Project 1
Software Design Specification

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

-- 2-9-2021 --

-- v1.4 --

Table of Contents

1. SDS Revision History

Date	Author	Description
2-9-2021	tc	v1.0 Removed template instruction and modified all but section 4 to fit Project 1 by The Classy Coders
2-9-2021	tc, yc, bg jr, sp	v1.1 Updated Section 4 for each group member's allocated module(s)
2-10-2021	tc, bg	v.1.2 Updated table of contents to better reflect changes from v1.0
2-10-2021	bg, sp	v.1.3 Updated alternative design sections
2-10-2021	tc, bg	v.1.4 Proofread and finalized formatting

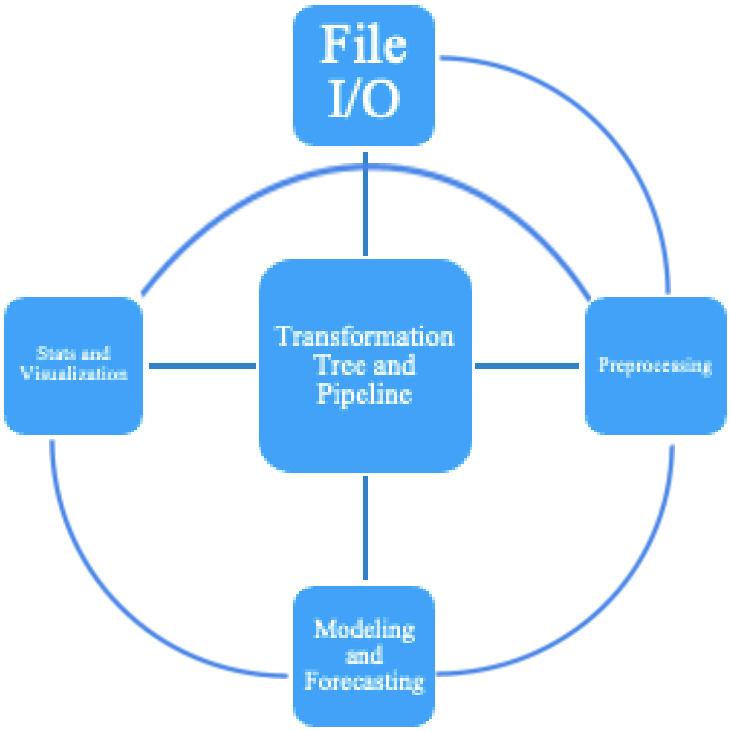
2. System Overview

Our Project 1 supplies the user with an all-inclusive library that gives the ability to create a transformation tree and execution pipeline. These object-oriented data structures allow the user to create and modify the tree, and also to create pipelines that successively execute different processes regarding time series files. These processes include various preprocessing methods (such as file I/O, data massaging involving pandas DataFrame creation/manipulation, and more), modeling and forecasting, statistics, and visualization. As an overall job, the user would typically begin by reading in a time series data file and would then proceed to call zero or more data-massaging methods. This modified time series would then be passed into a time series-to-database method that parses the data into training, testing, and validation sets. These sets are used to pass into the models as arguments, which are then trained and used to forecast new data. This forecasted data can be compared to the test data through various statistical tests and then visualized graphically. Data scientists can further use this transformation tree design to run different pipelines and see which forecasting model works best.

3. Software Architecture

Set of Components

- File I/O
    - Reads to and from a time series data file
- Preprocessing
    - Data-massaging methods
    - Splitting the data into training, testing, and validation sets
    - Transfers the time series into a database
- Modeling and Forecasting
    - Trains a model with the training set
    - Predicts values given the test set
- Stats and Visualization
    - Plots the time series data
    - Statistically compares the test sets to the predicted sets
- Transformation Tree and Pipeline
    - Creates an all-encompassing transformation tree that stores operation nodes
    - Functionality to add and delete nodes, create branches, and execute static pipelines
    - Tree and pipeline file I/O



Interface Specification

File I/O reads from a data file containing a time series and stores the information in a pandas DataFrame object. This has a direct connection to preprocessing in that preprocessing takes this DataFrame object and can perform various forms of manipulation. The Preprocessing module connects to the Modeling and Forecasting by the ts2db method, where the data is parsed and then fed into the models to train them. Preprocessing also connects to the Visualization methods because the plots pass in time series DataFrame objects as arguments. The Forecasting component has outputs that directly feed into the Stats component by direct comparison with test sets. Hence, the Preprocessing module connects to the Stats component as well (it supplies the test sets needed for error analysis). The Transformation Tree and Pipeline module has direct connections to all other modules by storing their operations as nodes in the tree. The Pipeline then takes branches of the tree and forms static objects that can be stored into files.

Design Rationale

Connections between modules were added on a need basis. We structured our project by splitting the project specification into these modules and then evaluated the necessary connections between them as we developed their respective functionalities. We decided to structure our design as such because upon first inspection, it seemed most sensical to divide up the modules by

major section. We thought this was an efficient decision because otherwise the software development process could have become messy as modules would have crossed over on themselves as we continuously programmed them.

4. Software Modules

4.1 Preprocessing / File IO

4.1.1 Role

The role of preprocessing is data manipulation. Included in preprocessing is also file IO, which is used to read and write data to / from CSV files. Preprocessing can assign time and date values that are required in order to plot time series data. This module can also extract information between specified dates, impute missing values and outliers so the data is in a cleaner format. Preprocessing can also find the longest continuous run of a data set (meaning longest run with valid numbers) and provides information to our prediction modules.

4.1.2 Interface with other modules

Preprocessing returns (or creates) a TimeSeries object that contains data from the input file. We store the TimeSeries data in a variable called self.data which is then usable by other modules by accessing the data variable and performing the desired action on that data.

4.1.3 Static Model

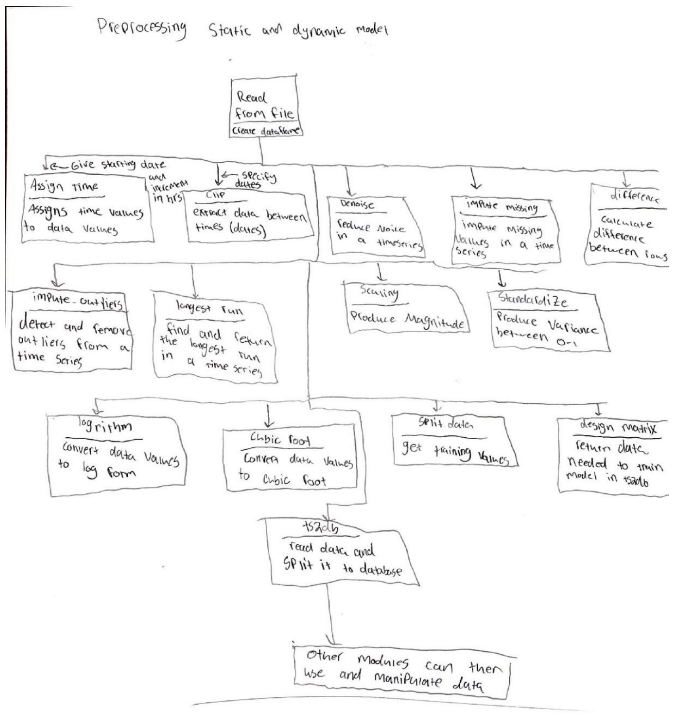


Figure 1. Preprocessing Static model

4.1.4 Dynamic Model

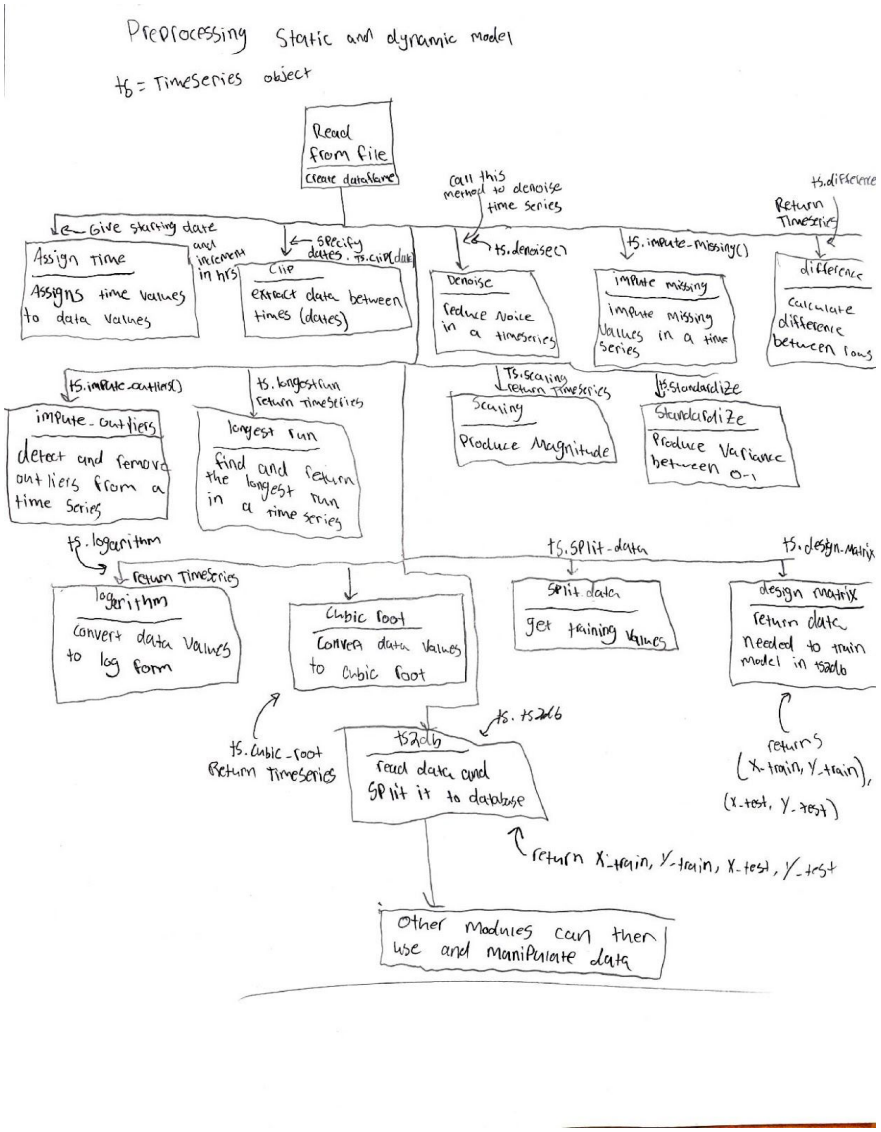


Figure 2. Preprocessing Dynamic Model

4.1.4 Design Rationale / Alternative design

Since we were given project specifications for this project, our design rationale for preprocessing was decided for us. The only caveat to this is within the specified functions themselves, which is described in the preprocessing documentation. ***Alternative Design:*** When designing preprocessing, we were not able to consider an alternative design process as we had to follow along with the project description and adapt our implementation to fit the specification handout.

4.2 Modeling and Forecasting

4.2.1 Role

The role of Modeling and Forecasting is to take a slice of the time series data as an input training set and then pass this set into a machine learning model to create a trained model. This trained model is then supposed to take in an input test set from the same data and use it to predict future data.

4.2.2 Interface with Other Modules

The Modeling and Forecasting module takes time series data from the Preprocessing module, models and predicts it, and then passes the predicted set to the Statistics and Visualization module.

4.2.3 Model

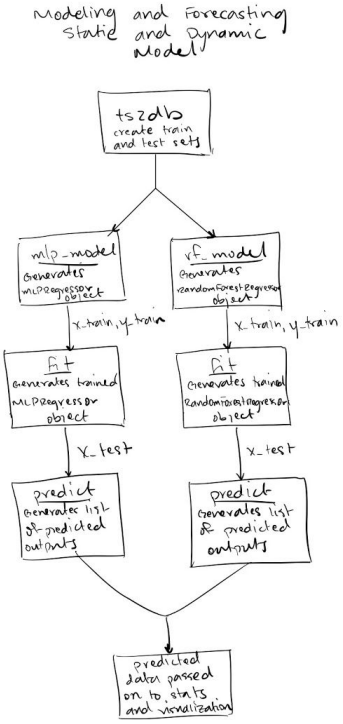


Figure 3. Modeling and Forecasting Static/Dynamic Model

4.2.4 Design rationale

Standard models from scikit-learn were adapted and wrapped to fit our project specification document. These models were chosen because they had the functionality we wanted as far as their input and outputs and because they are very well-documented neural network models, so we were able to integrate them into our project quite easily. ***Alternative Designs:*** Some background research was done on several different machine learning models, but we decided that the multilayer perceptron and the random forest models were the most reliable and were best suited for this project.

4.3 Statistics and Visualization

4.3.1 Role

The role of statistics and visualization is to help visualize the data using plots, histograms, and box plots and to run statistical tests on the time series data. These statistical tests include a Shapiro-Wilkinson Normality Test and three error checking tests: MSE, MAPE, and SMAPE.

4.3.2 Interface with Other Modules

The plot, histogram, box_plot, and normality_test functions all take in a TimeSeries object from the preprocessing module and use the data stored within that object. This data is obtained from the read_from_file method so that must be called *before* you use these functions.

Additionally, before the error calculation functions (mse, mape, and smape) can be run, you must run the ts2db function in Preprocessing to obtain the y_test argument and you must create a forecasting model using the predict function in Modeling and Forecasting to create the y_forecast argument.

4.3.3 Static Model

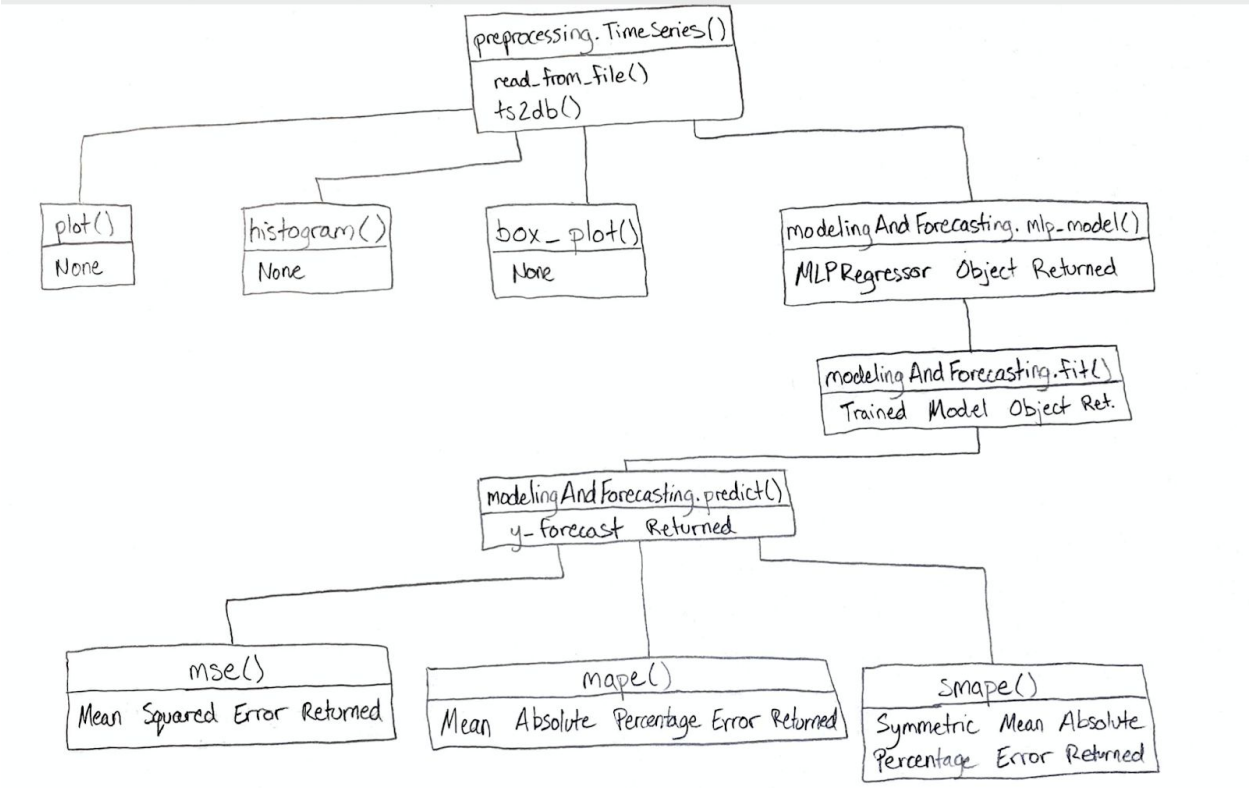


Figure 4. Statistics and Visualization Static Model

4.3.4 Dynamic Model

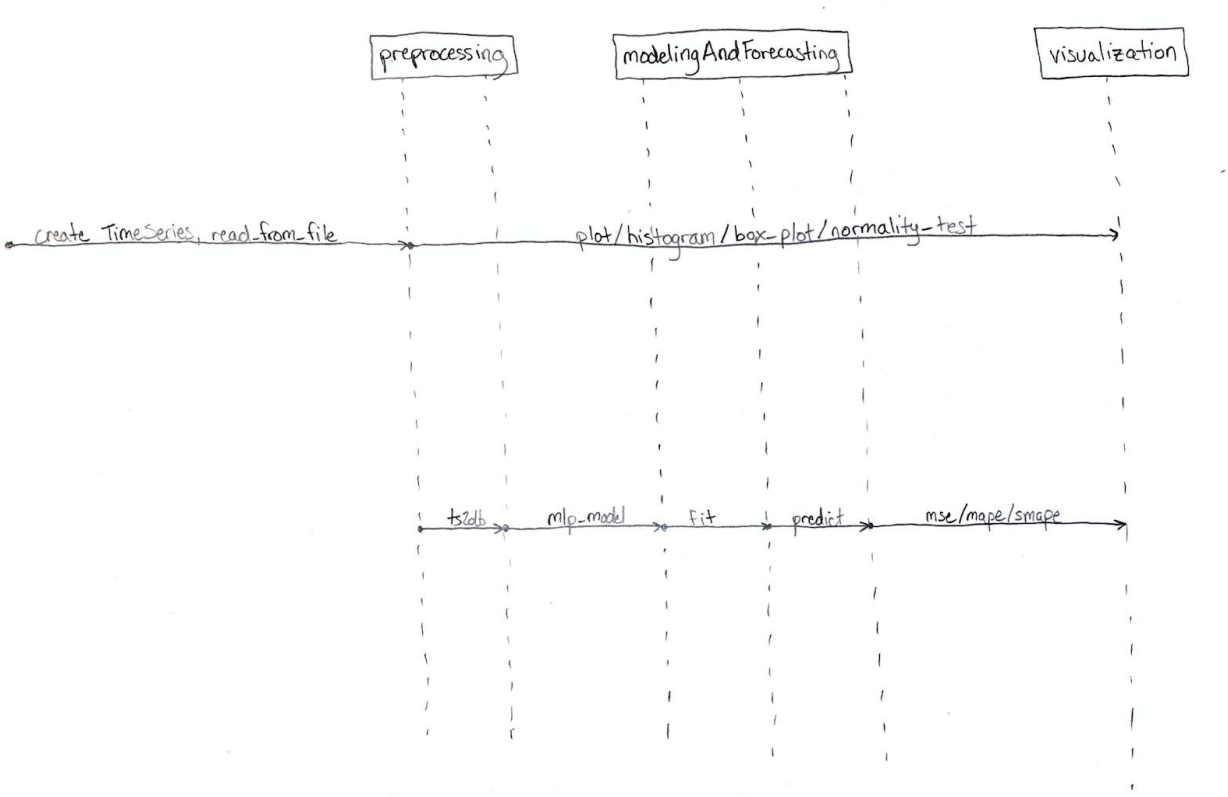


Figure 5. Statistics and Visualization Dynamic Model

4.3.5 Design rationale

The major design decisions that came with preprocessing ended up happening after the rough draft of the entire component was completed.

Alternative Design 1: Initially the plot, histogram, box plot, and normality test methods all took csv filenames. These files were supposed to be the files containing time series data. With that being the case, plot, histogram, box plot, and normality test would've used panda's read_from_csv() method to create a dataframe of the data.
Final Design Decision 1: Since we realized we were already doing this in the preprocessing section, we found it unnecessary to do it again in the statistics and visualization section. For this reason we decided to make the functions take a TimeSeries object defined in the preprocessing section instead.

Alternative Design 2: Another major design decision came to the plot function when we were integrating all the components together at the end. Initially this function took a list of TimeSeries objects so that multiple time series could be plotted at the same time.

Final Design Decision 2: With how our transformation tree is set up, it didn't make sense for this to be a list so we decided to make it accept only one TimeSeries object at a time. This made building branches in the tree much simpler.

4.4 Transformation Tree and Pipeline

4.4.1 Role

The role of the transformation tree and pipeline module is to provide a tree like structure that allows the data scientist to build, edit, execute, and export branches of operators easily while searching for the right pipeline of operators.

4.4.2 Interface with Other Modules

The TransformationTree class includes functionality to store and execute functions from the Preprocessing, Statistics and Visualization, and Modeling and Forecasting modules.

4.4.3 Static Model

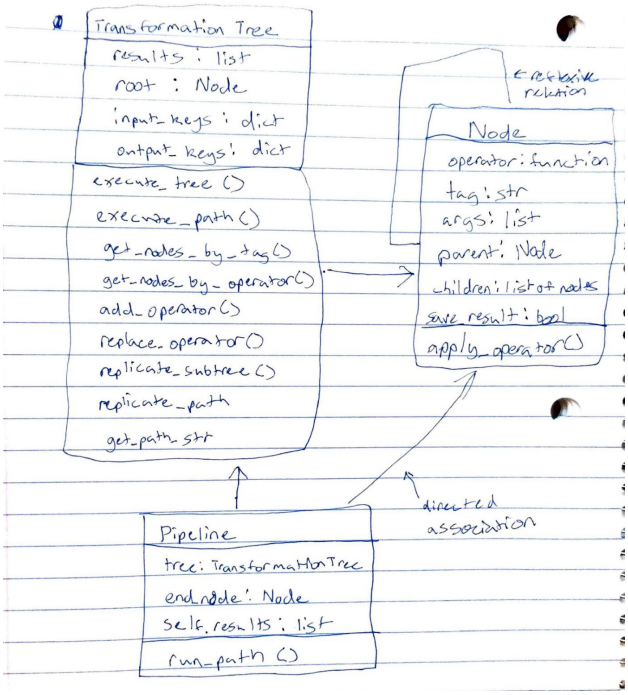


Figure 6. Transformation Tree and Pipeline Static Model

The above figure shows the three main classes in the tree module and their relationships, with the TransformationTree class containing a reference to a Node object, the Node class containing a reference to multiple Node objects, and the Pipeline class containing references to both Node and TransformationTree objects.

4.4.4 Dynamic Model

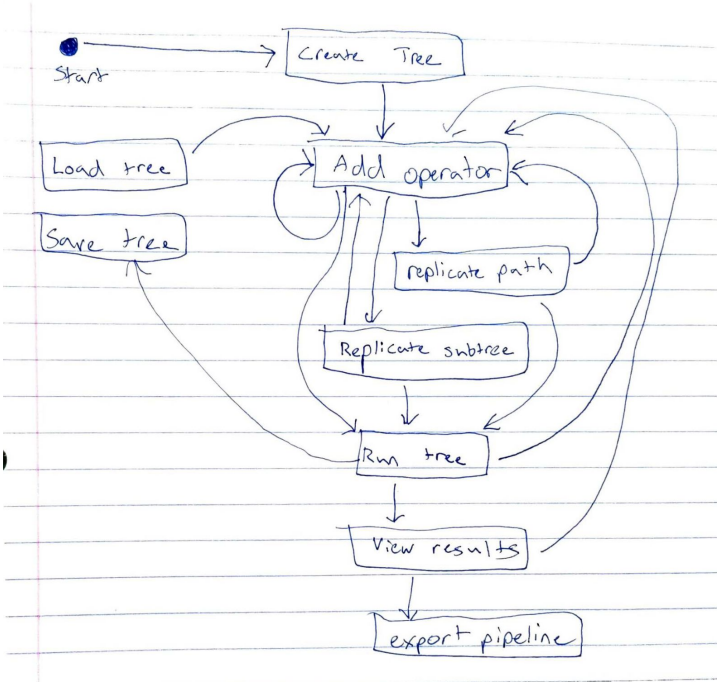


Figure 7. Transformation Tree and Pipeline Dynamic Model

The above figure shows a standard flow of operations for a TransformationTree during its use. Because any method of the tree can be called at any time this model doesn't show every way operations can be ordered, but instead one that's likely to appear.

4.4.5 Design Rationale

From the project specifications, we knew that we needed to build a tree of operators that we could execute. This requirement to be able to execute operators led to us storing references to functions in our tree's nodes, rather than other types of values that represent certain operations like strings.

Another decision we made was how to pass the results of previous operators onto their descendants farther down the tree. Initially, an ***alternate design*** we looked at included only passing the output of each operator directly to each of its children, but decided this wouldn't work. Because some operators rely on data from multiple previous operators, we decided that the tree would need to:

1. Track the results of every operator in a branch as the branch executes
2. Pass the necessary data from all the previous operator executions to the currently executing operator.

To achieve this we used a branch specific dictionary that stores the results of each operator as it executes, using predetermined keys. Then, when executing another operator, we'd use a second set of predetermined keys to pull those stored values

from the branch dictionary, and feed them into the executing operator. This allows branches of operators to pass results to future operators in a non-sequential way, something that's necessary for most pipelines.

5. Dynamic Model of General Operation Scenario

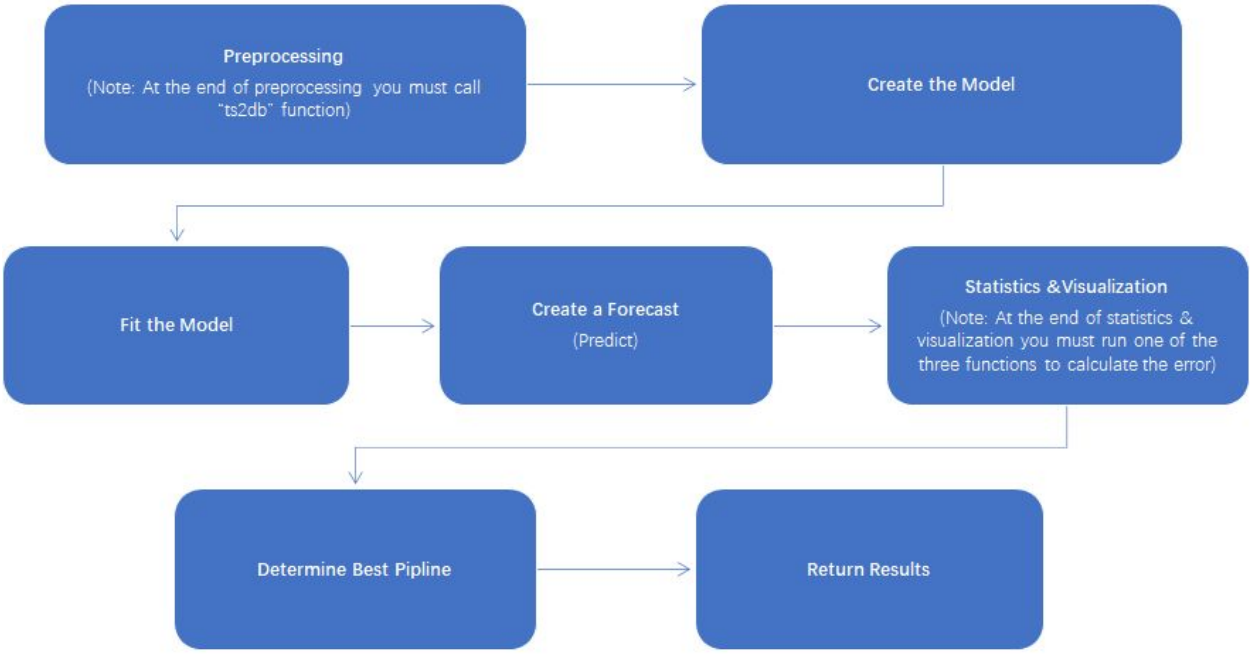


Figure 8. Dynamic Model of Overall Operational Usage

6. References

Faulk, Stuart. (2011-2017). CIS 422 Document Template. Downloaded from https://uocis.assembla.com/spaces/cis-f17-template/wiki in 2018. It appears as if some of the material in this document was written by Michal Young. Last edited by Anthony Hornof in 2020.

7. Acknowledgements

This document builds on the template document last edited by Anthony Hornof in 2020.