SOFTWARE REQUIREMENTS SPECIFICATION

for

Comparative Study of Machine Learning Methods to Estimate Water Demand in a Coastal Tourist Town

Version 1.0

Prepared by: Jesuino Vieira Filho
Submitted to: Giovani Gracioli

March 14, 2022

Contents

1 Introduction

1.1 Purpose

Proper management of water resources is essential for human well-being and socio-economic development in our contemporary cities. In fact, water is necessary and indispensable for sustaining human life and due to its great importance, many civilizations developed along riverbanks. However, the use of increasing amounts of water brought a series of problems that compromise the quality and durability of available water resources. Much of this increase in water demand is the result of a combination of population growth, economic development and changes in consumption patterns (UN-Water, 2021), which emphasizes the importance of proper planning and management of water supply systems. An important component in optimizing and implementing effective management programs for these systems is accurate forecasting of water demands.

Forecasting water consumption demand plays an important role in the optimal operation of water collection, treatment and distribution systems. In these cases, capturing water consumption data can help, for example, in monitoring the system and also in obtaining consumption forecasts, supporting system operation decisions (Singapore, 2016). This process makes it possible to understand the underlying factors influencing water use, optimize pump operation and save energy due to controlled pumping, as well as inform the population about likely peaks in demand and avoid any water shortages. As a result, cities with a smaller water and carbon footprint can be observed, a growing trend due to the pressures of climate change – one of the biggest challenges of our time according to the United Nations UN (2015).

1.2 Intended Audience

This document is intended for developers responsible for designing the tests and foreground implementation. Also, it is designed to guide both manufacturers of components for AMRs and those responsible for final applications with robots of this nature who wish to adopt LIN as a new standard in their products.

1.3 Intended Use

This project is educational in nature and is part of a broader project, a bachelor's degree thesis. A researcher can use this project to replicate the results or reuse the software framework, as well as continue the work developed.

1.4 Project Scope

The main objective of this work is the study and evaluation of machine learning (ML) methods for forecasting water consumption demand in a Brazilian coastal city, namely Guaratuba – state of Paraná. Real data from the water distribution system (water consumption data) and a weather station (meteorological data) will be considered. In addition, data referring to holidays, vacation periods and school holidays will be part of the analysis, as the city receives a large number of visitors from other locations during these periods. Due to its significant population variation due to tourism, the application scenario is both interesting and challenging. It is hoped that, with such a comparison, specific methods may be indicated as more or less suitable for the data in question.

The project scope can be summarized in the steps of the knowledge discovery process in databases (KDD), as illustrated in Figure 1.1. The final software can be thought of as a framework developed in a structured way the execution these steps, which can also serve as a basis for the organization and development of other similar projects.

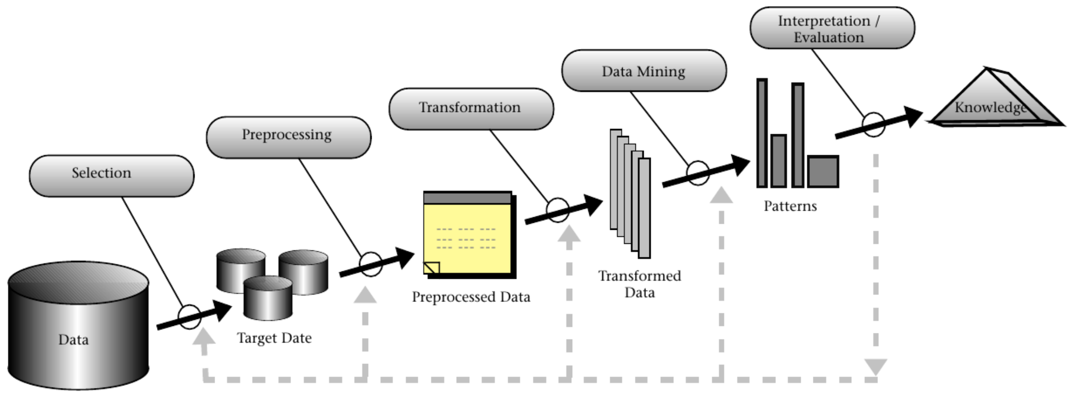

Figure 1.1: KDD process (Fayyad et al., 1996).

- **Selection:** production of treated water and meteorological data collected from 2016 to 2019 will be used. These databases are already available for use, having been provided by SANEPAR (Companhia de Saneamento do Paraná) and SIMEPAR (Sistema Meteorológico do Paraná). In addition, data referring to holidays, vacation periods and school holidays will be collected from the internet.

- **Preprocessing:** the data will be submitted to exploratory analysis and preprocessing in order to identify and remedy possible problems in the data.

- **Transformation:** once the data is pre-processed, the necessary transformations will be applied in order to feed the model. It's important to note that different methods may require different transformations.

- **Data Mining:** having the data in a format suitable for analysis, different forecasting methods will be used. Within the scope of this project, the comparison of at least three different methods is foreseen.

- **Iterpretation/Evaluation:** the results of the computational experiments carried out with different methods will be evaluated together, allowing to identify not only the method that comes closest to the desired results, but also which subsets of descriptive attributes are more suitable for the problem.

The methods that will be used in the comparative study will be defined throughout the project. In this sense, the identification of relevant methods for the scenario in question is also an objective of the project. As a first suggestion, linear regression (usually used as a baseline), k-nearest neighbor and support vector regression will likely be used.

To carry out the project implementations, the programming language `Python` will be used. The motivating factor is that most of the available libraries and sources that can be used in the project are developed in `Python`, such as the Python Data Analysis Library (McKinney, 2010), known as Pandas, and Scikit-learn (Pedregosa et al., 2011).

1.4.1 Feasibility

The project does not require face-to-face laboratory activities and can be developed entirely remotely. Thus, the project is viable even as a result of the pandemic. For the study and implementation of detection methods, conventional personal computers (desktops or notebooks) available will be used. For the analyses, real data already in the possession of the project proponent will be used. All libraries that will be used for the analysis of the referred project are open source. Thus, since all the material needed to carry out the project is available, it is believed that the project is fully feasible.

1.5 Acronyms

KDD Knowledge Discovery in Databases

ML Machine Learning

RAM Random-access Memory

SANEPAR Paraná Sanitation Company

SIMEPAR Paraná Meteorological System

SRS Software Requirements Specification

2 Overall Description

The present work aims to study, evaluate and compare machine learning methods for predicting water consumption demand in a coastal city, in which a high variability of water consumption due to tourism is observed. It is hoped that with this assessment it will be possible to indicate the most suitable methods for forecasting in this specific scenario, considering its peculiar characteristics. Besides that, the final software can be thought of as a framework developed in a structured way, which can serve as a basis for the organization and development of other similar projects.

2.1 User Needs

This project is designed mainly for researchers, although the result (i.e., the best method obtained through the comparison) might be employed in a water management system. For the user who wants to reproduce the obtained results, entry point functions will be provided to execute the KDD steps described in Section 1.4.

2.2 Assumptions and Dependencies

It is assumed that the user has a Linux operating system, as if necessary, the instructions provided may not work on Windows. As a single dependency, we have Python in its version greater than or equal to 3.8.12. Nevertheless, it is recommended to have a computer with a configuration equal to or greater than the following.

- Random-access memory (RAM): 16 GB
- Processor: Intel Core i7 8th Gen

This is not strictly necessary, especially since the running time of the algorithms is not important for the analysis. However, it is to put the developer on the safe side, as it is in this configuration that the project will be developed.

3 System Features and Requirements

3.1 Functional Requirements

- FR-1: User should be able to run KDD pipeline.
- FR-2: Data that is not at hand should be downloaded from the internet.
- FR-3: Predictions should be saved for each model.
- FR-4: Visual representations resulting from the evaluation should be saved.

3.2 External Interface Requirements

- ER-1: User interfaces are provided in a Makefile.
- ER-2: The computer should be connected to the internet.

3.3 Nonfunctional Requirements

- NR-1: Data should be saved in a single file after Preprocessing task.
- NR-2: Error metrics should be used to evaluate models.
- NR-3: The software should be modularized.
- NR-4: The software should be well documented.
- NR-5: The software should be easy to use.

4 System Modeling

This chapter describes the conceptual model that defines the structure and behavior of the system. The project proposal is to build a system capable of comparing different machine learning models to predict the water demand in a coastal city. The system executes all the KDD steps shown in Figure 1.1 and at the end of its execution is able to present the results clearly through tables and diagrams. Models are saved for future use, such as to optimize the operation of SANEPAR's water distribution systems.

4.1 Use Case Diagram

Figure 4.1 shows the use case diagram. A single actor is described: the SANEPAR developer. This person can edit the configuration file and start the KDD pipeline. Note that the user does not have much interaction with the software. The complexity lies in the sequential execution of the KDD steps, which are briefly described in this diagram.

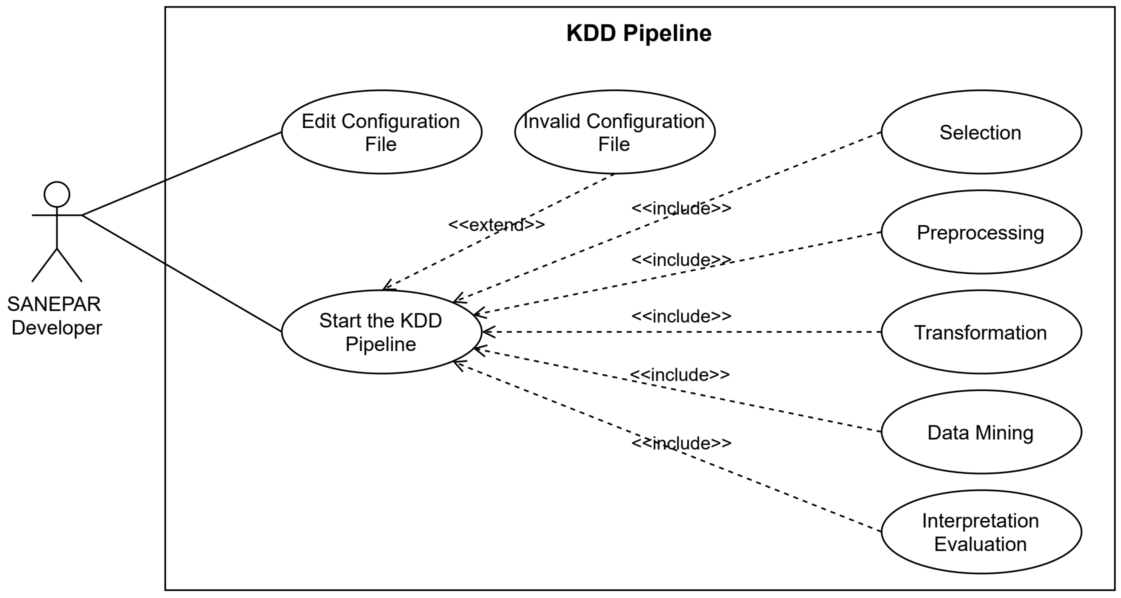

Figure 4.1: Use case diagram of the present project.

At the end of the execution, the developer is able to inspect the results, as well as use the models (in principle, the best classified) within SANEPAR's software to optimize the functioning of the water management system.

4.2 Class Diagram

The class diagram in Figure 4.2 presents an overview of the classes that make up the system. Basically, the class KDDPipeline is the one that manages the execution of KDD steps, which are in fact executed by Source, Processor and Evaluator.

The class Source is responsible for collecting data from different sources, preprocessing them and saving the preprocessed data in the folder outputfolder at the end of its execution. Preprocessor, in turn, is responsible for transforming the data into an appropriate format to feed the model, as well as training and executing predictions on this data. The results are managed by the ProcessorResults class and saved in a folder. Finally, the evaluator takes all the results and creates graphs and tables to evaluate the best model.

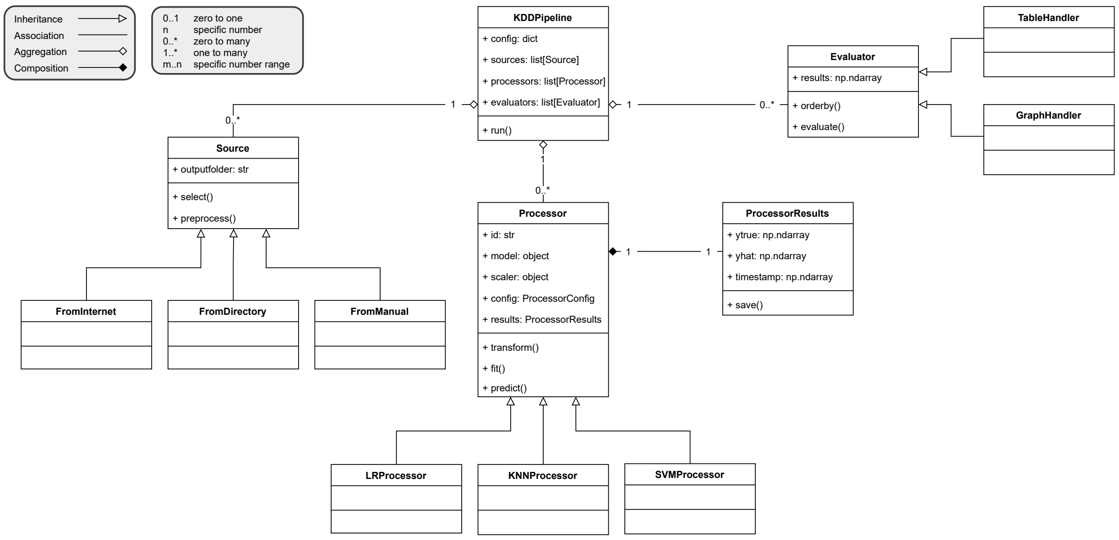

Figure 4.2: Class diagram of the present project.

This design allows the user to extend the approach by collecting data from different sources, creating new processors and evaluating them in a different way.

4.3 State Machine Diagram

The state machine diagram shown in Figure 4.3 exemplifies the possible states that the system can be in. Once started, the system simply runs the KDD steps sequentially. The “Fitting the Model” state refers to the “Data Mining” step, and “Prediction” is added for clarity.

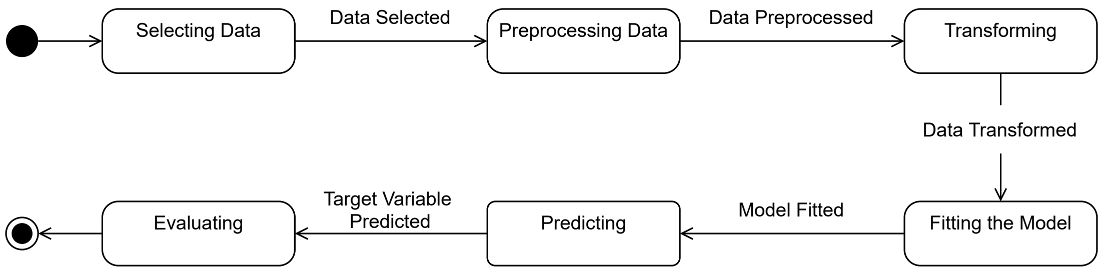

Figure 4.3: State machine diagram of the present project.

4.4 Sequence Diagram

The Figure 4.4 shows the sequence diagram assuming that all the steps are enabled in the configuration file. Since a KDDPipeline can have multiple Source, Processor and Evaluator, they are all executed in a loop. For each Source, select and preprocess methods are executed. For each Processor, transform, fit and predict are executed. Finally, ProcessorResults saves the results, which are then evaluated by each Evaluator.

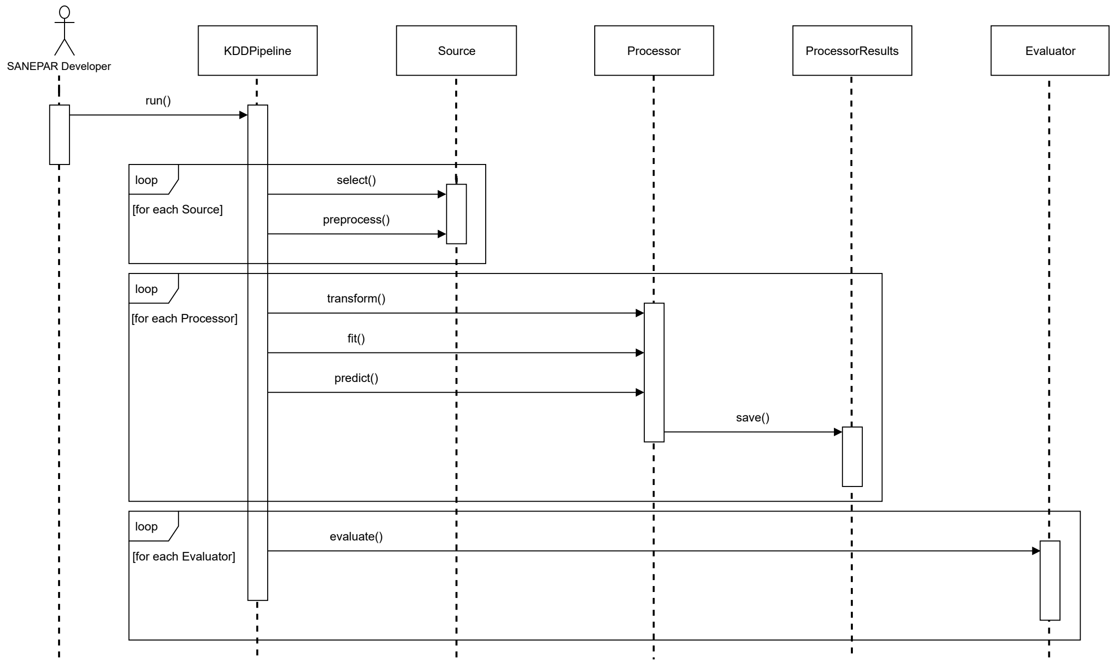

Figure 4.4: Sequence diagram of the present project.

This diagram very well shows the sequential nature of the system. Note that objects do not exchange messages with each other, each one does its job and nothing else.

5 Partial Implementation

The purpose of this chapter is to discuss the partial results of the project implementation, as well as the unit tests developed to guarantee the correct functioning of the program. Details of the proposed structure were adapted (at first, temporarily) due to the obstacles faced during the development. Anyway, it can be said that the software already has a basic structure and just needs some polishing.

The user can adapt the execution of the software through a configuration file, as shown in Listing 5.1. These settings allow you to select the KDD process steps that will be executed, change the directory where the results will be saved and select which methods and template settings will be executed. It is important to note that the options can still change and it is also necessary to add settings for the Sink class.

Listing 5.1: Default configuration file.

```
{
    "kdd": {
        "select": true,
        "preprocess": true,
        "transform": true,
        "fit": true,
        "evaluate": true
    },
    "source": {
        "raw-folder": "data/raw",
        "preprocessed-folder": "data/cache",
        "output-folder": "data"
    },
    "processor": {
        "methods": ["LR", "KNN", "SVR", "MLP"],
        "feature-set": ["FS1"],
        "validation-method": ["EW", "SW"],
        "train-window": [730],
        "test-window": [3]
    }
}
```

The implementation followed the class diagram in Figure 4.2. It was noticed that with object-oriented programming the code became more organized and modularized, facilitating its maintenance and extensibility. Despite that, it was difficult to implement the Source and Sink classes, which were not implemented modularly as proposed and a simplified implementation was made with the aim of a functional code.

The proposed software requirements are listed in Table 5.1, which indicates whether or not they have been met. There are still some tweaks to make regarding the structure of the software (which is the hardest part), but most of the remaining items are done.

Table 5.1: Software requirements in partial implementation. OK means the requirement has been met, NOK not met, and – means not applicable.

Software Requirement	Done
FR-1: User should be able to run KDD pipeline	OK
FR-2: Data that is not at hand should be downloaded from the internet	OK
FR-3: Predictions should be saved for each model	OK
FR-4: Visual representations resulting from the evaluation should be saved	OK
ER-1: User interfaces are provided in a Makefile	NOK
ER-2: The computer should be connected to the internet	–
NR-1: Data should be saved in a single file after Preprocessing task	OK
NR-2: Error metrics should be used to evaluate models	OK
NR-3: The software should be modularized	NOK
NR-4: The software should be well documented	NOK
NR-5: The software should be easy to use	OK

Thus, the focus in the final phase of the project will be on: (i) improving the Source and Sink classes (ii) finalizing the tests (iii) and other requirements such as providing a user interface through a Makefile and document the software.

6 Final Implementation

The purpose of this chapter is to discuss the final results of project implementation. All major software requirements were met and the details of the proposed structure were adapted due to the obstacles faced during development. The software can be said to have a basic (yet extensible) framework for comparing different machine learning models and the development provided useful insights for future work, such as behavioral testing for machine learning systems that were not planned but are very important.

6.1 System Features and Requirements Review

The proposed software requirements are listed in Table 6.1. The tweaks to make regarding the software structure mentioned in the previous chapter were done and therefore, all major requirements were completed. The interface through a Makefile (ER-1) was not provided, however all the commands necessary to install the requirements and run the software are described in the README. Also, although the software is well described in this document, docstrings are missing and the NR-4 is not complete either.

Table 6.1: Software requirements. FR means functional requirements, ER means external interface requirements and NR means nonfunctional requirements.

Software Requirement
FR-1: User should be able to run KDD pipeline
FR-2: Data that is not at hand should be downloaded from the internet
FR-3: Predictions should be saved for each model
FR-4: Visual representations resulting from the evaluation should be saved
~~ER-1: User interfaces are provided in a Makefile~~
ER-2: The computer should be connected to the internet
NR-1: Data should be saved in a single file after Preprocessing task
NR-2: Error metrics should be used to evaluate models
NR-3: The software should be modularized
~~NR-4: The software should be well documented~~
NR-5: The software should be easy to use

6.2 System Modeling Review

The system architecture proposed in the class diagram shown in the Figure 4.2 was slightly changed due to some obstacles faced during the development. As shown in the new class diagram in the Figure 6.1, the most noticeable changes are: the Evaluator class changed its name to Sink and the Source and Sink classes doesn't have any child class anymore. Besides that, some methods and attributes were added or changed.

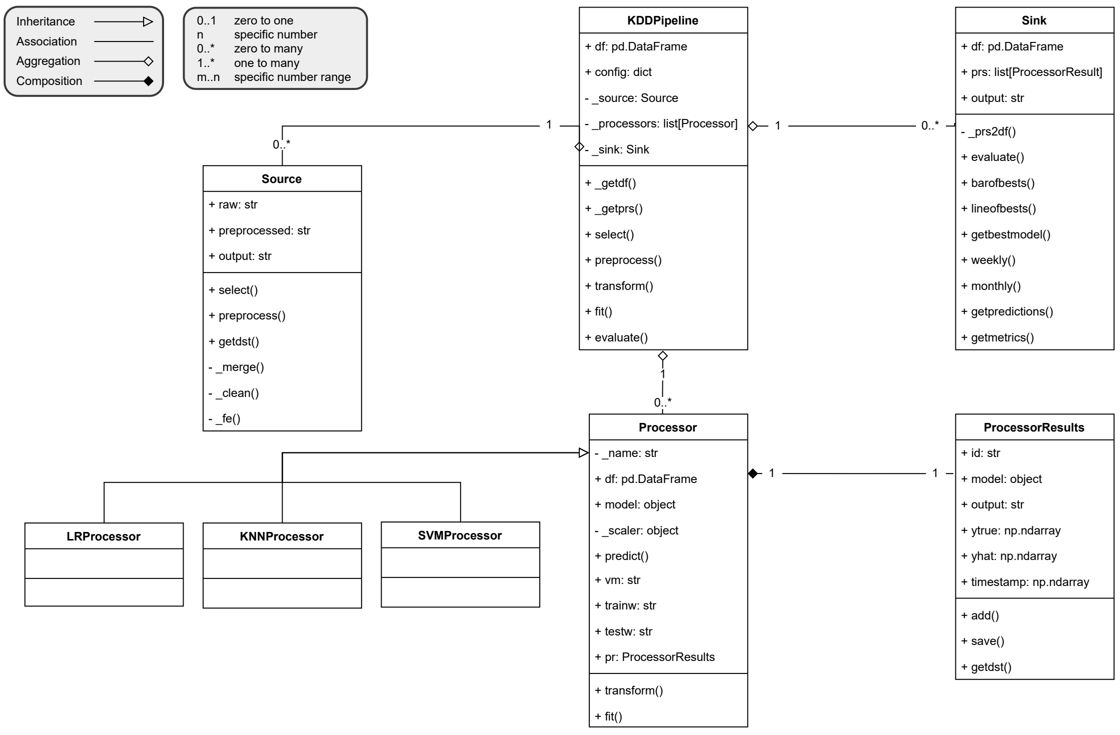

Figure 6.1: Final class diagram of the present project.

On the other hand, the sequence diagram shown in Figure 6.2 represents a more detailed execution of the software. As the user can configure the steps that will be executed through the configuration file mentioned in the previous chapter, each KDD step has a condition for execution and some of them (pre-processing and tuning) automatically get the necessary input for the next step if they are not selected. This is very important when developing a machine learning project, as we don't want to run every step (especially select and preprocess) every run, and it saves a lot of time.

Besides that, the use case diagram and state machine diagram respectively shown in Figure 4.3 and Figure 4.1 have not changed.

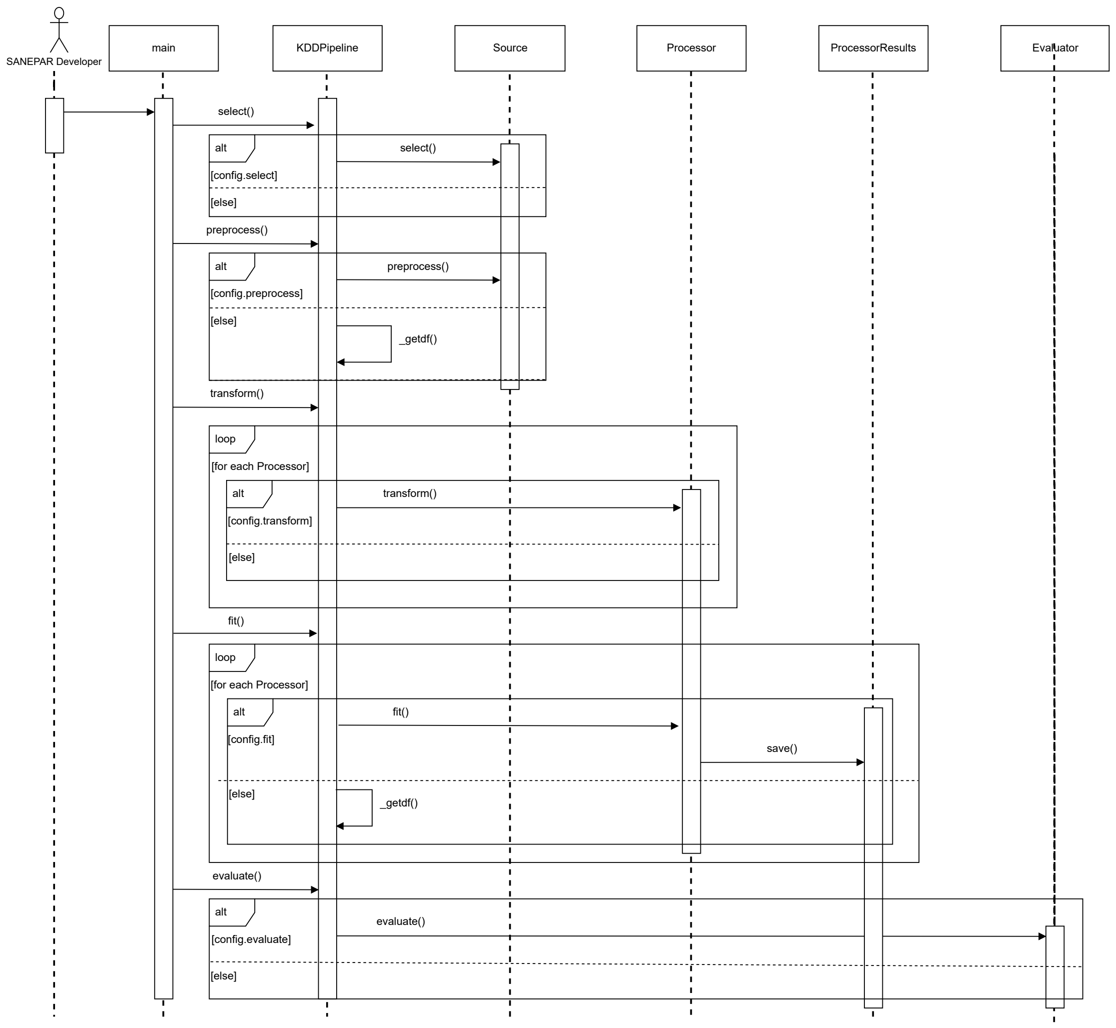

Figure 6.2: Final sequence diagram of the present project.

6.3 Software Testing

At first place, it was being developed a traditional software testing suite (for model development infrastructure), but the focus changed for a model testing suite (for trained models) after discussing with the professor. Although they were not implemented due to time issues, these tests can be divided as:

- Pre-train tests: allow us to identify bugs early on and avoid incorrect training a model, for example, by making assertions about your datasets.
- Post-train tests: use the trained model to inspect behaviors for a variety of important scenarios by interrogating the logic learned during training.

The latter was found to be a great learning experience during project development and can have a big impact during the development lifecycle. These post-training tests can be further divided into categories, where two will be highlighted and implemented in the future: directional expectation tests and minimal functionality tests.

Directional expectation tests allow us to define a set of perturbations on the input that should have a predictable effect on the output of the model. For example, it is expected that by increasing the average temperature, the output (water demand) will be greater. On the other hand, minimal functionality tests allow us to quantify model performance for specific cases found in our data. For the current project, two cases will be implemented, they are the performance of forecasts by season and by day of the week.

6.4 Results

The software provides a framework for developing and comparing different machine learning models. At the end of its execution, the results are graphically displayed to the user, who is able to determine the best method and configuration for the intended purpose, in this case, water demand forecast. Figure 6.3 shows four bar graphs with R^2, mean absolute error (MAE), mean square error (MSE) and mean square error (RMSE) of the best model of each method.

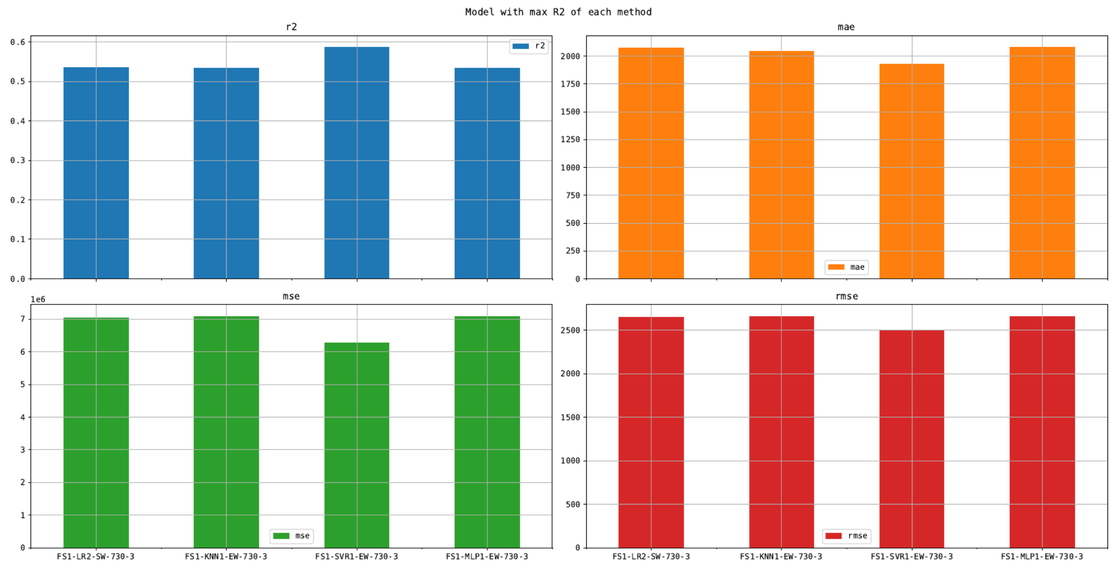

Figure 6.3: Best model of each method. The method name is identified as the second string in each label, for example, FS1-**MLP**1-EW-730-3 is the best MLP model.

The user can also view the test set prediction for the best model for each method. Figure 6.4 shows the test set prediction and actual value for the best SVR model (which currently has the highest R^2 among all other models) and Figure 6.5 displays these predictions for four random months, allowing the user to see the model's behavior in a smaller window.

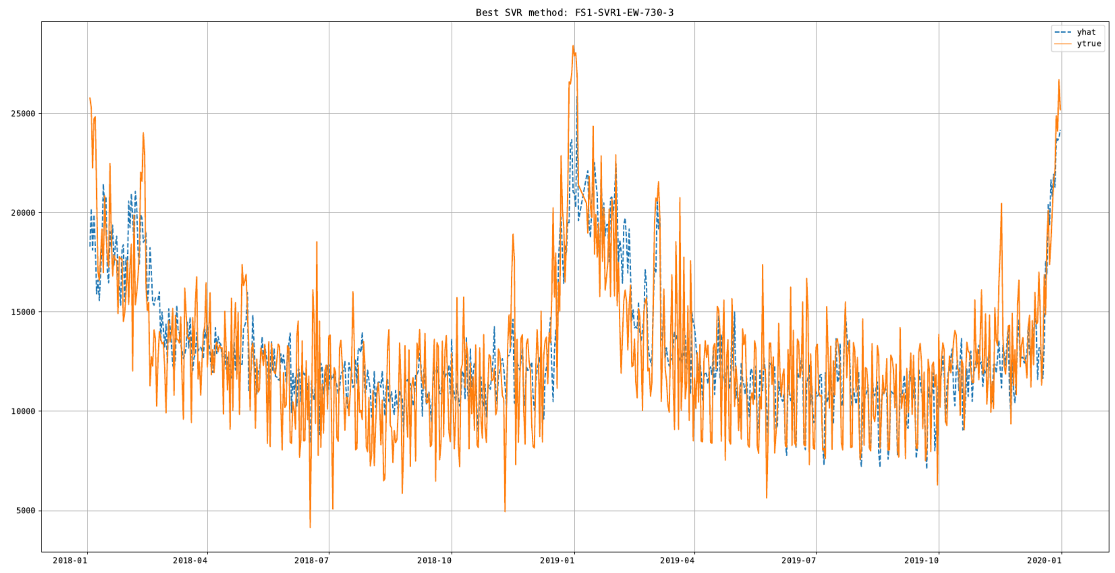

Figure 6.4: Test set forecast for the best SVR model plotted with the actual value.

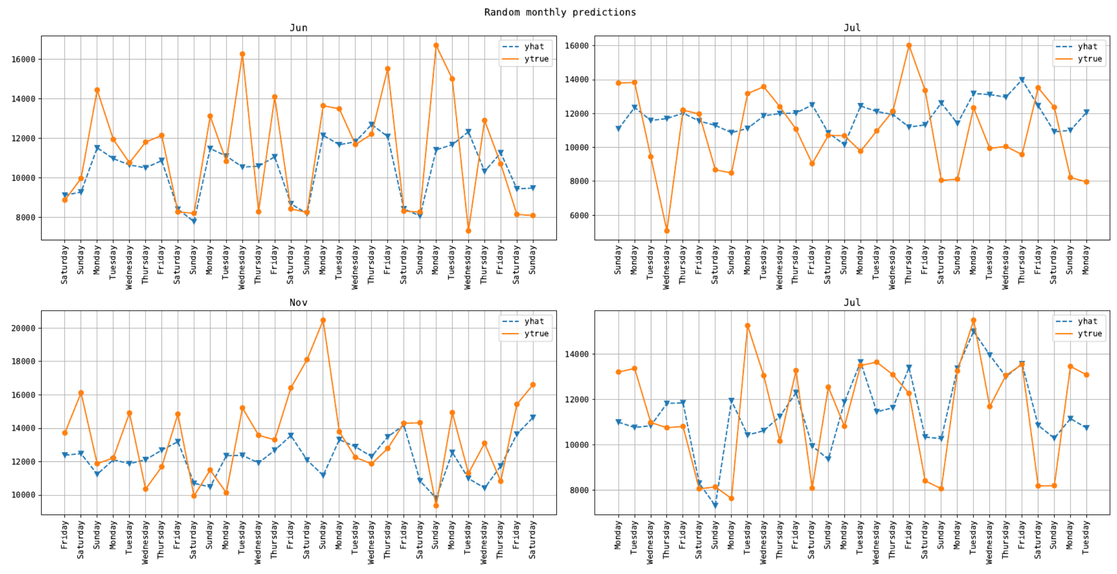

Figure 6.5: Monthly forecast of the best SVR model plotted with the actual value.

Bibliography

Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. (1996). From data mining to knowledge discovery in databases. *AI magazine*, 17(3):37.

McKinney, W. (2010). Data structures for statistical computing in python. In van der Walt, S. and Millman, J., editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Singapore, P. U. B. (2016). Managing the water distribution network with a smart water grid. *Smart Water*, 1(1):4.

UN, U. N. (2015). Transforming our world: the 2030 agenda for sustainable development. Working papers, eSocialSciences.

UN-Water (2021). Un world water development report 2021.