User Documentation

T. Christenson (tc), Yifeng Cui (yc), Brian Gunnarson (bg), Jacob Rammer (jr), Sam Peters (sp),

-- 2-10-2021 --

-- v1.0 --

Table of Contents

Prerequisites

- Python 3.8

How to create a virtual environment (optional) and install our system

- Go to the project directory
- Optional: If you would like to create a virtual environment
    - Run the command: “python -m venv proj-1-env”
        - Or “python3 -m venv proj-1-env" for Unix systems
    - You will need to run these commands depending on your operating system
        - On Unix/macOS run: “source proj-1-env/bin/activate”
        - On Windows run: “proj-1-env\Scripts\activate.bat”
- Run: "pip install -r requirements.txt”

How to accomplish real world tasks

This section includes some use cases for our system. We believe these will be the most used functions of our system. You can also read our documentation (programmer documentation)

for more use cases and how exactly to use each method. More use cases is included in demo.py.

- Creating a new tree for a new TimeSeries object
    - Important: We have imported our transformation tree module as “tsas”
    - First, we must create a tree object. We can do this by using this line: tree = tsas.tree.TransformationTree(tsas.operatorkeys.operator_input_keys, tsas.operatorkeys.operator_output_keys)
    - **Important: This operation must be performed on every tree so that data can be supplied to the TimeSeries.**
    - We then need to add a TimeSeries object into the root of the tree. This can be accomplished by the following line of code: read = tree.add_operator(tsas.preprocessing.TimeSeries.read_from_file, [filename], t.root, tag="read")
        - For every action performed on the tree, we must add an operator, that is why we are calling tree.add_operator. Next, we add the operator to that node, using tsas.“filename”.TimeSeries.”method”. In this example, from preprocessing, we are using the read_from_file method. Then arguments are provided to these methods in a form a list. Since preprocessing.py -> read_from_file only take one argument, we only include the filename in the list. We then state where to store the TimeSeries object, which we obviously want to store it at the root of the tree, we specifiy this as tree.root. Finally, we give the node a descriptive tag such as read.
        - Note: if a method takes no argument, you must include an empty list when calling add_operator.
        - Make note of the variable name you created here as you will need it later.

- Assigning missing date and time data to a time series
    - In the previous use case, we read in a file that only had data values such as temperature, missing the dates and time associated with those temperature values. We can accomplish this by adding another operator to the tree that references the previous node (in this example, the previous node is read)
    - assign_time = tree.add_operator(tsas.preprocessing.TimeSeries.assign_time, ["01/01/2000 12:30", 24], read)
        - The first argument is similar to the previous use case: tsas.preprocessing.TimeSeries.method_name.
        - The second argument is the list of arguments, included in the assign_time documentation, we know that we need to provide a string with the date

in mm/dd/yyyy hh:mm format and a whole number for the number of hours to increment. So, these two arguments are provided in the list as comma separated values.
    - As previously stated, we need to know the name of the previous node, which is this case is read, so we include that variable as the final argument.

- Extracting data between date intervals
    - Data scientist may also want to extract part of a time series and manipulate that data over the specified time interval. Our system also provides the user this functionality. We first need to add an operator to the tree extract = t.add_operator(tsas.preprocessing.TimeSeries.clip, ["01/01/2020", "01/01/2021"], ts2db)
    - In the programmer documentation, we describe how to use the clip method in our system. For sake of time, we will omit those details here. Per the programmer documentation, we know that the two arguments used by clip are date strings in the for of mm/dd/yyyy. We supply our starting date in the argument list and the ending data. This method returns a separate time series with the extracted data that can later be manipulated.
    - There are no compatibility checks with this method, the only requirement is that the TimeSeries stored at the node has data that includes dates and times.
- Write Tree object data to file
    - After data has been manipulated and there are several leaves to a tree, the data scientist may want to save the current state of the tree. To save the state of the tree, you can simply call the save method.
        - Tsas.tree.save(name_of_tree, file_string)
- Create a database using TS2DB
    - In order to predict a model, we need to create a database with certain time series data. Similar to read and assign time, we have to add an operator to the tree.
    - ts2db = t.add_operator(tsas.preprocessing.TimeSeries.ts2db, [None, 0.8, 0.1, 0.1, 2, 1, None], denoise, tag="db", save_result=False)
        - In the programmer documentation, we go into more detail about each argument in the argument list represents, so we are going to leave that out in this section.
        - We once again include tsas.preprocessing.TimeSeries.method as the first argument.
        - Then in our argument list, we supply our desired arguments.
        - Next, we include the previous leaf in the tree, which is denoise in our case. Generally, you first denoise time series data before you run ts2db,

but for the sake of this document, we have already run it without describing the exact steps. More info can be found in demo.py
        - Since this is a major operation, we give this node a tag of “db”

- Plotting data
    - After data has been manipulated to the data scientist’s satisfaction, they can now plot the data very easily with our software. We will be using plot for this example, but the same method applies to histogram and box plot.
    - Note: extract, model, fit, predict, and mse methods must first be ran to avoid compatibility issues. Our software will raise a compatibility error is these conditions aren’t met.
    - Plotting is a simple as plot = t.add_operator(tsas.visualization.plot, [], mse)
        - This is because plot does not take any arguments.
- Creating a model
    - The data scientist may also want to model the data of a time series or predict how the time series data is going to react. We have several modeling methods in our system that are included in our programmer's documentation, for this particular use case, we are going to focus on predict.
    - predict = t.add_operator(tsas.modelingAndForecasting.predict, [], fit)
    - As before, we must add an operator to the tree and assign it to a variable. Since the predict method doesn’t require any methods, our argument list is left empty.
    - Prerequisites for the predict method are model and fit, failure to run these methods on the time series will result in a compatibility error, and the program will terminate.