About project2

Your task: predict the throughput

- Input data: "***training_data/trace*.json***". It consists of multiple records with the following fields
  - ***timestamp***: the timestamp of the current record
  - ***speed***: the normalized moving speed of the device (cell phone)
  - ***signal_strength***: LTE signal strength (higher means stronger)
  - ***neighbour_cell_signal_strength***: the signal strength of the neighbor cell
  - ***extra_event***: the connectivity related event, can be "disconnected", "handover-command" or "connection-request"
  - ***carrier_bandwidth***: the available bandwidth measured inside the carrier, will not be affected by signal strength
  - ***throughput***: the ground truth throughput, you need to predict this

- During the testing, you will see all the fields EXCEPT the ground truth ***throughput***. You need to generate your own prediction.

Introduction to timeline processing tool

Installation and running

Install the dependencies

- The tool needs the following dependencies
  - Java runtime
  - Scala 2.13.8
  - Python3 with the following packages
    - notebook, pandas, numpy, matplotlib

- We provide a helper script "***install_dependency.sh***" to install the dependencies
  - Use "***bash install_dependency.sh***" on your macbook/linux server will automatically install the dependencies for you
  - To check the installation, try opening a new shell and type "***java***" and "***scala***"

Run the example notebook

- We provide an example jupyter notebook called "***Example.ipynb***", you can run it with jupyter notebook

- If you are not familiar with jupyter notebook
  - Navigate to the project folder, and type "python3 -m notebook"
  - It will try to automatically bring up your browser and show the notebook in the webpage

Overview of the tool

Overview

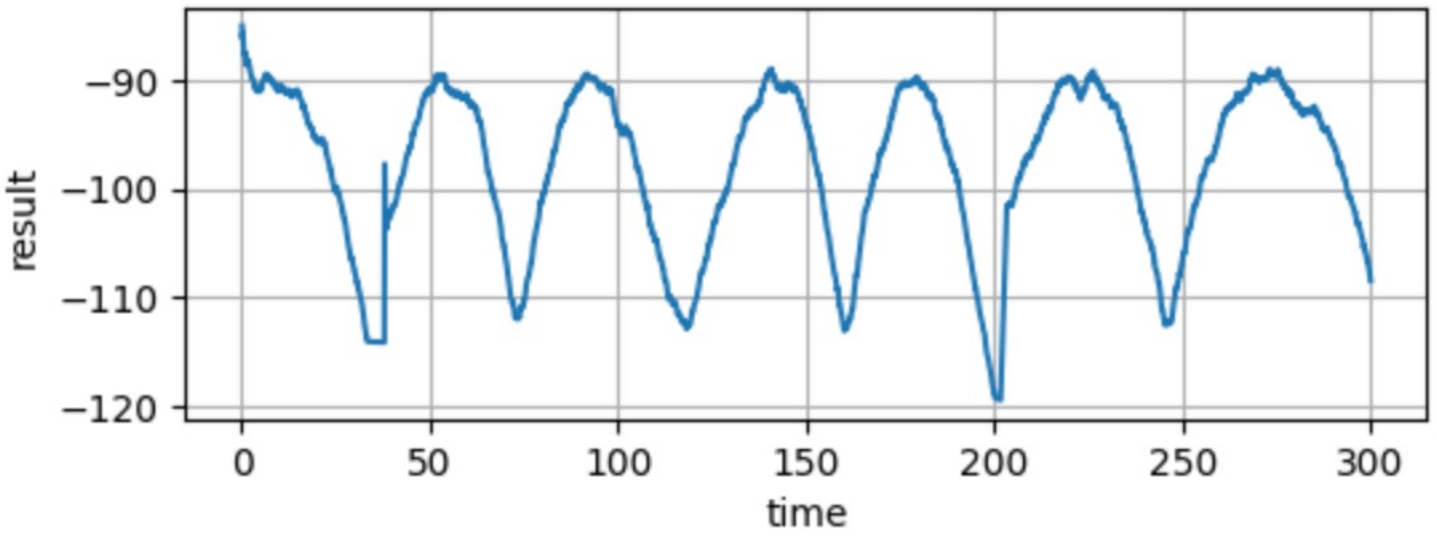
Read

Process

Visualize

```
datafile = Datafile("training_data/trace1.json")
input_data = datafile.read()
```

```
signal_strength = input_data.get("signal_strength", value_type="number")
avg_signal = signal_strength.averageWithin(3)
```




```
plot_timeline(avg_signal)
```

Open a data file for reading

The Datafile class, defined in data_parser.py

The location of the file

```
datafile = Datafile("training_data/trace1.json")

input_data = datafile.read()
```

Read the datafile to an object of class “Timeline” (also defined in data_parser.py)

Get a field as a timeline (timeseries) from the datafile

Interface: "***get()***" method of a timeline object

Value type, can be one of "number", "boolean" or "string" (case sensitive)

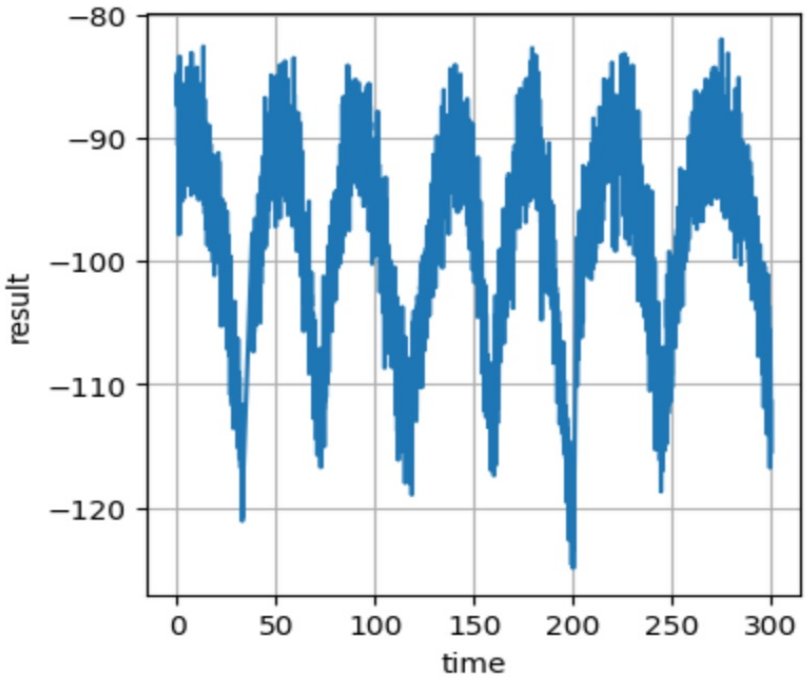
The field name you want to get

```
signal_strength = input_data.get("signal_strength", value_type="number")
```

Returns a new Timeline object

Visualize a single timeline

- Function ***"plot_timeline(timeline: Timeline)"***




```
datafile = Datafile("training_data/trace1.json")
input_data = datafile.read()

signal_strength = input_data.get("signal_strength", value_type="number")

plot_timeline(signal_strength)
```

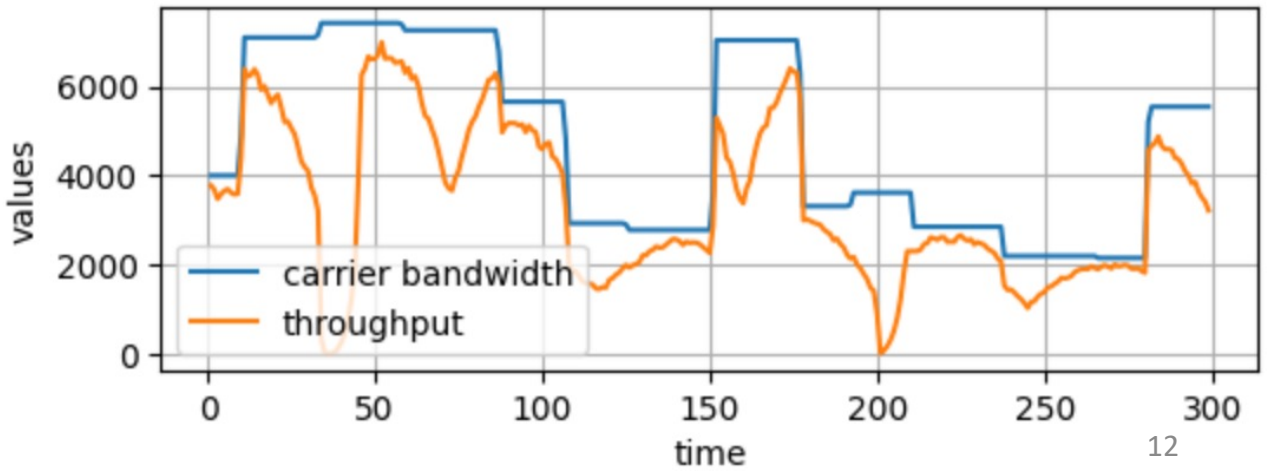
Visualize multiple timelines

- Function *"plot_multiple_timeline(timlines: List[Timeline], labels: list[String])"*

```
datafile = Datafile("training_data/trace1.json")
input_data = datafile.read()

bandwidth = input_data.get("carrier_bandwidth", value_type="number")
throughput = input_data.get("throughput", value_type="number")

plot_multiple_timeline([bandwidth, throughput],
                       labels = ["carrier bandwidth", "throughput"])
```



Compute the RMSE between 2 timelines

- Function: "***calculate_rmse(groundtruth: Timeline, prediction: Timeline)***"

```
datafile = Datafile("training_data/trace1.json")
input_data = datafile.read()

bandwidth = input_data.get("carrier_bandwidth", value_type="number")
throughput = input_data.get("throughput", value_type="number")

rmse = calculate_rmse(bandwidth, throughput)
print("rmse is: ", rmse)
```

Output:

rmse is: 1970.3353746429768

Convert a timeline object to pandas dataframe

```
datafile = Datafile("training_data/trace1.json")
input_data = datafile.read()

signal_strength = input_data.get("signal_strength", value_type="number")

dataframe = timeline_to_dataframe(signal_strength)
display(dataframe)
```

	start	end	value
0	0.1	0.1	-85.061883
1	0.2	0.2	-87.400492
2	0.3	0.3	-84.717133
3	0.4	0.4	-87.453078
4	0.5	0.5	-90.596972
...	...	...	...
2937	299.5	299.5	-108.021720
2938	299.6	299.6	-109.215920
2939	299.7	299.7	-113.241399
2940	299.8	299.8	-115.607637
2941	299.9	299.9	-111.395648

2942 rows × 3 columns

Timeline operation references

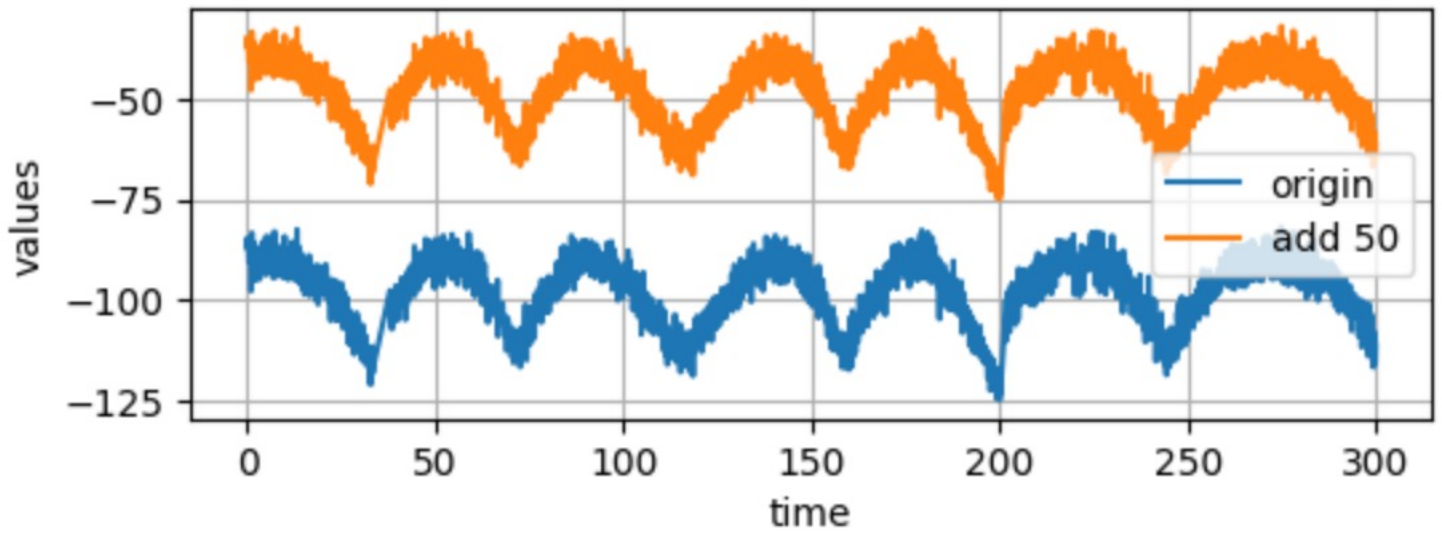
Timeline operation usually takes 0 or 1 parameters, and returns a new timeline objects

Timeline operation: addConst

- Function: ***Timeline.addConst(value: float) -> Timeline***
  - Add a const to the timeline

```
signal_strength = input_data.get("signal_strength", value_type="number")
new_timeline = signal_strength.addConst(50)

plot_multiple_timeline([signal_strength, new_timeline],
                       labels=["origin", "add 50"])
```

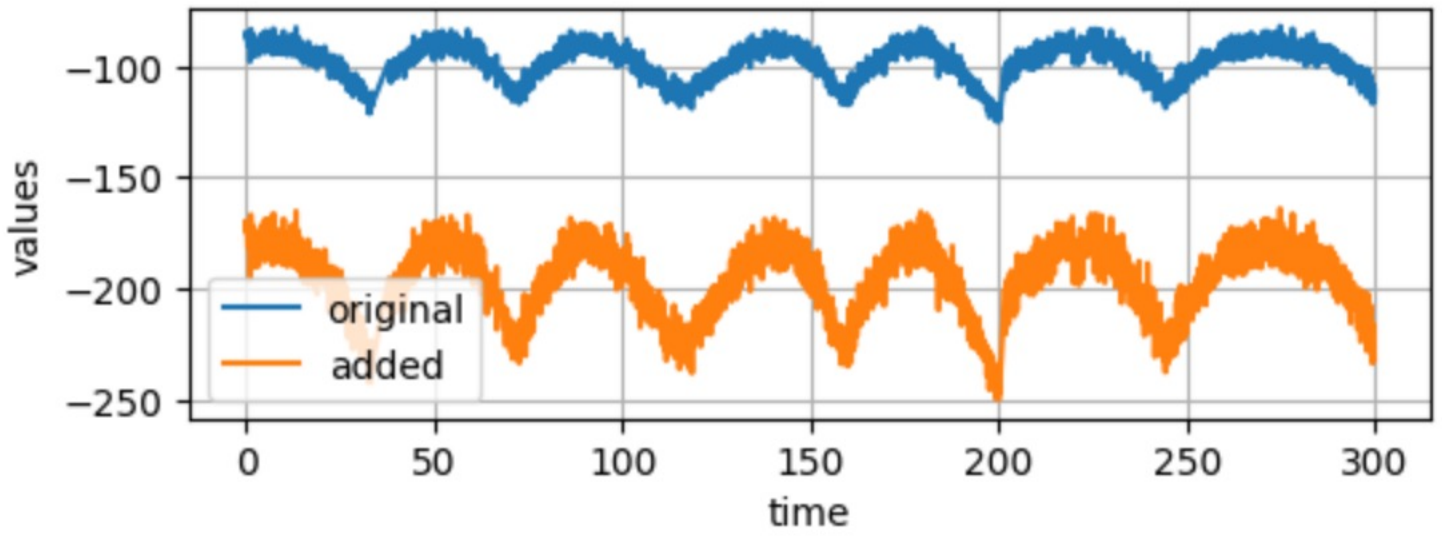


Timeline operation: add

- Function: ***Timeline.add(other: Timeline) -> Timeline***
  - Add to timelines together

```
signal_strength = input_data.get("signal_strength", value_type="number")
added_signal_strength = signal_strength.add(signal_strength)

plot_multiple_timeline([signal_strength, added_signal_strength],
                       labels=["original", "added"])
```

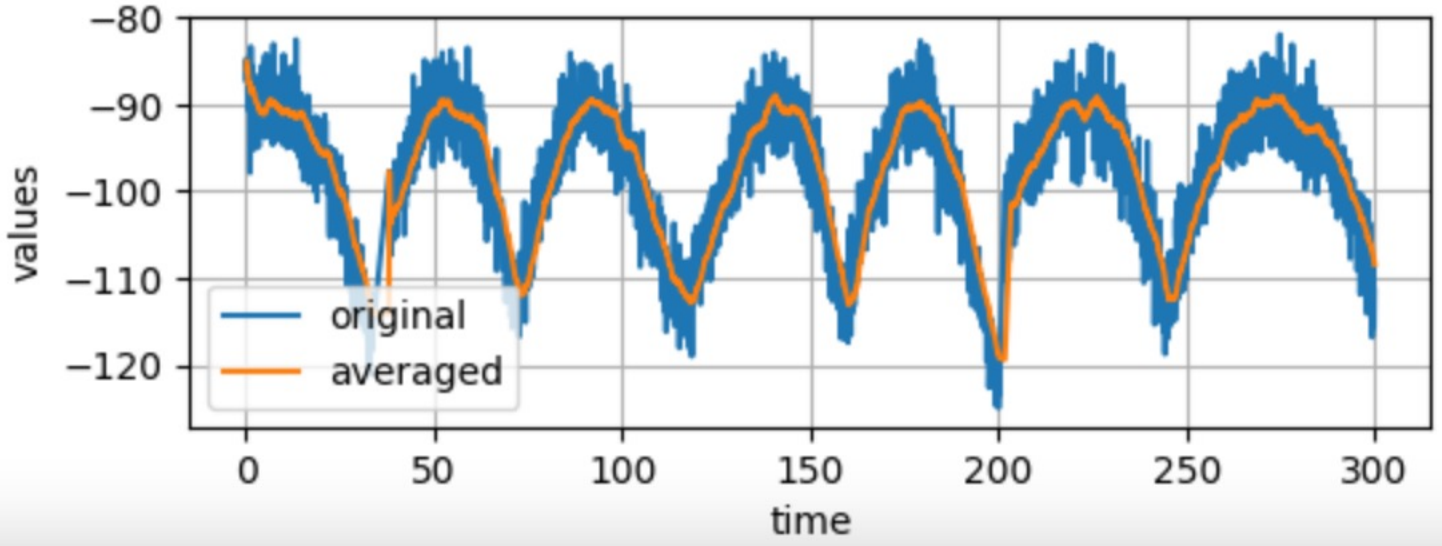


Timeline operation: averageWithin

- Function: ***Timeline.averageWithin(window_len: float) -> Timeline***
  - Compute the windowed average from $[t - window\_len, t]$

```
signal_strength = input_data.get("signal_strength", value_type="number")
average_signal_strength = signal_strength.averageWithin(window_len=3)

plot_multiple_timeline([signal_strength, average_signal_strength],
                       labels=["original", "averaged"])
```



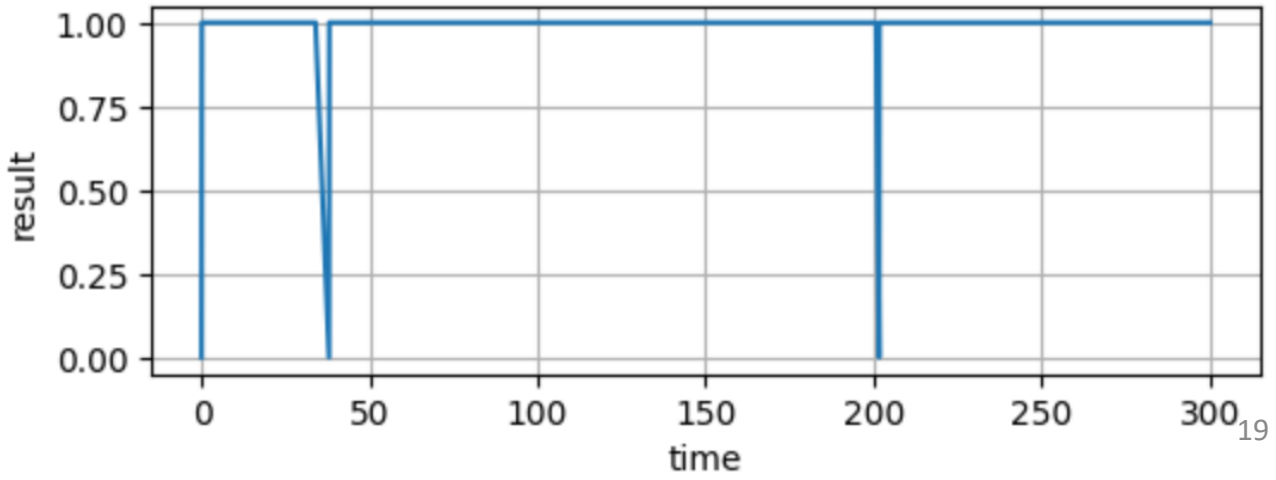
Timeline operation: hasDataWithin

- Function: ***Timeline.hasDataWithin(window_len: float) -> Timeline***
  - The value at time t of the result timeline will be "True" if there is any datapoint in the old timeline within window $[t - window\_len, t]$

```
datafile = Datafile("training_data/trace1.json")
input_data = datafile.read()

signal_strength = input_data.get("signal_strength", value_type="number")
has_signal_strength = signal_strength.hasDataWithin(window_len=0.5)

plot_timeline(has_signal_strength)
```

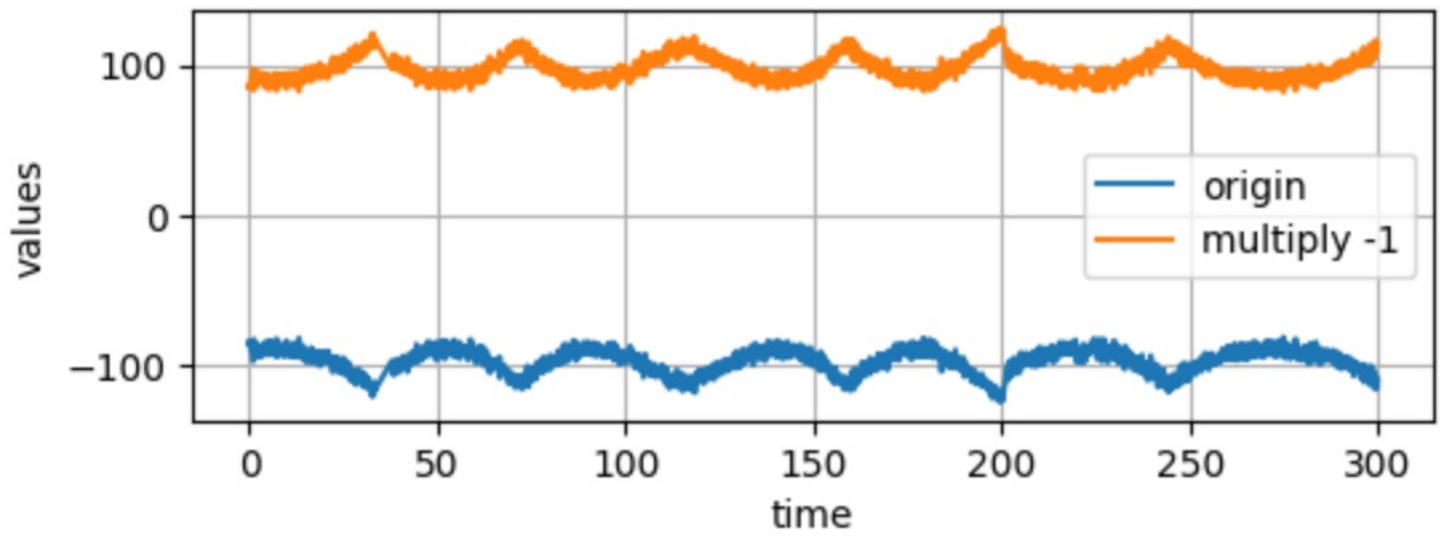


Timeline operation: multiplyConst

- Function: ***Timeline.multiplyConst(value: float) -> Timeline***
  - Multiply a constant number to the timeline

```
signal_strength = input_data.get("signal_strength", value_type="number")
new_timeline = signal_strength.multiplyConst(-1)

plot_multiple_timeline([signal_strength, new_timeline],
                       labels=["origin", "multiply -1"])
```

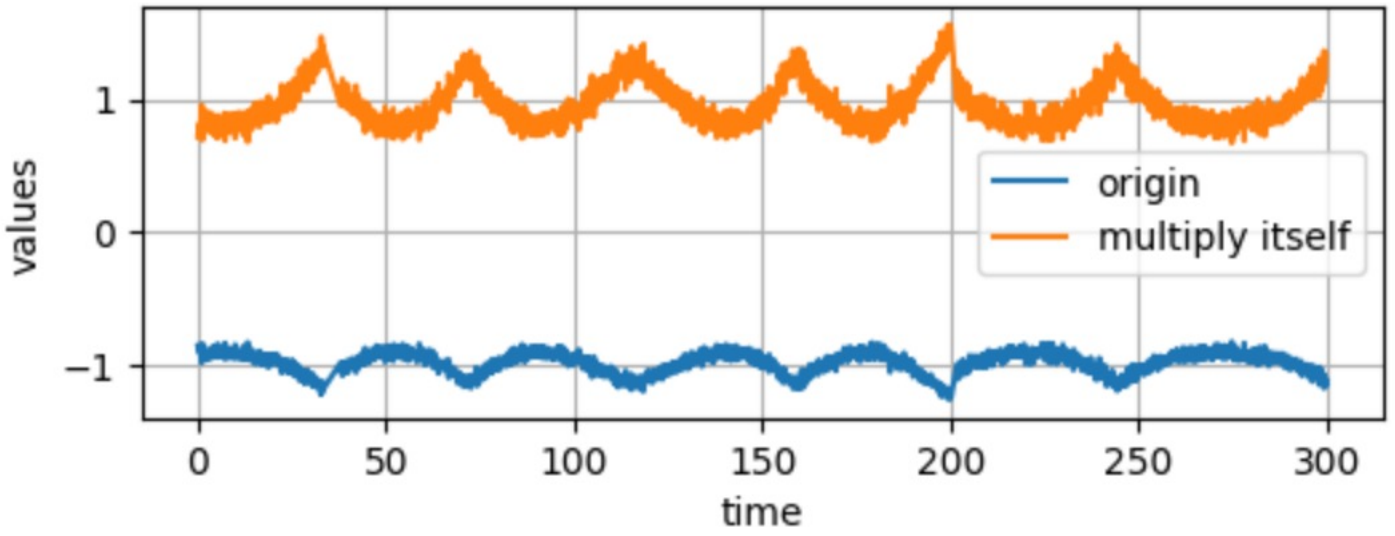


Timeline operation: multiply

- Function: ***Timeline.multiply(other: Timeline) -> Timeline***
  - Multiply with another timeline

```
signal_strength = input_data.get("signal_strength", value_type="number").multiplyConst(0.01)
new_timeline = signal_strength.multiply(signal_strength)

plot_multiple_timeline([signal_strength, new_timeline],
                       labels=["origin", "multiply itself"])
```



Timeline operation: divide

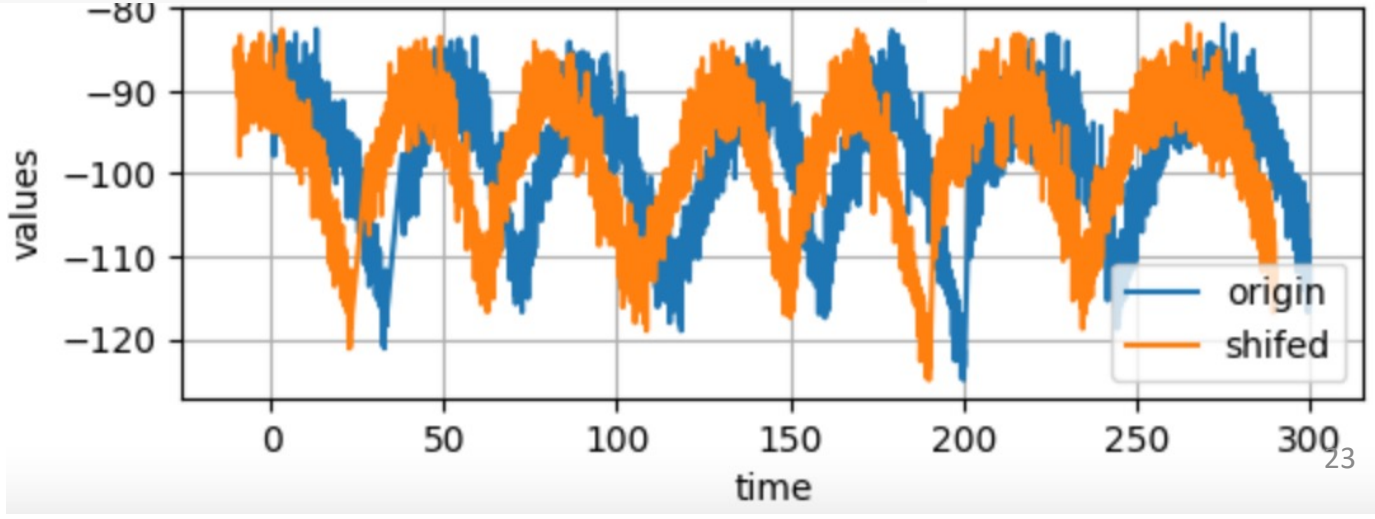
- Function: ***Timeline.divide(other: Timeline) -> Timeline***
  - Divide by another timeline
  - Usage is similar to “multiply” and “add”

Timeline operation: shift

- Function: ***Timeline.shift(left: float) -> Timeline***
  - "Shift" the timeline to the left by xxx seconds. If parameter "left" is negative, then it will shift the timeline to the right

```
signal_strength = input_data.get("signal_strength", value_type="number")
shifted_signal = signal_strength.shift(left=10)

plot_multiple_timeline([signal_strength, shifted_signal],
                       labels=["origin", "shifed"])
```



Timeline operation: latestEventToState

- Function: ***Timeline.latestEventToState() -> Timeline***
  - Convert a “event” time to a step-function like timeline

```
throughput = input_data.get("throughput", value_type="number")
step_function = throughput.latestEventToState()
timeline_to_dataframe(throughput)
timeline_to_dataframe(step_function)
```

	start	end	value
0	1.0	1.0	3753.617067
1	2.0	2.0	3604.849228
2	3.0	3.0	3617.854959
3	4.0	4.0	3696.341917
4	5.0	5.0	3636.002662

LatestEventToState()

	start	end	value
0	1.0	2.0	3753.617067
1	2.0	3.0	3604.849228
2	3.0	4.0	3617.854959
3	4.0	5.0	3696.341917
4	5.0	6.0	3636.002662