normalized) / W/g'])
 8.
 9. # Label the x-axis
10. plt.xlabel("Temperature (°C)")
11.
12. # Label the y-axis
13. plt.ylabel("Heat Flow (Normalized) (W/g)")
14.
15. # Display the plot
16. plt.show()
```

The plot generated is shown in Figure 5.

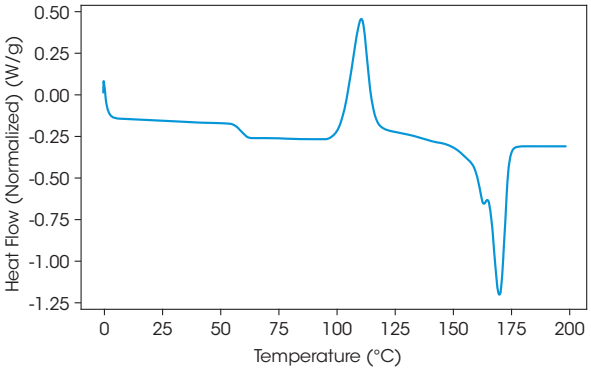

Figure 5. Basic plot of the first heat dataframe showing the heat flow against temperature

CONCLUSIONS

This technical brief highlights the basic steps required to interact with the dataframe generated from the JSON file. The basic steps to producing a plot are also highlighted.

REFERENCES

1. Matplotlib — Visualization with Python

ACKNOWLEDGMENTS

This technical brief was written by Philip Davies, Principal Applications Scientist, TA Instruments.

For more information or to request a product quote, please visit www.tainstruments.com to locate your local sales office information.