<Time Series Analysis Support for Data Scientists>
Software Requirements Specification
by Keyboard Warriors

Table of Contents

1. SRS Revision History

CJ = Cameron Jordal
JK = JT Kashuba
NK = Noah Kruss
NT = Nick Titzler
RV = River Veek

Date	Author	Description
1/25/2021	CJ + JK	Initial creation of document using template provided by Prof A. Hornof
1/25/21	CJ + JK	Worked on Current System or Situation, Justification, Operational Features, and User Classes
1/26/21	CJ + JK	Worked on Use Cases
2/6/21	NK	Worked on Use Cases
2/8/21	NK	Worked on Use Cases
2/8/21	JK	Editing Use Cases
2/8/21	JK + NK	Finalized Majority of Use Cases
2/9/21	NK	Worked on sections 3.1 and 3.2
2/9/21	CJ + JK	Finalized Sections 2.2, 2.3, 3.3
2/9/21	RV	Finalized Sections 3.4, 3.5
2/9/21	NK	Finalized Sections 3.1 and 3.2

2. The Concept of Operations (ConOps)

2.1. Current System or Situation

A time series is a chronological collection of data. Data scientists, for example, can use time series to predict (or forecast) future data points in fields such as meteorology or the stock market. Currently, Machine Learning (henceforth referred to as ML) models are frequently used for predicting the short-term future by processing time series.

Some tools at the disposal of those in the pursuit of analyzing and/or visualizing time series are Pandas, matplotlib, and scikit-learn, all of which are open-source Python modules. Pandas is specifically designed for data science applications and can take data from a multitude of sources, such as spreadsheets, csv files, etc. Matplotlib can then be used to visualize the data in a meaningful way by generating representative graphs. Scikit-Learn is used for the generation and prediction of various machine learning models.

The motivation for the proposed system is to provide a comprehensive time series manipulation and visualization library for machine learning. The system will use data analysis and a variety of functions to modify or visualize data points that reference specific information collected over time.

2.2. Justification for a New System

There are currently a multitude of ways to manipulate time series data; however, the purpose of this course is to work as a cohesive team to address a contrived problem in a professional manner to produce a robust, functional application.

"Time series forecasting is an important task in the fields of Data Science and Machine Learning. Time series originate from almost any area of human activity, and forecasting these time series provides a valuable insight into things that may happen in the future. Predictions of future outcomes and the underlying process to derive predictions can inform and influence the decision-making process. For instance, when generating renewable energy with solar panels, the two most important variables that impact the amount of energy produced by a solar cell are solar irradiance and ambient temperature. Each of these variables can be measured over time to create a time series. Solar power plant managers and engineers are interested in estimating how much energy the plant will produce in the near future."

-TIME SERIES ANALYSIS SUPPORT FOR DATA SCIENTISTS by Juan Flores

2.3. Operational Features of the Proposed System

The system we are creating will allow for a comprehensive time series analysis and prediction platform from which a data scientist could load a set of time series data, process and analyze the data as required, generate a model to predict future data points, and save these models and predicted / altered time series.

This process should allow for most common use cases for processing and analyzing data, such as the identification and removal of outliers, generating models that forecast future data points, and visualization of these forecasted models. All of these functionalities are accessible via a Time Series Tree object, which has its operations explained in great detail in Section 2.6 (Use Cases).

2.4. User Classes

Data Scientists who want to use a collection of data in a variety of different situations.This could include individuals who are motivated to make predictions in a particular field. Some examples would be stock market day-traders, businesspeople looking to maximize profits for the next quarter based on their past sales, or a businessperson wanting to produce a forecast to meet the necessary supply for future demand.

2.5. Modes of Operation

There will be no specialization of functions based on different users.

2.6. Operational Scenarios (Also Known as "Use Cases")

Use Case 1: Initialize a Tree to manipulate/visualize a time series

Brief Description: Demonstrates how to create a TS_Tree object using the "tree.py" library in order to process and forecast time series data.

Actors: A Data Scientist

Pre-Condition:

1. Users have downloaded and placed the Time Series folder (library) into the same directory as the file they would like to use the functions in.
2. At the top of the file the User is writing, they have included the following import statement: from tree import *

Steps to Complete the Task:

For creating a TS_Tree

1. Execute the following command: tree_var_name = TS_Tree()

For loading a previously saved TS_Tree

1. Execute the following command: tree = load_tree("fileName.txt")

Post-Condition: Users will have a TS_Tree object which they can pass time series data into through an execute_path or execute_tree command to process the data. (More on execute_path & execute_tree in Use Cases 5 & 6 respectively)

Use Case 2: Adding a node to an existing TS_Tree

Brief Description: Demonstrates how to add an operation as a leaf node to a specified path within an already existing TS_Tree

Actors: A Data Scientist

Pre-Condition:

1. Users have downloaded and placed the Time Series folder (library) into the same directory as the file they would like to use the functions in.
2. Users have already created (or loaded a pre-existing) TS_Tree (see Use Case 1). *Note: A TS_Tree can have a node added via add_node either before or after other calls to add_node, replace_node, add_subtree, execute_path, or execute_tree.*
3. Users have determined that they would like to perform an additional operation from the functions provided in the preprocessing, visualization, or modeling modules.

Steps to Complete the Task:

1. Users identify an operation they want to add to their TS_Tree.
2. Users determine the node_index of the **parent** for the node being added. This can easily be done in a few simple steps.
   a. Display the tree (in terminal) by running tree.print_tree()
   b. Identify the number next to the **parent node**
3. Users check ******CHANGE******** SECTION 666 to identify the proper parameters for the specific function they are adding.
4. Users add the following command to their code
   a. Generic example:
      tree.add_node(operation: str, node_index, input_parameters)
      Some clarifications:
      1. operation: str could be (as one example) “impute_missing_data”
      2. Input_parameters ****CHANGE SECTION 666******
      Concrete examples:
      tree.add_node(“impute_missing_data”, node_index)

tree.add_node("assign_time", node_index, data_start = 1.0, increment = 10.0)

Post-Condition: Users will have a TS_Tree object which they can pass time series data into through an execute_path or execute_tree command to process the data.

Note: More on execute_path & execute_tree in Use Cases 5 & 6 respectively

Use Case 3: Replacing a node in an existing TS_Tree

Brief Description: Demonstrates how to replace an existing node in a TS_Tree with a different operation

Actors: A Data Scientist

Pre-Condition:

1. Users have downloaded and placed the Time Series folder (library) into the same directory as the file they would like to use the functions in.
2. Users have already created (or loaded a pre-existing) TS_Tree (see Use Case 1). *Note: A TS_Tree can have a subtree added via add_subtree either before or after other calls to add_node, replace_node, add_subtree, execute_path, or execute_tree.*
3. Users have determined that they would like to replace a node in a TS_Tree, meaning they would like to perform a different operation than the current operation in that node.

Steps to Complete the Task:

1. Users identify which operation they want to replace within their TS_Tree, as well as which operation they want to replace it with.
2. Users determine the node_index for the node whose operation is being replaced. This can easily be done in a few simple steps.
   a. Display the tree (in terminal) by running tree.print_tree()
   b. Identify the number next to the node
3. Users check ******CHANGE******** SECTION 666 to identify the proper parameters for the specific function they are changing to.
4. Users add the following command to their code
   a. Generic example:

      tree.replace_node(operation: str, node_index, input_parameters)

      Some clarifications:

      1. operation: str could be (as one example) "impute_missing_data"

2. Input_parameters ****CHANGE SECTION 666******
Concrete examples:

**************EXPLAIN WHY THESE 2 HAVE DIFFERENT # OF ARGUMENTS*****

tree.replace_node("impute_missing_data", node_index)

tree.replace_node("assign_time", node_index, data_start = 1.0, increment = 10.0)

Post-Condition: Users will have a TS_Tree object which they can pass time series data into through an execute_path or execute_tree command to process the data.

Note: More on execute_path & execute_tree in Use Cases 5 & 6 respectively

Use Case 4: Adding a subtree to an existing TS_Tree

Brief Description: This use case describes how users add a sub-tree from an existing Time Series Tree to another Time Series Tree they had created.

Actors: A Data Scientist

Pre-Condition:

1. Users have downloaded and placed the Time Series folder (library) into the same directory as the file they would like to use the functions in.
2. Users have already created (or loaded a pre-existing) two TS_Tree objects (see Use Case 1). We will refer to the TS_Tree we'll be adding the subtree *to* as TREE_1 and the TS_Tree we'll be using as our subtree as TREE_2.

Steps to Complete the Task:

1. Users identify the index of the operation node in TREE_2 that they wish to become the root of their sub-tree (subtree_root_index). This can easily be done in a few simple steps.
   a. Display the tree (in terminal) by running tree.print_tree()
   b. Identify the number next to the node
2. Users create a new TS_Tree object in their code which is a copy of the subtree they want. Users add the following line to their code
   a. subtree = copy_subtree(TREE_2, subtree_root_index)
3. Users identify the index (node_index) of the operation node in TREE_1 that they wish to add the newly created subtree to.
4. Users add the subtree to TREE_1 by adding the following command to their code

a. add_subtree(TREE_1, node_index, subtree)

Post-Condition: Users will have a TS_Tree object which they can pass time series data into through an execute_path or execute_tree command to process the data.

Note: More on execute_path & execute_tree in Use Cases 5 & 6 respectively

Use Case 5: Executing a path in an existing TS_Tree

Brief Description: Returns the output from executing the functions along the path from the root to a specified node in a TS_Tree on a given time series.

Actors: A Data Scientist

Pre-Condition:

1. Users have downloaded and placed the Time Series folder (library) into the same directory as the file they would like to use the functions in.
2. Users have already created (or loaded a pre-existing) TS_Tree (see Use Case 1). *Note: A TS_Tree can be executed either before or after prior calls to add_node, replace_node, add_subtree, execute_path, or execute_tree.*
3. Users have selected a time series (TS) they would like to manipulate and/or visualize via executing a path in an existing TS_Tree.

Steps to Complete the Task:

1. Users determine the node_index for the node at the end of the path they want to execute (the path will always start at the root). This can easily be done in a few simple steps.
   a. Display the tree (in terminal) by running tree.print_tree()
   b. Identify the number next to the node
2. Users execute a path by adding the following command to their code
   a. tree.execute_path(TS_data, node_index)
   Note: TS_data can be a direct reference to a file containing a time series *OR* a variable containing the time series read in from a file if the User calls replace_node on the root with another operation, examples as follows:
   tree.execute_path("fileName.txt", node_index)
   OR
   my_data = read_from_file("fileName.txt")

tree.execute_path(my_data, node_index)

Post-Condition: The user has an output stored in the result variable which can be one of the following options

a. A modified time series
b. A design matrix
c. A forecasting model
d. Error amount of a forecast
e. Float amount of how normal the time series data is

Use Case 6: Executing a tree (from an existing TS_Tree)

Brief Description: Calls execute_path on every leaf node.

Actors: A Data Scientist

Pre-Condition:

1. Users have downloaded and placed the Time Series folder (library) into the same directory as the file they would like to use the functions in.
2. Users have already created (or loaded a pre-existing) TS_Tree (see Use Case 1). *Note: A TS_Tree can be executed either before or after prior calls to add_node, replace_node, add_subtree, execute_path, or execute_tree.*
3. Users have selected a time series (TS) they would like to manipulate and/or visualize via executing all paths to leaf nodes in an existing TS_Tree.

Steps to Complete the Task:

1. Users execute a tree by adding the following command to their code
    a. tree.execute_tree(TS_data)
       Note: TS_data can be a direct reference to a file containing a time series *OR* a variable containing the time series read in from a file if the User calls replace_node on the root with another operation, examples as follows:

       tree.execute_tree("fileName.txt")

       OR

       my_data = read_from_file("fileName.txt")
       tree.execute_tree(my_data)

Post-Condition: Users have a dictionary variable containing the output from executing the functions along the various paths from the root to leaf nodes in a TS_Tree on a given time series. The dictionary contains *key:value* pairs where the *keys* are the names of the leaf nodes (where the name is the operation referenced in the node plus it's index in the tree. EX: denoise-1, impute_outliers-2, etc.) and the *values* are the output of executing the path from the root to that leaf node given the particular time series.

Use Case 7: Saving a TS_Tree

Brief Description: Demonstrates how to save a TS_Tree that Users want to keep for future use.

Actors: A Data Scientist

Pre-Condition:

1. Users have downloaded and placed the Time Series folder (library) into the same directory as the file they would like to use the functions in.
2. Users have already created (or loaded a pre-existing) TS_Tree (see Use Case 1). *Note: A TS_Tree can be executed either before or after prior calls to add_node, replace_node, add_subtree, execute_path, or execute_tree.*

Steps to Complete the Task:

1. Users determine a name (save_file_name) for the file in which they would like to save their tree. *Note: save_file_name **can** be a file that already exists. In this case the save command will overwrite the data stored in that file. If save_file_name does not already exist then the save tree command will create a new file with this name.*
2. Users save the tree by adding the following command to their code
    a. save_tree(tree, save_file_name)

Post-Condition: A text file with name (save_file_name) will be created (or updated) in the same directory as the code the User is running. This file will contain the TS_Tree information and can later be used to recreate the tree by running a load_tree command (see Use Case 1).

3. Specific Requirements

3.1. External Interfaces (Inputs and Outputs)

Execute Tree

- Inputs
    - tree <TS_Tree>
    - time_series_data (<str> OR <panda_dataframe>)
        - Either the string name of a csv file possessing time series data or a variable holding the already opened time series data
- Outputs
    - Result_dictonary <dictionary>
        - A dictionary variable containing the output from executing the functions along the various paths from the root to leaf nodes in a TS_Tree on a given time series. The dictionary contains *key:value* pairs where the *keys* are the names of the leaf nodes (where the name is the operation referenced in the node plus it's index in the tree. EX: denoise-1, impute_outliers-2, etc.) and the *values* are the output of executing the path from the root to that leaf node given the particular time series.

Execute Path

- Inputs
    - tree <TS_Tree>
    - time_series_data (<str> OR <panda_dataframe>)
        - Either the string name of a csv file possessing time series data or a variable holding the already opened time series data
- Outputs
    - Result (a variable object output that can be one of the following objects depending on the the final node within the executed path)
        - A modified time series
        - A design matrix
        - A forecasting model
        - Error amount of a forecast
        - Float amount of how normal the time series data is

Add Node and Replace Node

- Inputs
    - tree <TS_Tree>
        - The TS_Tree object the User is wanting to edit

- parent_index / node_index <int>
    - Input from the User that specifies where to add/replace the node within the tree
        - for add_node User specifies parent_index which is the index of the node the User wants to be the parent of the new node
        - for replace_node User specifies node_index which is the index of the node the User wants to replace.
- Operation Inputs (*Inputs that are stored within each operation node. If not specified by the User will default to have a value of None*)
    - data_start <float>
        - Input from the User that is used for the following operations:
            - design_matrix
            - clip
            - assign_time
            - ts2db
    - data_end <float>
        - Input from the User that is used for the following operations:
            - design_matrix
            - clip
            - ts2db
    - increment <float>
        - Input from the User that is used for the following operations:
            - assign_time
            - denoise
    - perc_training <float>
        - Input from the User that is used for the following operations:
            - split_data
            - ts2db
    - perc_valid <float>
        - Input from the User that is used for the following operations:
            - split_data
            - ts2db
    - perc_test <float>
        - Input from the User that is used for the following operations:
            - split_data
            - ts2db
    - input_filename <str>
        - Input from the User that is used for the following operations:
            - ts2db
            - mse

            - mape
            - smape
        - output_filename <str>
            - Input from the User that is used for the following operations:
                - ts2db
                - write_to_file
        - layers <list>
            - Input from the User that is used for the following operations:
                - Mlp_model
- Output
    - tree <TS_Tree>
        - Users will have an updated version of the TS_Tree object (tree) which they can pass time series data into through an execute_path or execute_tree command to process the data.

Save Tree

- Inputs
    - tree <TS_Tree>
        - TS_Tree object the user wants to save
- Outputs
    - save_file <.txt file>
        - A new text file will be created in the User directory storing that TS_Tree data that can be called by a load_tree command

Load Tree

- Inputs
    - tree_file_name <str>
        - The name of the text file in which the TS_Tree that the User wishes to load is saved into.
- Outputs
    - tree <TS_Tree>
        - Users will have an updated version of the TS_Tree object (tree) which they can pass time series data into through an execute_path or execute_tree command to process the data.

3.2. Functions

Every time a node index value is passed into a TS_Tree function the first thing the function will do is run an indexing check on the node index value that was inputted. This check will confirm that the index value was non-zero and less than the number of nodes within the TS_Tree (less than because nodes within a TS_Tree are zero indexed). If the inputted index fails this validate check, then an error message will be printed to the User's terminal stating that the index was out of bounds and the tree function called will stop and return None.

Additionally, upon adding a node or replacing a node within the TS_Tree three different validate functions are called in the following order. The functions and their descriptions are stated below

validate_operator(operation)

> This function checks that the operation imputed by the User is a valid function that the tree can reference. If the operation is valid this error check will return True. Otherwise an error message will be printed stating that the operation string inputted by the user is invalid, the called tree command will stop and return None.
>
> Called by:
> - TS_Tree.add_node
> - TS_Tree.replace_node

validate_operation_order(operation, parent_operation)

> This function checks that the operation imputed by the User will not break the order of the tree structure. If the order after editing the tree to possess the operation is valid this error check will return True. Otherwise an error message will be printed stating that the parent_operation cannot be followed by operation and the called tree command will stop and return None.
>
> Called by:
> - TS_Tree.add_node
> - TS_Tree.replace_node

validate_inputs(operation, data_start, data_end, increment, perc_training, perc_valid, perc_test, input_filename, output_filename, m_i, t_i, m_0, t_0, layers)

> This function checks that the operation imputed by the User also possesses correct function inputs (type, and existence). If the operation was inputted along with the proper inputs then this error check will return True. Otherwise an error message

will be printed stating that the operation needs some set of function inputs and the called tree command will stop and return None.

Called by:

TS_Tree.add_node
TS_Tree.replace_node

Note for mlp_forecast:

The input filename that the User will pass into the function node as input_filename must store the time-series data in db format.

3.3. Usability Requirements

The user should be able to easily visualize trees and pipelines. Function outputs will be the inputs to following functions. Users should be able to clean up, scale, and visualize time series data.

3.4. Performance Requirements

This system was tested with CSV files that held as many as 43,000 values; 100% of the operations upon data sets of this size, or smaller, were able to be calculated in less than three seconds. Adding more functions to a tree pipeline positively correlates to a longer calculation time.

3.5. Software System Attributes

This software product was designed with maintainability, portability, and reliability in mind. To ensure that this product was able to be maintained henceforth, all technical roles associated with the system thoroughly documented their function implementation. This includes commented each variable declaration, design choice, and logic choice; additionally, all technical developers included descriptive docstrings explaining function parameters, return values, and any additional information needed to maintain and extend the project in the future.

Portability was also prioritized during every stage of building this system. All requirements were noted and were kept organized in the requirements.txt file. Ample documentation was also put in place to explain any version requirements expected of the user.

Lastly, reliability was an important pillar in the design of the product. High amounts of attention was given to this last stipulation to guarantee that the user could always count on the system. To accomplish this, all technical roles contributed to a variety of different testing suites; each suite tested a different module of the project.

4. References

IEEE Std 1362-1998 (R2007). (2007). IEEE Guide for Information Technology–System Definition–Concept of Operations (ConOps) Document. https://ieeexplore.ieee.org/document/761853

IEEE Std 830-1998. (2007). IEEE Recommended Practice for Software Requirements Specifications. https://ieeexplore.ieee.org/document/720574

ISO/IEC/IEEE Intl Std 29148:2011. (2011). Systems and software engineering — Life cycle processes — Requirements engineering. https://ieeexplore.ieee.org/document/6146379

ISO/IEC/IEEE Intl Std 29148:2018. (2018). Systems and software engineering — Life cycle processes — Requirements engineering. https://ieeexplore.ieee.org/document/8559686

Faulk, Stuart. (2013). Understanding Software Requirements. https://projects.cecs.pdx.edu/attachments/download/904/Faulk_SoftwareRequirements_v4.pdf

Oracle. (2007). White Paper on “Getting Started With Use Case Modeling”. Available at: https://www.oracle.com/technetwork/testcontent/gettingstartedwithusecasemodeling-133857.pdf

van Vliet, Hans. (2008). Software Engineering: Principles and Practice, 3nd edition, John Wiley & Sons.

5. Acknowledgements

The SRS template was provided by Juan Flores, who updated it from Anthony Hornof for CIS 422.