DATA ANALYSIS AND VISUALIZATION OF MULTIVARIATE TIME SERIES SOFTWARE

Victor C, Flor De Luz

2024-05-20

Contents

1 Introduction

The project explores, analyzes, and integrates Vue.JS and Python code with different models for clean and data completion.

2 Goals

- Complete the code by integrating different Python models into the software project.
- Data structure improvement to adapt new ML models for cleaning data and completion.
- Technical documentation of the software.

3 State of the art

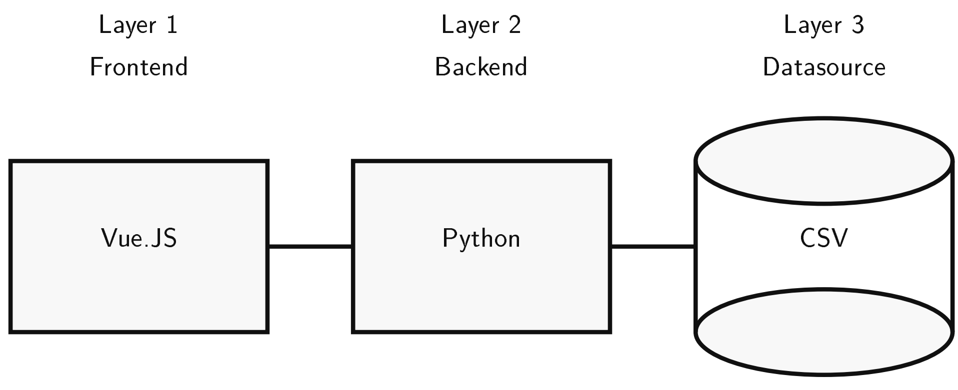
- Source code of the project(1), with documentation to install and deploy the software. The software currently reads data and graphs time series. It has options for data completion through rolling mean and kNN and partially cleaning data alternatives. It tracks for changes while navigating through the Diagram Operator interface graphically. Integrates Radial Chart for time series cycles.
- Self-documented notebook with multiple machine-learning techniques and their variants for data completion like Rolling Mean, Decision Trees, Stochastic Grading Boosting, Locally Weighted Regression, Legendre Polynomials Regression, Random Forest Regressor, k-nearest Neighbors. Includes removing features with no data at all. Computes a Dicky Fuller Stationarity Test. Automatically computes Weighted MAPE and R-Score + RMSE to detect and suggest which model fits better. Computes Autocorrelation, Fourier, and Hodrick Prescott to detect Cyclicity.

4 Pre-implementation analysis

4.1 Multilayer design

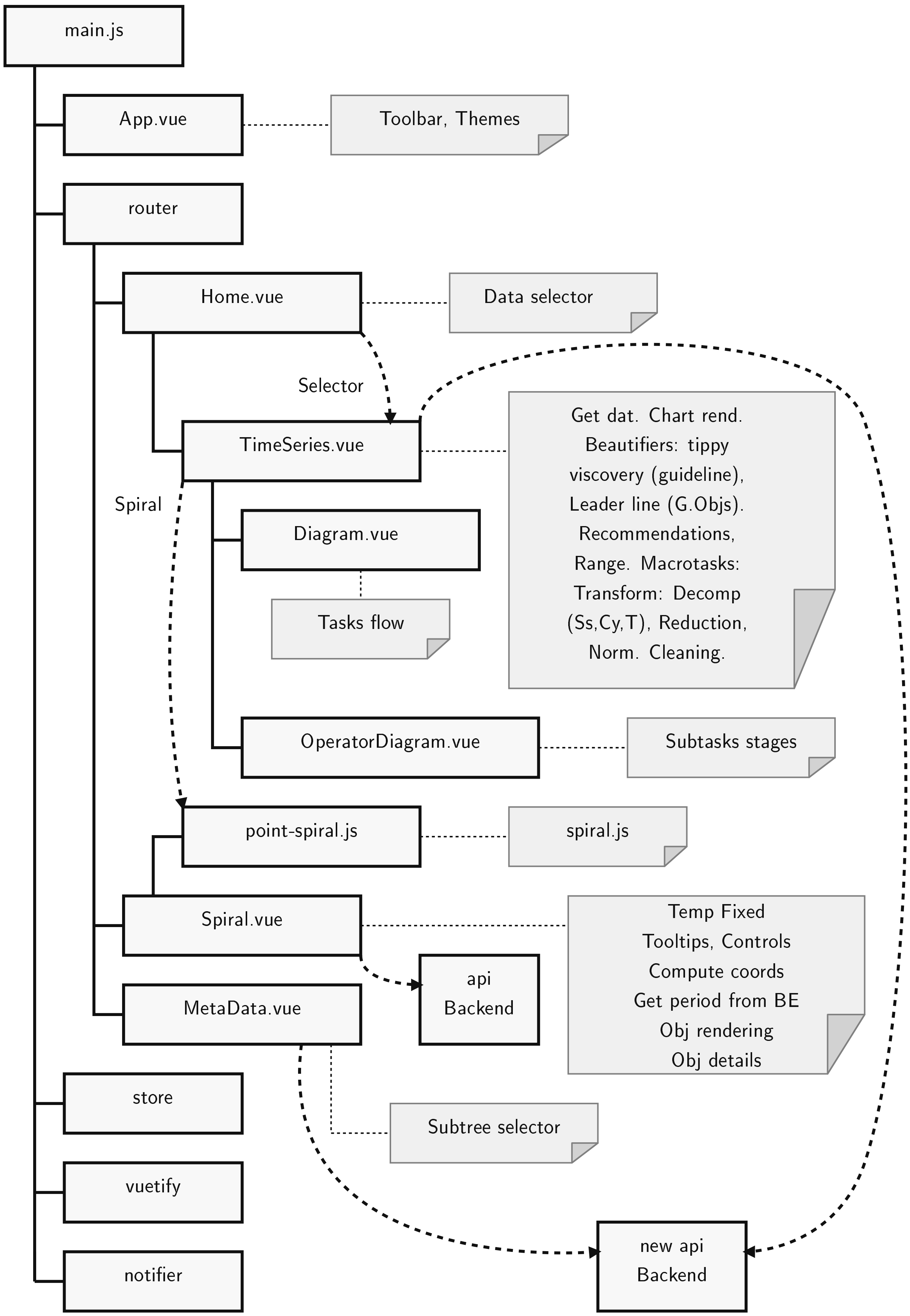
The critical aspects of the software are:

- Frontend developed in Vue.JS
- Backend developed in Python
- The data source layer primarily are CSV files [1]

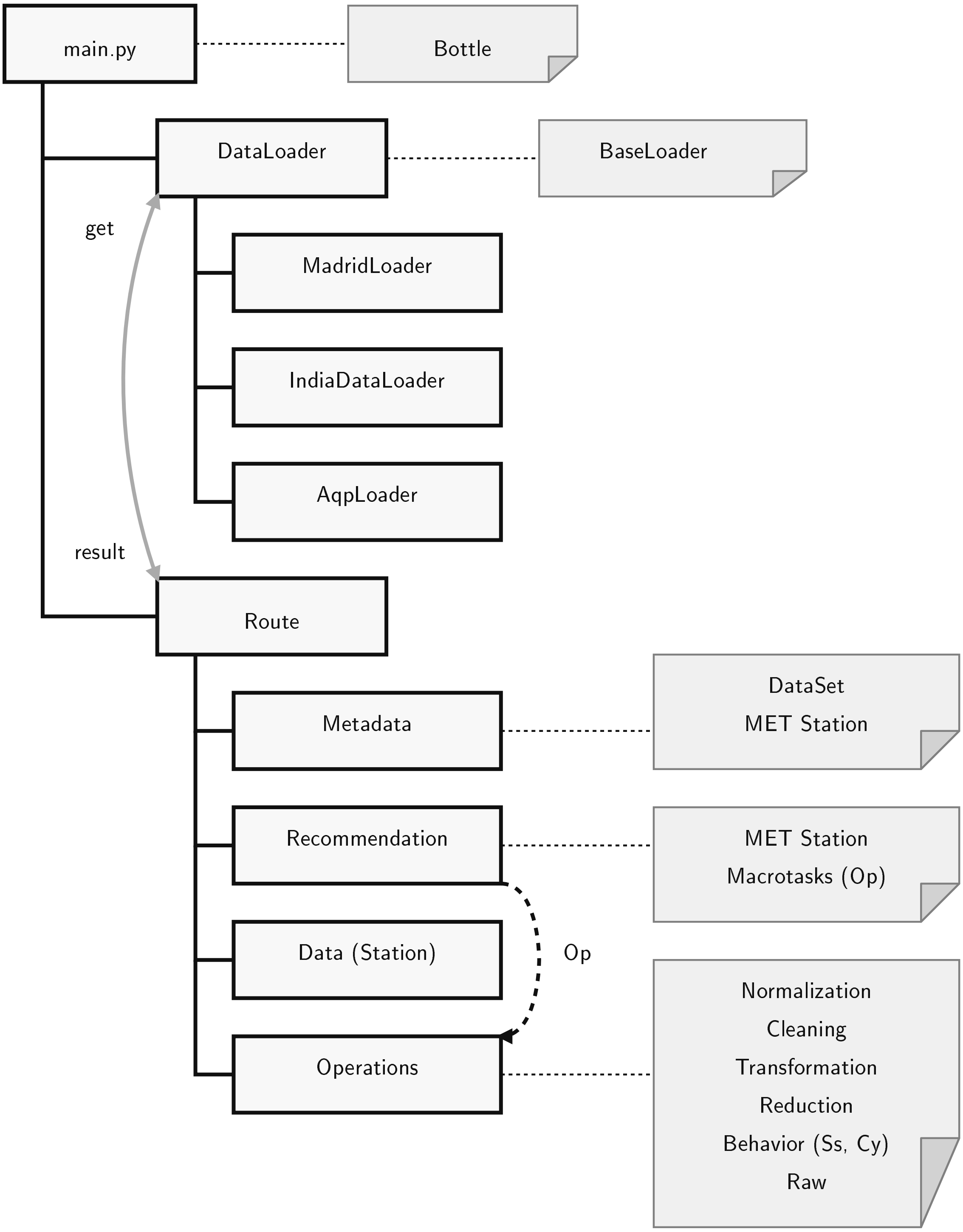


[1] HDF5 detected

4.2 Frontend specifics



4.3 Backend specifics



4.4 Detected issues

- Data structures. It requires adapting the current data structures of the program to the new dictionary of the tested and trained models.
- Exceptional cases. There are some treatments in the frontend code for specific data sources [2]. Those treatments will done in the backend.
- The Radial Diagram component [3] hardcode for temperature and precipitation. We will rewrite the code to accept different datasets and features. Additionally, it is currently consuming the old backend[4]. The old backend has to be re-implemented in the new backend[5].

[2] TimeSeries.vue, temperature, and precipitation, line 308
[3] Spiral.vue
[4] api/main.py
[5] new_api/main.py

5 Implementation

5.1 Activities summary

- Adapt the Backend Data Structure.
- Data Structure Integration. It will include saving a dictionary and allowing notebook compatibility.
- Adapt the Frontend Data Structure.
- Implement new ML models in the back end.
- Adapt the front end for new ML models.
- Frontend UI additional improvements.
- Format multivariate time series data.

5.2 Activities execution

5.2.1 Change log

Tracking changes on the program will be done by using diff to create a patch component that allows the creation of a checkpoint and registering the changes by the size of modify or created code[6]. The folder structure is:

Table 1: Change log folder structure

A	File System Structure	Description
drwx	VisWeb-AlgoritmosLimpieza.incr	Incremental Checkpoint
drwx	VisWeb-AlgoritmosLimpieza.diff	Diff/Patch Repository
drwx	VisWeb-AlgoritmosLimpieza	Development Folder
-r–	checkpoint.lisp	The program for recording changes

5.2.2 Backlog

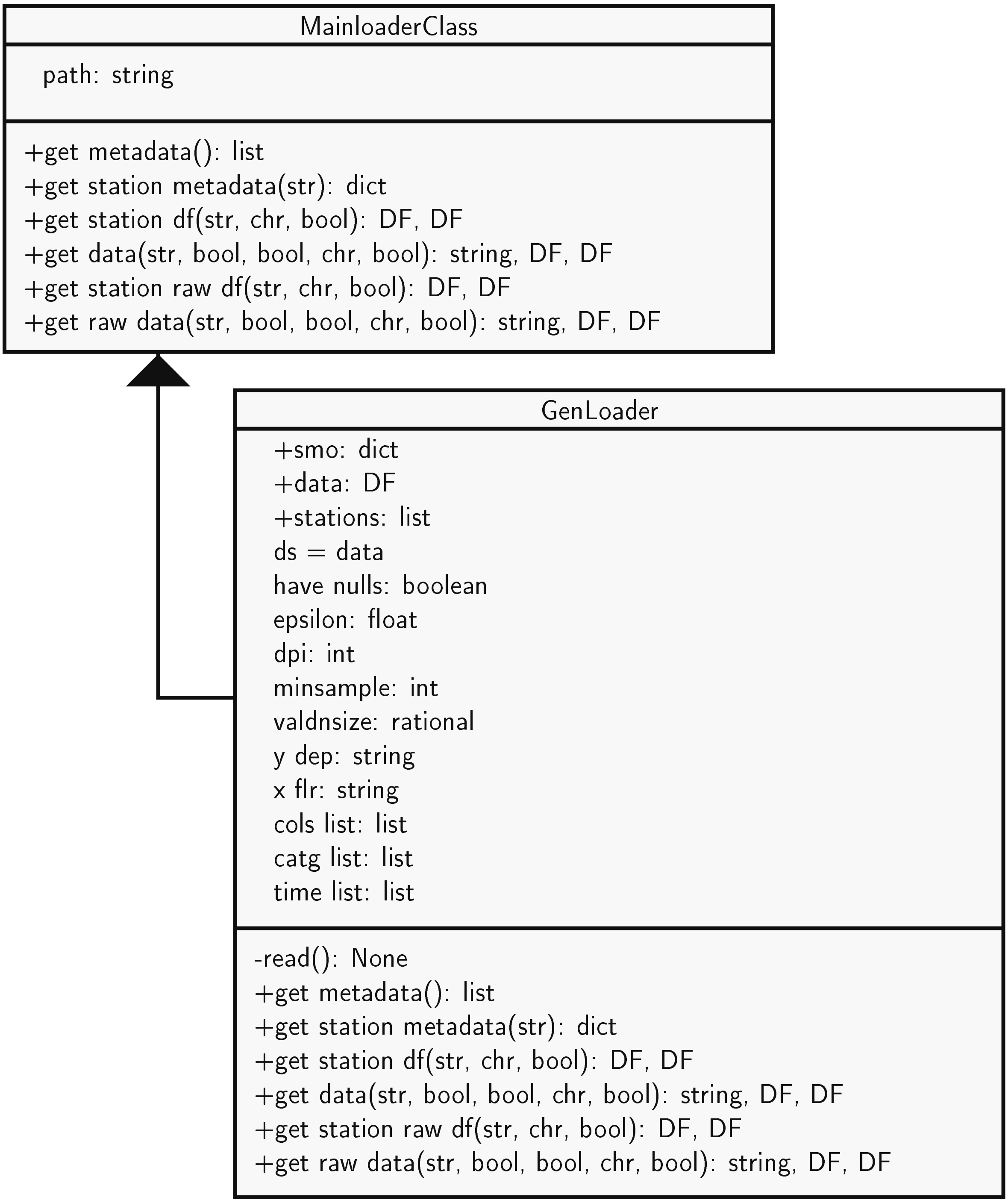
The pending activities are in the appendix backlog section. These activities correspond to changes or reviews that depend on multiple program files around the software, and their resolutions will come on the project's timeline. Bugs will gradually fixed.

5.2.3 Backend data structures

Programs adaptation in the new_api.

- Creating a new MainloaderClass and its derived GenLoader class as a generic data loader that extends MainloaderClass[7]. Methods' names remain unchanged to keep compatibility with the original code. **Created**.

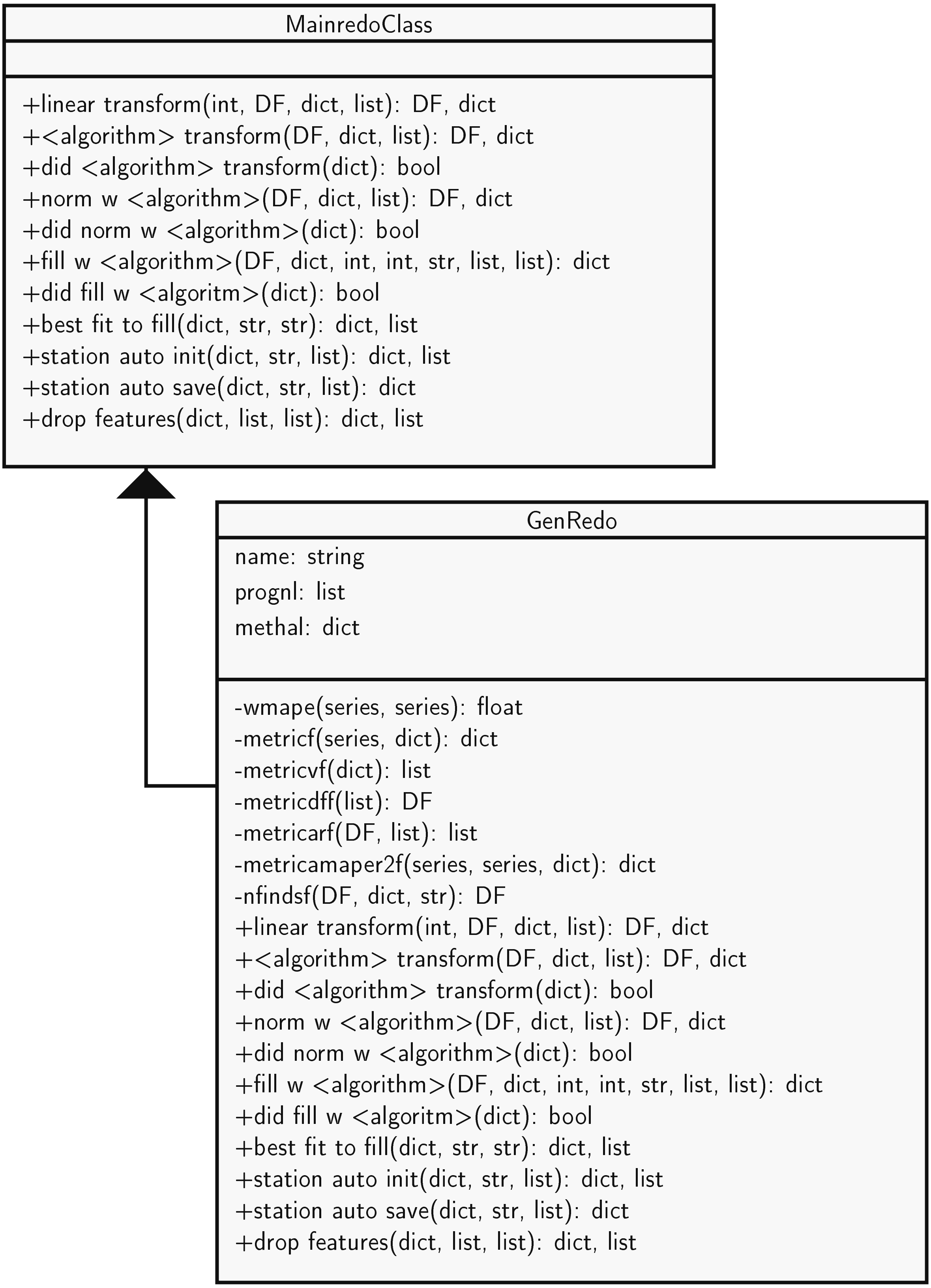
[6] Tracking log in the appendix
[7] The new Class defines the notebook's data structure



- The derived class GenLoader improves:
  - The differences in the data structures returned by some Data Loaders. Those differences crash the Python kernel. **Fixed**.
  - Some old Classes' methods trunk data to 1000 rows for interprocess communication. **Fixed**.
- Normalization will use MaxAbsScaler. Max scaler detected without considering negative values in some normalizations through the *main.py* in the new_api code. **Fixed**.
- The GenLoader-derived class allows different datasets to load. It returns a new loader object with the needed structure. It loads the data from the source when the constructor creates an instance of the class. The main program will map correctly the different datasets when the front end requires it. **Implemented**.
- The front end becomes slow with too much data. A resample by day solves the issue. Internally, it will keep the original data, and it will transform the data without resampling. The resamples are just for visualization purposes. **Fixed**.
- Code refactoring is mandatory[8]. Some functions have no real input for data observations and use a random dataset created inside the function. Other functions presented incomplete treatment. Dimensionality reduction rewritten functions (2). **Implemented**.

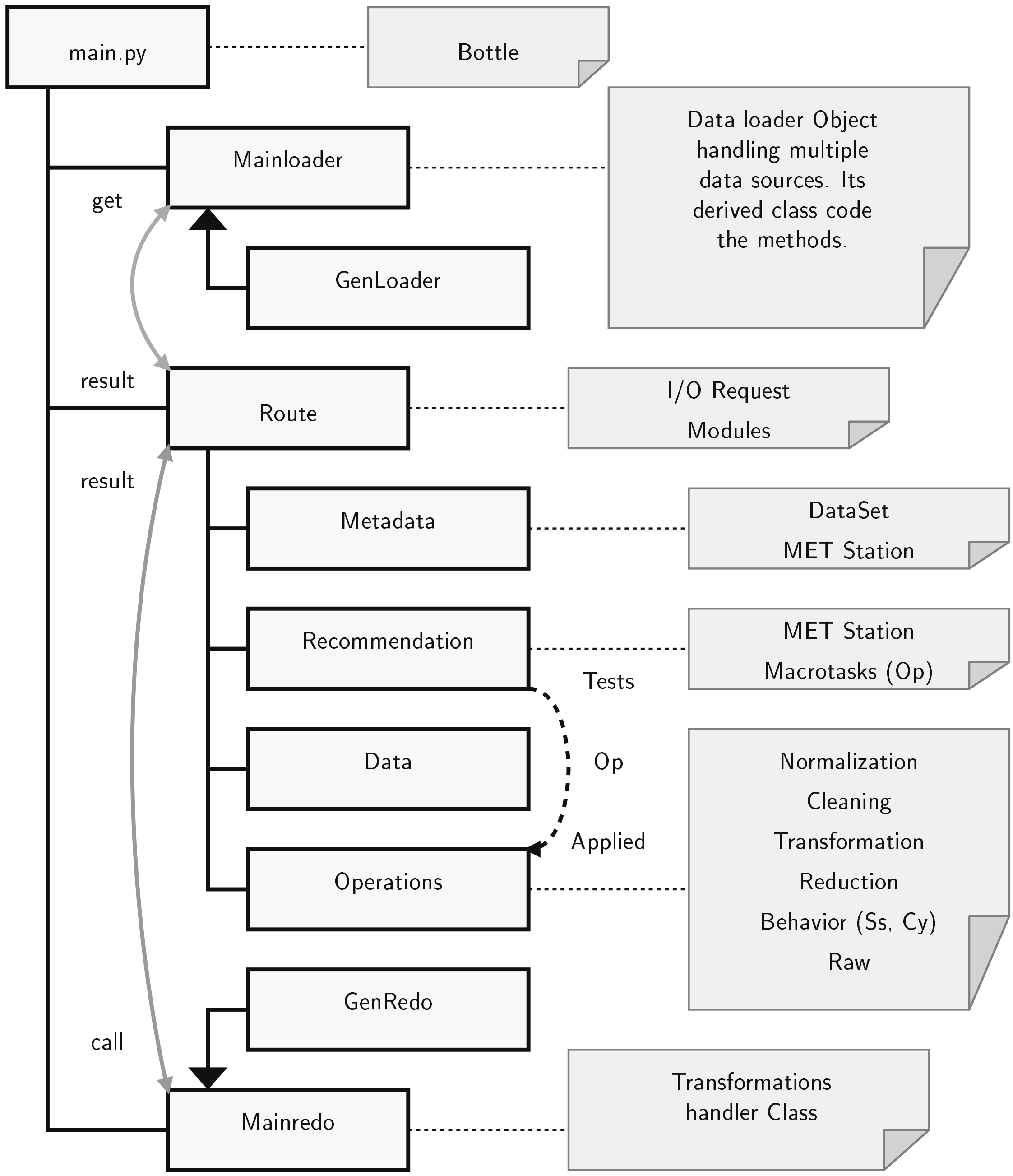
[8] Testing functions in the appendix

- Duplicated and non-relevant functions have been removed. **Fixed**.
- The transformations will be packed in the MainredoClass and its derived class GenRedo, where the methods will be different techniques to clean and complete data. **Created**.

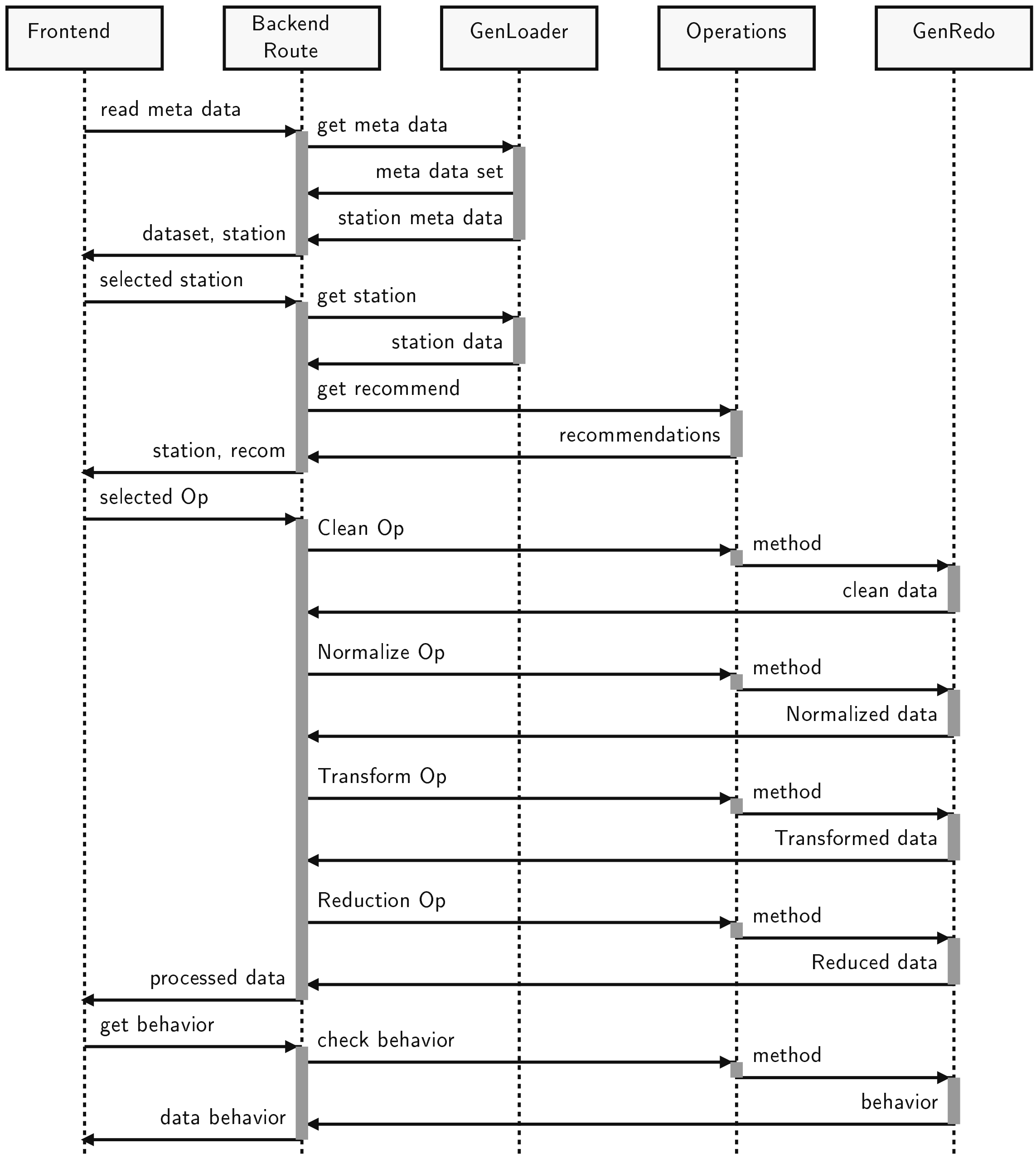


5.2.4 Backend new specifications

1. Classes and modules



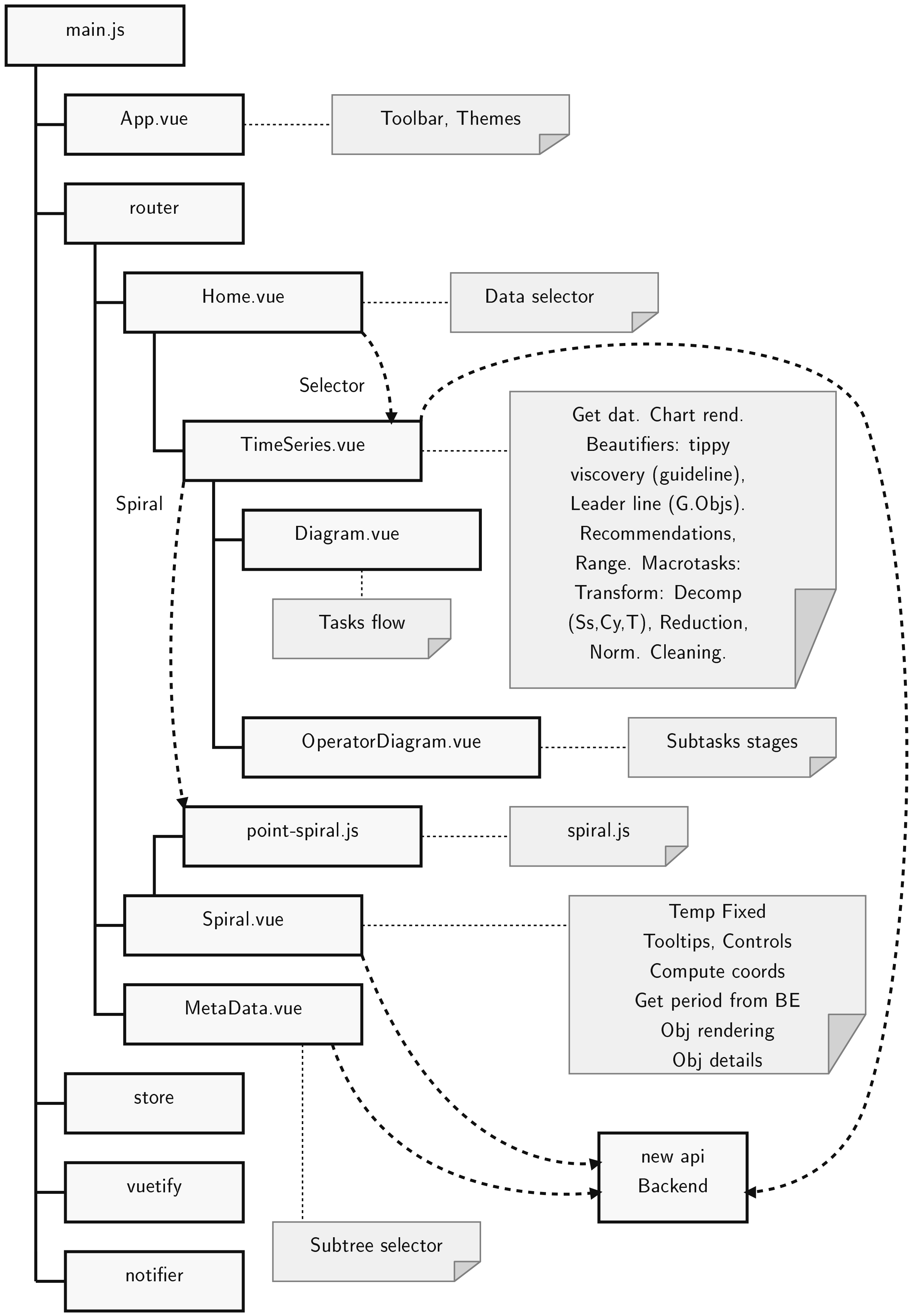
2. Sequence diagram



5.2.5 Backend new data structure and frontend integration

The initial state of the front end had two APIs. All the methods to grab data were migrated and transformed in the new API, leaving just one centralized processing Python API.

The main structure of the front end has not changed too much. The most relevant changes were in coding because the program was created for Peruvian temperature and precipitation datasets.



5.2.6 Data repository and cache

The backend will support dataset uploading. The structure has been defined on folders as an ID inside a .data repository. It will allow us to keep the data treatment progress in folders with the same ID inside a .cache repository.

6 Requirements

6.1 Software

- Python 3.12.9
- NodeJS 18.19.1
- GNU/Linux distribution with kernel 5.15.19 or higher
- Windows has not been tested, but it may work through Anaconda

6.2 Minimum Hardware

- Processor AMD64 or x86_64 architecture
- 8GB RAM
- 32GB SSD SWAP
- 64GB HD (SSD recommended)
- GPU (Optional)

7 Visual Analytics Guidance Development

7.1 Spiral diagram analysis improvement

- Compute segments and timespan for the Spiral. **Done**.
- Multiple time series integration. **Done**.
- Fix polygon coordinates when there are more than three dimensions. **Done**.

7.2 Independent section for statistics

- Statistics of the time series (size, nulls, type of columns, or dimensions). **Done**.
- Distribution type of the time series (e.g., normal). **Done**.
- Outliers. **Done**.
- Correlation matrix. **Done**.

7.3 Flow diagram improvement

- Data cleaning, Normalization and transformation, and Time-series behavior. **Done**.
- Stoppers and prereqs control between stages of the Guidance flow. **Done**.
- Network graph controller. **Done**.

8 Format Multivariate Time Series Data

The program makes data preprocessing by inspecting data and giving recommendations through a hierarchical network graph navigation (3). The steps (5) related to this are:

- Load data. Data from multiple sources can include data synchronization techniques like aggregation, disgregation, or data alignment in terms of date, creating additional missing values.
- Define appropriate data types and structures for time series.
- Inspect data for missing values, addressing them through interpolation or imputation methods (4).
- Stationarity testing with Augmented Dickey-Fuller test.
- Look for characteristics that change the model assumptions, like exponential growth, periodicities, or ciclity patterns.
- Transformation like a logarithmic or square root to stabilize variance and reduce skewness.
- Data normalization by scaling data to enhance comparability.
- Decomposition into a trend, seasonality, and residuals to isolate cyclic behavior.

9 References

1. https://github.com/flordeluz/VisWeb

2. Mahmood Al-khassaweneh, Mark Bronakowski, Esraa Al-Sharoa (2023). Multivariate and Dimensionality-Reduction-Based Machine Learning Techniques for Tumor Classification of RNA-Seq Data. Engineering, Computing and Mathematical Sciences, Lewis University, Romeoville, USA. Computer Engineering Department, Yarmouk University, Jordan. Electrical Engineering Department, Jordan University of Science and Technology, Jordan.

3. Stefan Gladisch, Heidrun Schumann, Christian Tominski. Navigation Recommendations for Exploring Hierarchical Graphs. Institute for Computer Science, University of Rostock, Germany.

4. https://ch.mathworks.com/help/econ/multivariate-time-series-data-structures.html

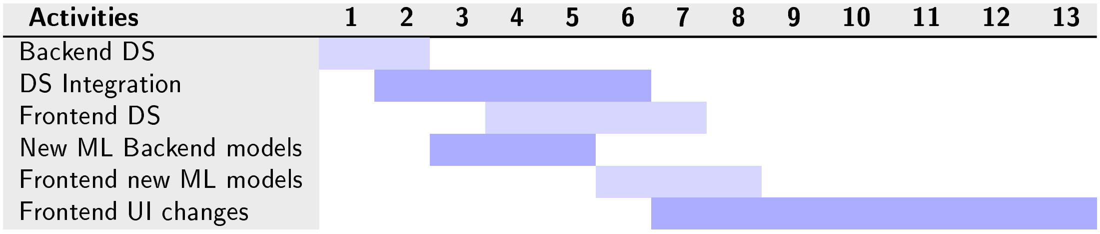
5. Can Zhou, Masami Fujiwara, William E. Grant (2016). Finding regulation among seemingly unregulated populations: a practical framework for analyzing multivariate population time series for their interactions. Springer Science Business Media New York. https://www.researchgate.net/figure/Steps-in-analyzing-multivariate-time-series-data-a-Af fig2_288179902

10 Appendix

10.1 Time Estimation Plan

- 2 months +4 backup weeks

Table 2: Activities in weeks



10.2 Testing functions

Testing functions are executed to give advice when the system is processing the signal. In the case of nulls, it uses WMAPE to prioritize the imputation algorithms; the rest are selected by the majority because each algorithm evaluates the same. When testing, more than 50% evaluate true and false otherwise. Trend algorithms do not detect; they just use decomposition to break the signal into its main components.

Table 3: Testing function modules

Module	Functions	Hint
Nulls imputation	null_values_data(DF): bool	Nulls detection
	fill_w_meanmedian(*dict)	Rolling mean
	fill_w_decisiontree(*dict)	Decision tree regressor
	fill_w_gradientboosting(*dict)	Gradient boosting
	fill_w_locallyweighted(*dict)	Locally weighted regression
	fill_w_legendre(*dict)	Legendre polynomials
	fill_w_randomforest(*dict)	Random forest regressor
	fill_w_kneighbors(*dict)	kNN regression
Cleaning	obtener_ruido_de(DF, int): bool	Standard deviation of noise
	obtener_ruido_cv(DF, float): bool	Noise covariance Kalman filter
	obtener_outlier_iqr(DF): bool	Interquartile range
	obtener_outlier_zscore(DF, int): bool	Z-score
	obtener_outlier_grubbs(DF, float): bool	Grubbs
Normalization	obtener_no_patrones_estacionalidad(DF, int): bool	Variance and noise
	obtener_distribucion_conocida(DF): bool	Deviation and log-normal
Transformation	obtener_no_estacionariedad_adf(DF, float): bool	Augmented Dickey-Fuller
	obtener_no_estacionariedad_kpss(DF, float): bool	KPSS
	obtener_comportamiento_persistente_hurst(DF): bool	Hurst
Reduction	verificar_correlacion_pearson(DF, float): bool	Pearson correlation
	verificar_correlacion_spearman(DF, float): bool	Spearman correlation
	verificar_correlacion_kendall(DF, float): bool	Kendall correlation
	check_multicollinearity(DF, int): bool	Multicollinearity test
	check_dimensionality_reduction_pca(DF, float): bool	PCA
	check_dimensionality_reduction_fa(DF, int, float): bool	Factor Analysis
Decomposition	seasonality_detection(Series, Array): LOB	Seasonal decompose
	trend_detection(Series, Array): LOB	Seasonal decompose trending
	noise_detection(Series, Array): LOB	Seasonal decompose residuals

10.3 Dictionary

The dictionary is dynamic, it grows at any algorithm or action made. Data is mostly held on Dataframes. Classes can retain information on the object like testing functions described in the previous subsection. The function contains the signal.

The following is an excerpt from the dictionary. It repeats for each loader class.

```
loaders (dict): Class type, one per dataset
├── aqp: GenLoader
├── brasil: GenLoader
├── btc: GenLoader
└── chiguata: GenLoader
```

```
│
├── india: GenLoader
└── madrid: GenLoader

aqp: Main dataset
├── RM (dict): Imputation algorithm applied
│   └── MAJES (dict): Group or substation
│       ├── PPT: function (Dimension)
│       ├── TNM: function (Dimension)
│       └── TXM: function (Dimension)
├── cache (dict)
├── full: DataFrame: Full group or substation-treated data
├── iqr: DataFrame: An entry of data for each executed algorithm
├── raw: DataFrame: Full group or substations unchanged data
└── redo (dict)
    ├── fill (dict)
    │   ├── decisiontree: bool
    │   ├── gradientboosting: bool
    │   ├── kneighbors: bool
    │   ├── legendre: bool
    │   ├── locallyweighted: bool
    │   ├── meanmedian: bool
    │   └── randomforest: bool
    ├── norm (dict)
    │   ├── maxabs: bool
    │   ├── minmax: bool
    │   ├── robust: bool
    │   └── standard: bool
    ├── outliers (dict)
    │   ├── iqr: bool
    │   └── sdv: bool
    └── transform (dict)
        ├── diff: bool
        ├── linear: bool
        ├── log: bool
        ├── quadratic: bool
        └── sqrt: bool
```

10.4 Network coloring

Color attributes are applied with information that comes from the backend as arrays.

The coloring algorithm works as each node is gray by default, then each node name that comes in the subprocess array overwrites as red then adjacent nodes on the right are colored green, and left or parents are colored red. If an array exception shows up it replaces the red node with orange. The activities array carries information about when a dialog will be launched if the user picks the activity. The array made-path carries the executed actions and allows the algorithm to overwrite the colors of the nodes as blue, the coloring process follows the graph adjacency theory.

Trace:

Stage 1:

```
[ Algorithms priority ]:
Array [ "Rolling Mean" ]

[ Subprocesses ]:
Array [ "Clean", "Nulls" ]
```

Stage 2:

```
[ Made Path ]:
Array(3) [ "Clean", "Nulls", "Rolling Mean" ]

[ Subprocesses ]:
Array [ "Clean", "Outliers" ]
```

Stage 3:

```
[ Made Path ]:
```

```
Array(6) [ "Clean", "Nulls", "Rolling Mean", "Clean", "Outliers",
"Interquartile Range" ]

[ Subprocesses ]:
Array [ "DimRed" ]

[ Activities ]:
Array(3) [ "Multicollinearity Dim.Reduction=['PPT']", "PCA
Dim.Reduction", "FA Dim.Reduction=['PPT']" ]
```

Stage 4:

```
[ Made Path ]:
Array(8) [ "Clean", "Nulls", "Rolling Mean", "Clean", "Outliers",
"Interquartile Range", "DimRed", "Factor Analysis" ]

[ Subprocesses ]:
Array [ "Analysis" ]
```

10.5 Change Log

Table 4: List of patches by timestamp

A	Size	M	D	H	Patch
-rw-r–r–	918794	May	22	01:38	20240522_013818.diff
-rw-r–r–	32116	May	24	01:39	20240524_013931.diff
-rw-r–r–	55590	May	24	18:17	20240524_181739.diff
-rw-r–r–	40553	May	27	01:09	20240527_010932.diff
-rw-r–r–	64068	May	28	01:45	20240528_014458.diff
-rw-r–r–	160383	Jun	2	03:04	20240602_030410.diff
-rw-r–r–	110627	Jun	4	02:45	20240604_024500.diff
-rw-r–r–	63591	Jun	5	00:46	20240605_004621.diff
-rw-r–r–	68193	Jun	6	02:31	20240606_023122.diff
-rw-r–r–	33434	Jun	7	03:04	20240607_030453.diff
-rw-r–r–	91512	Jun	8	01:59	20240608_015902.diff
-rw-r–r–	76847	Jun	10	18:28	20240610_182855.diff
-rw-r–r–	78312	Jun	16	01:54	20240616_015442.diff
-rw-r–r–	24940	Jun	17	02:10	20240617_021030.diff
-rw-r–r–	73776	Jun	18	01:05	20240618_010542.diff
-rw-r–r–	27856	Jun	18	10:28	20240618_102832.diff
-rw-r–r–	138469	Jun	25	02:45	20240625_024532.diff
-rw-r–r–	65443	Jun	26	02:59	20240626_025901.diff
-rw-r–r–	56919	Jun	27	00:56	20240627_005611.diff
-rw-r–r–	24307	Jul	1	00:37	20240701_003738.diff
-rw-r–r–	30011	Jul	3	16:51	20240703_165116.diff
-rw-r–r–	91786	Jul	7	01:55	20240707_015505.diff
-rw-r–r–	46989	Jul	9	01:33	20240709_013313.diff
-rw-r–r–	96392	Jul	12	01:51	20240712_015153.diff
-rw-r–r–	231634	Jul	17	01:38	20240717_013822.diff
-rw-rw-r–	1025353	Jan	5	16:25	20250105_162546.diff
-rw-rw-r–	52880	Jan	20	12:08	20250120_120815.diff
-rw-rw-r–	108172	Jan	21	23:19	20250121_231951.diff
-rw-rw-r–	48118	Jan	29	01:22	20250129_012158.diff
-rw-rw-r–	12685	Jan	30	10:32	20250130_103221.diff
-rw-rw-r–	36912	Feb	2	00:08	20250202_000816.diff
-rw-rw-r–	115526	Feb	10	09:59	20250210_095953.diff

10.6 Backlog

Table 5: List of Backlog activities

Activity	Status
Some parts of the code are setting -1 to complete null values. It needs a review of the data with negative values like temperature. **Resolution:** Used for initial visualization purposes.	*Closed*
View action buttons have to be reviewed in the front end. There are buttons in the tree view requesting labels instead of the key data used to look for. **Resolution:** Remove action buttons on non-relevant leaf labels.	*Closed*
Integrate data and cache repositories with the front end. **Resolution:** Implemented.	*Closed*
Does get-raw-data() revert .ds to .smo["raw"] when invoked in the frontend? if so, uncomment the referred line in the get-raw-data(). **Resolution:** We are using different structures inside .smo.	*Closed*
For large datasets the prediction time of null values pre-evaluation goes like: decisiontree : too slow, > 2 mins kneighbors : too slow, > 2 mins gradientboosting: slow, > 1 min < 2 mins randomforest : medium, > 20 secs < 1 min meanmedian : regular, <= 20 secs locallyweighted : fast, <= 6 secs legendre : very fast, <= 2 secs **Resolution:** For large datasets predict with medium to fast algos. **Author's note:** While testing large dataset stations of Madrid's data, KNN gave better predictions. It is too risky to exclude slower algos in the early stage of nulls' pre-evaluation.	*Unsolved*