Authors -
Cameron Jordal
JT Kashuba
River Veek
Group - Keyboard Warriors
Last Modified - 2/10/21

The User Documentation goes over how a typical user can use the system and gives a walkthrough for a typical use case.

User Documentation

Basic Interaction:

The main interaction for this library is the use of the tree structure and the files. (Represented in Figure 1 at the end of this document)

Once the modules have been installed (see installation_instructions.txt and README.md), the user can create a tree, add functions to nodes, and execute the tree as a whole, or run individual paths from within the tree. For more information on this tree interaction, see SDS - Section 4.1.

Walkthrough:

To import the tree, import the tree module from the time-series module.

```
1 > from tree import *
```

The tree is an object, so the first thing that needs to be done is to create the tree.

```
2 > t = tree.TS_Tree()
```

The next step is to add functions that you would like to execute along the tree. This will add an impute_outliers operation as a child (index 1) of the root node (index 0).

```
3 > t.add_node("impute_outliers", 0)
```

This node will add a new operation, standardize, which takes the output of node 1, impute_outliers, as input upon node execution.

```
4 > t.add_node(“standardize”, 1)
```

This node adds a new operation, design_matrix, which takes the output of node 2, standardize, as input. It also takes in 2 arguments, data_start and data_end, which are each directly specified after the name of the function and the node it takes its input from.

```
5 > t.add_node(“design_matrix”, 2, data_start=20.0,
               data_end=10.0)
```

This node follows the same standard as before. However, it is also the only operation that can follow design_matrix in the tree. For instance, if you tried to put histogram after design_matrix, you would get an error: “Error - design_matrix needs to be followed by mlp_model.”

```
6 > t.add_node(“mlp_model”, 3, layers=[100, 50])
```

This node is also the only node that can follow mlp_model.

```
7 > t.add_node(“mlp_forecast”, 4,
               input_filename=”example_test_matrix.csv”)
```

These nodes show how a tree can be split into different branches, as write_to_file and mse are each taking the output from node 6, db2ts.

```
8 > t.add_node(“db2ts”, 5)

9 > t.add_node(“write_to_file”, 6,
                output_filename=”example_out.csv”)

10 > t.add_node(“mse”, 6,
                input_filename=”example_correct_out.csv”)
```

If you are constructing this tree, but you are confused about what a node’s index is, you can print the contents of the tree using the following statement.

```
11 > t.print_tree()
```

Now, when you are ready to execute, the simplest option is to execute the whole tree all at once. This can be done as follows; one important thing to note here is the file name. The execute_tree method will always call read_from_file as a default first node, and must take a file name as input. The execute_tree method will also return a dictionary of the returned values from the leaf nodes, nodes 7 (write_to_file) and 8 (mse) (see SRS, Section 6 - Use Case 6 for more details).

```
12 > t_dict = t.execute_tree(“example_test_in.csv”)
```

You can also execute individual paths along the tree and only get the return values of the specified path. This can be seen below where execute_path will take the same file as input, but run down only to node 8 (mse). The result will only be the output from the path to mse, and write_to_file will not be called.

```
13 > mse_eval = t.execute_path(“example_test_in.csv”, 8)
```

This saves the tree that has been constructed with the name provided. *Note: the function will automatically save it as a .txt file.*

```
14 > save_tree(tree, “example_tree”)
```

This will load a tree from a saved file. In this case, it's making a new tree from the one we just saved.

```
15 > saved_t = load_tree(“example_tree.txt”)
```

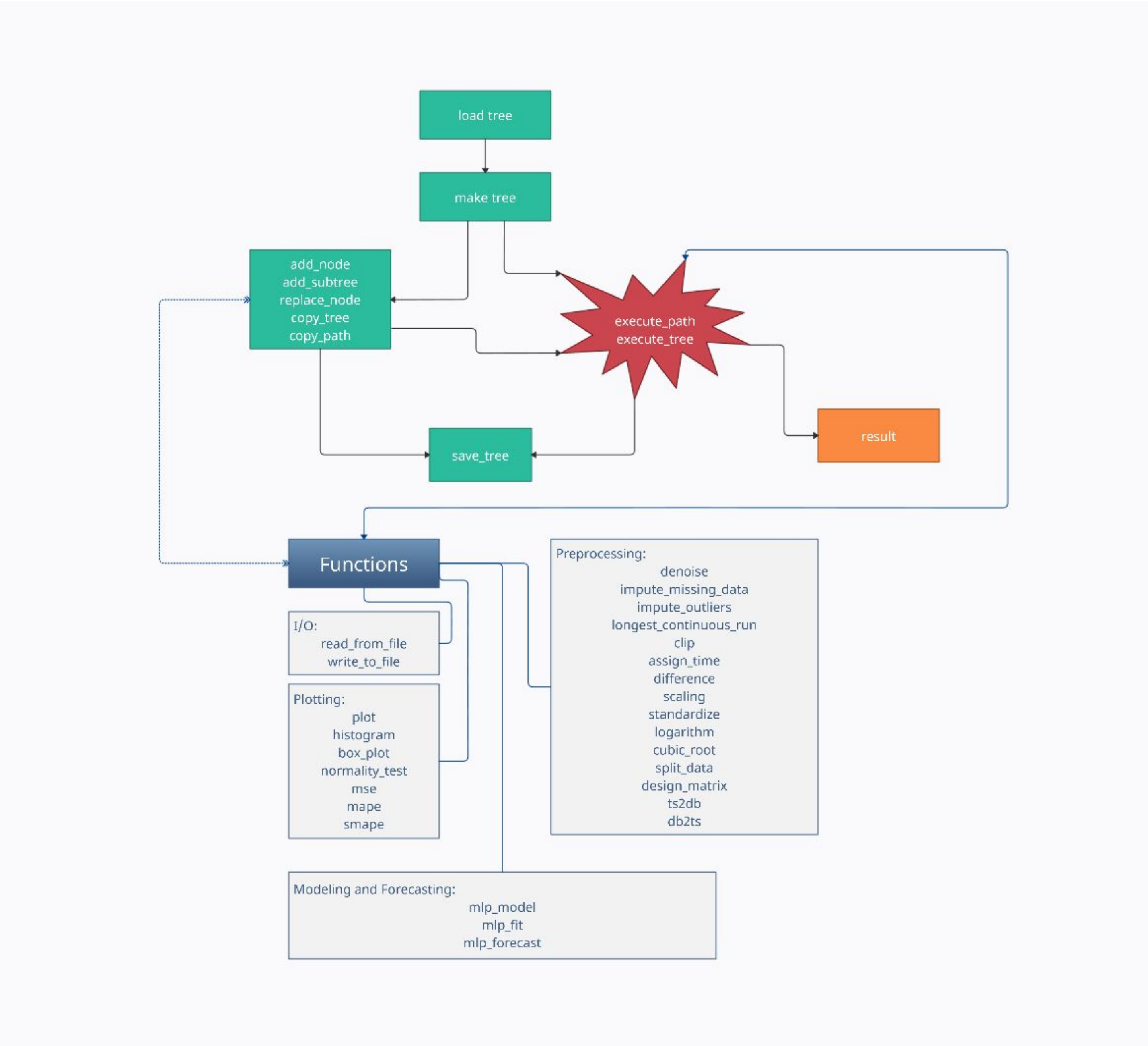

Figure 1

Figure 1 demonstrates a static model of the system architecture. *Functions* is connected to two boxes to illustrate that function calls are *referenced* in nodes created by the user (dotted line), but aren't actually invoked / called until execution (solid line). The user may also save a tree at any point in the construction process, either before or after execution.