Project 1
Software Requirements Specification

T. Christenson (tc), Yifeng Cui (yc), Brian Gunnarson (bg), Jacob Rammer (jr), Sam Peters (sp)
2-10-2021
v.1.2

Table of Contents

1. SRS Revision History

Date	Author	Description
02/09/2021	jr	v.1.0 Initial SRS documentation

02/09/2021	jr	v.1.1 Started concepts of operation
02/10/2021	jr, tc, yc, bg, sp	v.1.2 General revisions

2. The Concept of Operations (ConOps)

The concepts of operation for project 1 is to provide data scientist with tools that allow them to manipulate and visual time series data. Data is provided to our software in CSV file format that is either a complete data set (date, time, data value) or an incomplete CSV file that is missing the time and date and contains the data value (such as temperature). We needed to provide the tools for data scientist to manipulate the input data so that they can create a complete data set. In addition to data manipulation, we needed to provide data scientist the tools to plot and model this manipulated data so that data trends can easily be visualized by both data scientist and ordinary people. We believe that visualization is important because it is difficult for people unfamiliar with how to read data in a raw format to quickly and easily see trends within a supplied dataset. We needed to accomplish this using a machine learning pipeline, as it is up to the data scientist to choose the best path of operation as they should know how they would like to manipulate or display the data.

2.1. Current System or Situation

Timeseries analysis requires a large amount of exploration in order to get meaningful results. A data scientist might go through dozens of pipelines consisting of different operators and arguments before settling on a pipeline that accurately predicts future timeseries values. To aid in this exploration, our library provides many common data manipulations, modeling, forecasting, error measuring, and visualization functions that are necessary for the task of timeseries forecasting, as well as a tree like structure that allows the data scientist to build, edit, and test branches of operators easily. We were motivated to tackle this project when we weren't able to find a system that was as "all in one" as this one. There are existing tools that have been developed to manipulate time series, but none that we could find that had the same functionality of our system.

2.2. Justification for a New System

Current systems that have been previously developed rely on multiple libraries that make it difficult for the end user to efficiently use the software. In our project, we limited the required libraries and included setup instructions that will allow the user to effortlessly install required libraries. Since this project came from Professor Flores, who seems to have experience in the data science field, we assume that the specific functionality of this system is missing from what is currently offered in the field. We believe this is the huge motivating factor in the software requirements listed in the project handout.

We have provided an easy way for the user to manipulate data and import missing data so the supplied dataset can be considered complete. We have also provided easy methods to denoise and impute missing data values from an input file. In addition to data manipulation, we have added ways for the data scientist to plot and visualize time series data so that the trends can be easily seen. Finally, we have done this all with a machine learning pipeline that uses a tree structure that allows the data scientist to choose the route they would like to go.

2.3. Operational Features of the Proposed System

Our system, as stated previously, allows data scientist to manipulate and visualize data using a machine learning pipeline. We accomplish this by reading CSV files using Pandas, which is a data analysis tool that we are using to represent a time series. We can also write this manipulated time series data to a CSV file so it can be stored for later use. One of the most important functionalities of this software is to add missing data such as dates, times, and NaN (not a number) data values as well as visualization. Our system can also detect and remove outliers in the data, denoise a time series, and extract sections of the time series between a specified date, creating a new time series.

We also provided the tools to find the difference between data points so that the trend can be easily visualized. Finally, we provided tools that are easy to use that can also visually display the manipulated data on a graph, so that data trends can be easily seen. We believe this system will benefit the user by providing easy access to these tools.

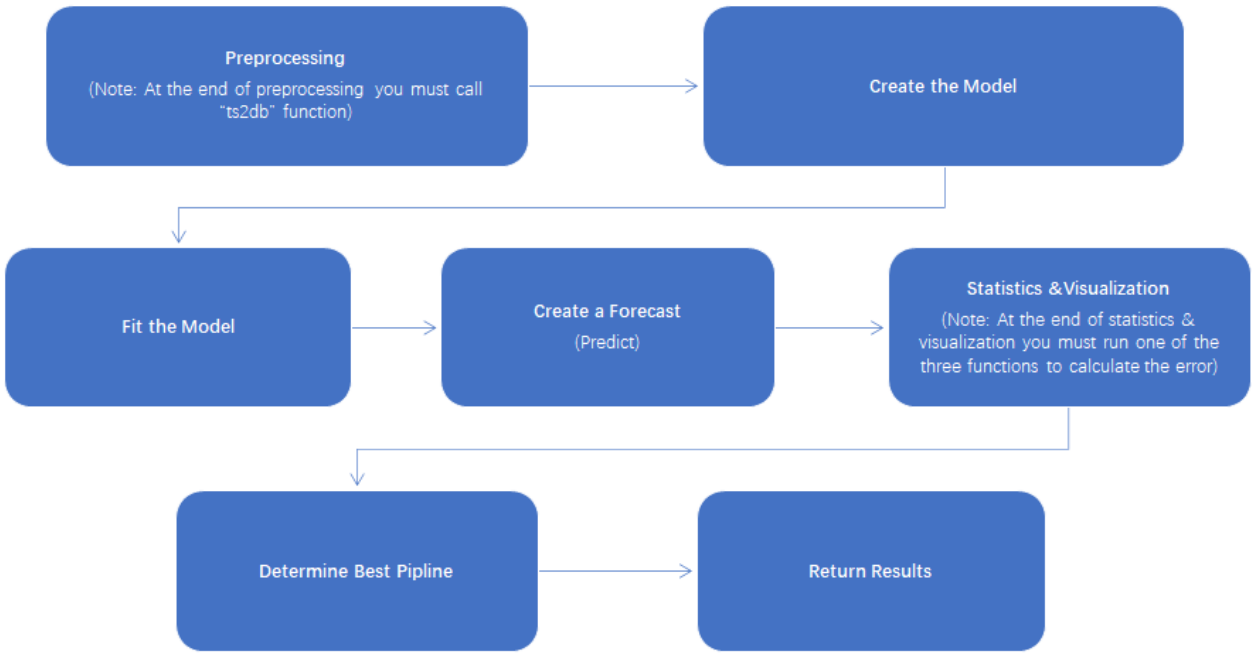
2.4. User Classes

When considering the user class for this system, we are assuming the user is a data scientist. This means that we expect the user to already be familiar with existing systems that are provided, and they need to know what they would like to accomplish while using our system. Since we expect the user to know what they need to accomplish, we don't provide any feedback, we just provide the user with the tools to perform these specific tasks. In short, we expect the user to have a high level of expertise in the field and for them to be familiar with time series.

2.5. Modes of Operation

Python Library: The user (a data scientist) will import our library in Python3 and use the included modules to create a transformation tree of operators. The expectation is that this library is nothing more than a set of tools. The data scientist will use their programming knowledge to produce a working timeseries forecasting pipeline using the included modules in our library.

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



Preprocessing
(Note: At the end of preprocessing you must call "ts2db" function)
Create the Model
Fit the Model
Create a Forecast
(Predict)
Statistics & Visualization
(Note: At the end of statistics & visualization you must run one of the three functions to calculate the error)
Determine Best Pipline
Return Results

Use Case: Creating a new TimeSeries object.
Brief Description: In order to manipulate any data in a time series, you first must create a TimeSeries object so that you can later manipulate or visualize the data.
Actors: A data scientist
Preconditions:

- The data scientist must have an input file in CSV format that has met one of these two conditions stated below and a header describing what each value represents.
  - Only has data values and no date or time associated with that value. We will talk about these types of files later.
  - OR has data values, time values, and data values in this order.
- **Steps to complete this task:**
  - Create a new TimeSeries object at the root of the tree.
  - Call the read_from_file method to import data from a CSV file.
- **Postconditions**
  - The data scientist now has a TimeSeries object at the root of the tree that contains data.

Use Case: Manipulating data where the associated times with each data value are missing, outside of a tree.
Brief Description: As talked about in the previous use case, some input files can be missing the dates and times associated with each data value. Here we want to add these missing times and dates to the TimeSeries object, we need to add these now so they can later be plotted accurately.
Actors: A data scientist
Preconditions:

- Have a TimeSeries object at the root of the tree (previous section)

Steps to complete this task:

- Add an operator to the tree.
- Supply the starting date as a string in the format of mm/dd/yyyy hh:mm where there is a space between yyyy and hh. For example, a valid date string is 01/01/2020 13:00. Time is in 24 hour format.
- Supply an hour increment. This must be a whole number (int) that increments the time by a specified whole hour.

Postconditions:
- A TimeSeries object that now has a date, time, and data column that can now be later manipulated or plotted.

Use Case: Training, modeling, and predicting data
Brief Description: The user may want to create and train a machine learning regression model that is capable of predicting future data.
Actors: A data scientist
Preconditions:
- Split time series into training and testing sets. Directly from ts2db.

Steps to complete this task:
- Create a model object by calling the preferred model function.
- Pass the preferred model object as well as input and output training sets into the fit function.
- Pass the trained model object and the input test set into the predict function.

Postconditions:
- A trained model object
- A list of predicted values (of length 'output_index/output_dimension')

Use Case: Plotting a time series.
Brief Description: Data scientist often want to plot a timeseries on a graph so that data trends can be easily recognized or to show data trends to ordinary people.
Actors: A data scientist
Preconditions:
- Have a TimeSeries object that has dates and times associated with each data point. Refer to the two previous use cases.

Steps to complete this task:
- Add an operator to the tree.
- Call the preferred visualization method.

Postconditions:
- A graph of the TimeSeries object data

3. Specific Requirements

3.1. External Interfaces (Inputs and Outputs)

Program Input:
- The name of each input file can vary as they are CSV files.

- The purpose of CSV files contain data that is stored into the TimeSeries object during creation.
- Source of input are files stored on the data scientist computer.
- CSV files must include a header stating what each value means.
- There are no units of measurement in our files
- Data formats are CSV files only

Program Output:

- The only output of our software are CSV files. The user specifies the name of the CSV file such as file.csv.

TS2DB:

- **Input:** Comes partially from the data stored in the TimeSeries object from read_from_file and partially from the data scientist using our library themself
    - The data scientist tells the program how to split the data by giving it the percent for training, the percent for validation, and the percent for test
    - The data scientist can choose to use the data stored in the TimeSeries object from read_from_file or he can choose to input a new file containing time series data
    - The data scientist can also choose whether they want to change the input and output indices of the design matrix that will be created or use the default values provided by the method (input_index=0, and output_index=25)
- **Output:** Parses out the TimeSeries object into different training and test sets
    - Specifically, x_train and y_train are training sets that are meant to be used in the models from the Modeling and Forecasting module
    - x_test and y_test are test sets that are used with the predict function from Modeling and Forecasting
    - Input_index and output_index describe the allocation for input (X) data and output (Y) data, respectively

Modeling and forecasting

- **Input:** Comes from the output lists from ts2db
    - The mlp_model takes the input_index as an argument
    - The fit function takes x_train and y_train as inputs to train the models
    - The predict function takes x_test in order to form a prediction set that is supposed to reflect y_test
- **Output**
    - Both model functions return model objects
    - The fit function returns a trained model (whichever model is passed in as an argument)
    - The predict function returns a list of forecasted data

Tree file

Our system allows a data scientist to save and load a tree to a file.

- **Input:** A .sav file that contains a tree information stored by Pickle
    - Purpose: To load a tree file that has time series data stored
    - The source of this file is stored on the data scientist computer
    - Format: .sav files
- **Output:**

- A .sav file that contains the current state of a tree

3.2. Functions

To process user input, the user must input a string representing the name of the file (ex: /Documents/file.csv). The read_from_file method has a try and except statement so that if an invalid file is provided, the system does not crash. We have also included other error handling in our system to catch invalid input, so the system does not crash when something unexpected is fed into the program. There are no error handling methods when writing the output file, as the write_to_file is using a method from the Pandas library. In other methods that require user input, we also used try and except statements so that the program would not crash, such as in assign_time if an invalid date string was entered.

Our tree also includes compatibility checks so that the user cannot add an operator to the tree if the required input for the operator is not expected to be created at the time of execution. As an example, the model.fit operator must precede the model.predict operator in the branch, otherwise trying to add the model.predict operator will throw an error because a trained model is not expected to be created before the model.predict operator is called.

3.3. Usability Requirements

We design our system to be as accurate in our computations as possible. We do not have a measurement on effectiveness, efficiency, and satisfaction criteria in specific contexts of use.

3.4. Performance Requirements

Our system also does not have any performance requirements but users should be cautious when running the normality test function. If they have a data set larger than 5000 entries, there could be issues calculating the p-value for the hypothesis tests. Since we are using a 3rd party library to perform this Shapiro-Wilkinson normality test, the warning is coming from them, not us.

3.5. Software System Attributes

Our system includes documentation on how to setup the systems requirements in a Python virtual environment. We also developed the system so that it can handle unexpected input without having a critical failure during runtime. The most important attribute for our system is to provide data scientist the tools to manipulate data in an easy and effective way, which we believe our system accomplishes.

4. References

This document currently has no references.

5. Acknowledgements

This document currently has no acknowledgements.