

A Hands-on Tutorial on Time Series Imputation with ImputeGAP

Quentin Nater & Mourad Khayati

Hands-on Tutorial @KDD'25, Toronto (Canada)

<https://imputegap-tutorials.github.io/KDD-2025>

<https://imputegap.readthedocs.io>
v1.1.1 with support of LLMs

July 31, 2025



UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Schedule

- ▶ Section 1: Introduction and Overview (20 min)
- ▶ Section 2: ImputeGAP Features (10 min)
- ▶ Section 3: Imputation Pipeline Synthesis (60 min)
- ▶ Break (15 min)
- ▶ Section 4: Downstream Analysis (20 min)
- ▶ Section 5: Imputation Explainability (20 min)
- ▶ Section 6: Library Extension (20 min)

Presenters

► Mourad Khayati

- Ph.D. from University of Zürich
- Senior Lecturer at the University of Fribourg (Switzerland)
- Time series analytics, data cleaning, missing data imputation, Time Series Database Systems
- VLDB 2020 Reproducibility Award, KDD 2025 Outstanding Reviewer Award



► Quentin Nater

- PhD student at the University of Fribourg (Switzerland) since 01/2025
- Web of Things, missing data imputation, multimodal learning
- Fantasy writer



Current Section

Introduction and Overview

ImputeGAP Features

Pipeline Synthesis

Downstream Analysis

Explainability

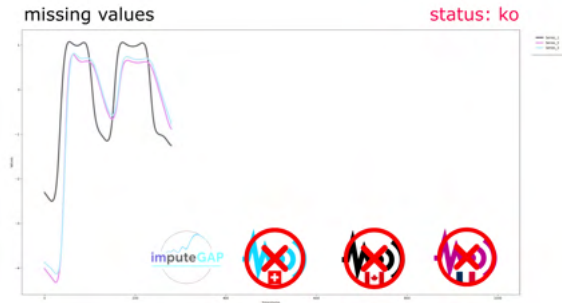
Library Extension

Data with Missing Values

- ▶ Combining data from multiple sources can yield missing entries
- ▶ Some stocks trade infrequently, so price updates may be missing on certain days
- ▶ Technical glitches or downtime in data collection systems
- ▶ During preprocessing, some data might be excluded if it is not worth being retained
- ▶ Data can be withheld due to privacy or proprietary concerns
- ▶ Organizations might not record all the statistical data
- ▶ Participants in a survey do not answer all the questions

Missing Values in Time Series

- ▶ Transmission and hardware problems often occur:
 - ▶ Intermittent connectivity
 - ▶ Power outages
 - ▶ Device failure/calibration drift
 - ▶ External interference



Missingness Statistics

- ▶ Intel-Berkeley Research Lab dataset is missing about 40% (source: A Pipelined Framework for Online Cleaning of Sensor Data Streams, Berkeley technical report, 2005).
- ▶ The UCI repository contains datasets with up to 20% missing data (source: A Survey on Missing Data in Machine Learning, J. Big Data, 2021).
- ▶ MIMIC-IV Clinical Database has up to 80% missing observations (source: Recurrent Neural Networks for Multivariate Time Series with Missing Values, Scientific Reports, 2018).
- ▶ Up to 49% data loss in wearable monitoring (source: Data Quality Evaluation in Wearable Monitoring, Scientific Reports, 2022).

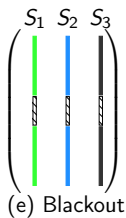
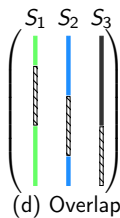
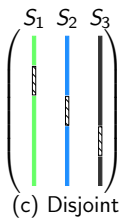
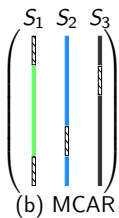
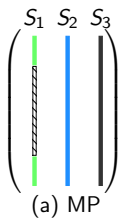
Why Imputation?

- ▶ Data management systems and models assume that the data is complete.
- ▶ Time Series Database Systems (TSDBs)¹ have begun to incorporate native support for missing value imputation.
- ▶ Ignoring missing values has a detrimental impact on time series tasks:
 - ▶ ↓ 97% AVG F1 in classification
 - ▶ ↓ 55% AVG RMSE in forecasting

¹khelifati A, Khayati M., Difallah D., Dignös A., and Cudré-Mauroux P.: "TSM-Bench: Benchmarking Time Series Database Systems for Monitoring Applications", PVLDB'23

Imputation Caveats

- ▶ Imputing time series is challenging:
 - ▶ temporal dimension
 - ▶ high-variability data
 - ▶ various features
 - ▶ blocks of consecutive values
- ▶ Wide palette of missingness patterns



Current Section

Introduction and Overview

ImputeGAP Features

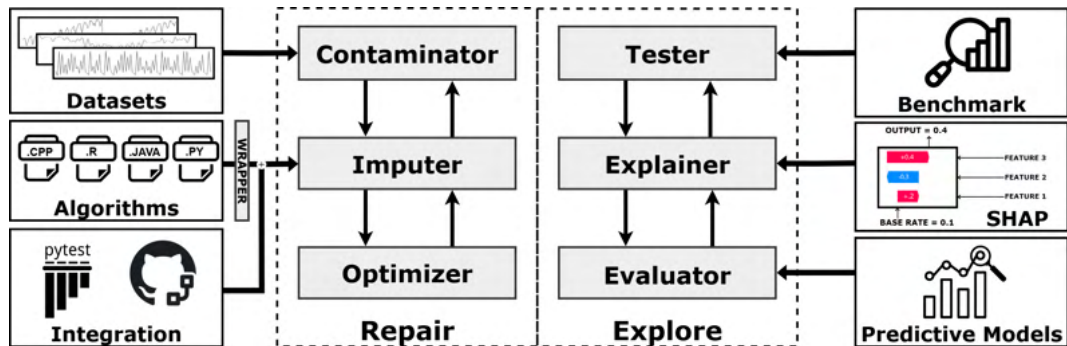
Pipeline Synthesis

Downstream Analysis

Explainability

Library Extension

Python Library

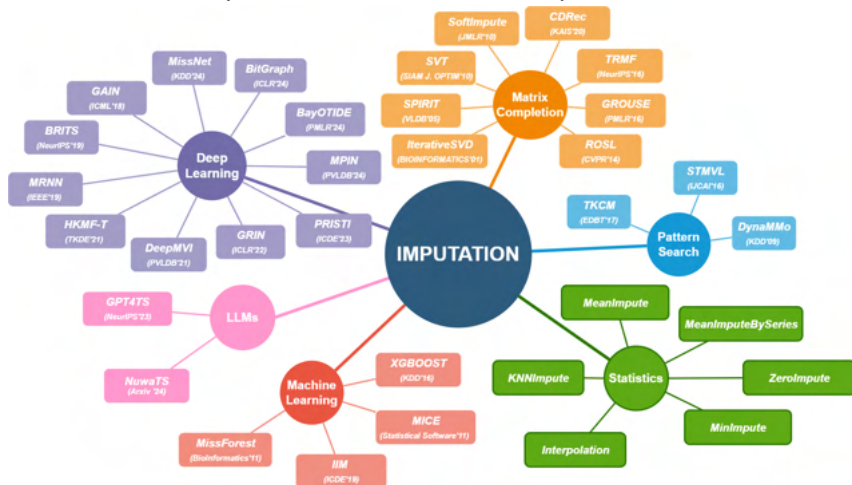


Comparison with Existing Libraries

- ▶ Other imputation libraries for time series imputation exist, e.g., PyPOTS, CleanTS, Amelia II, and sktime.
- ▶ Limited coverage of imputation algorithms, missingness patterns, and datasets.
- ▶ Do not support the explainability of the results.
- ▶ Lack of downstream analysis.
- ▶ Limited benchmarking options.
- ▶ Lack of integration module.

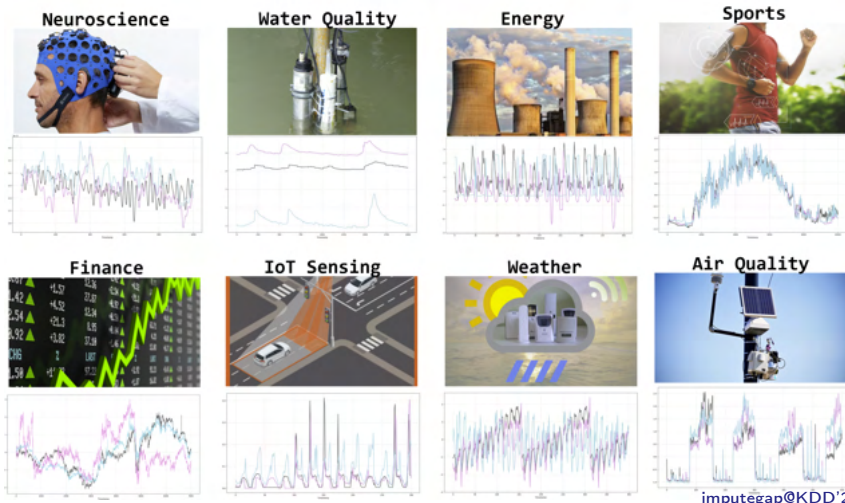
Available Algorithms

- ▶ 34 off-the-shelf imputation algorithms.
- ▶ Six families: matrix completion, ML, DL, statistics, pattern search, and LLMs.



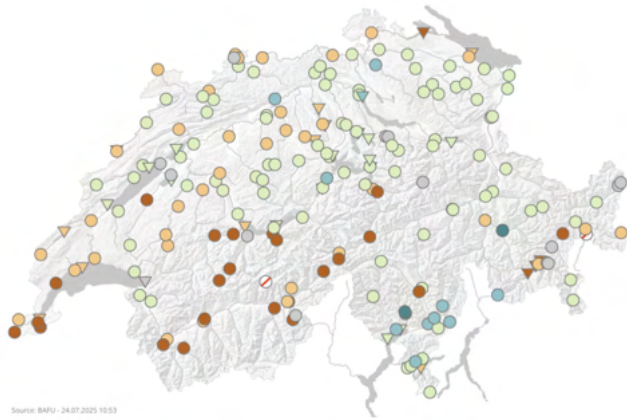
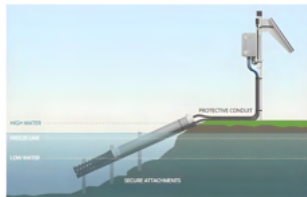
Available Datasets

- ▶ 17 ready-to-deploy time series datasets.
- ▶ 8 real-world applications.



Water Quality

- ▶ BAFU¹ deploys sensors to monitor the water quality in Swiss rivers.
- ▶ Multivariate time series: discharge, level, pH, temperature, etc.



Source: BAFU - 24.07.2025 10:53

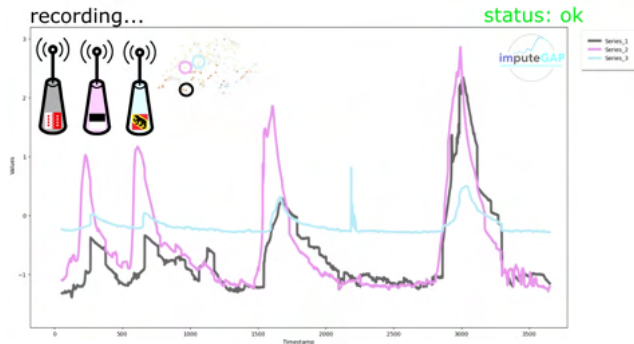
¹ Federal Office for the Environment in Switzerland

BAFU Dataset

- ▶ Monitoring of river levels across three regions, but the sensors suddenly stopped transmitting data.
- ▶ The missing data poses safety risks.



Viège (VS) - 21.06.2024



Current Section

Introduction and Overview

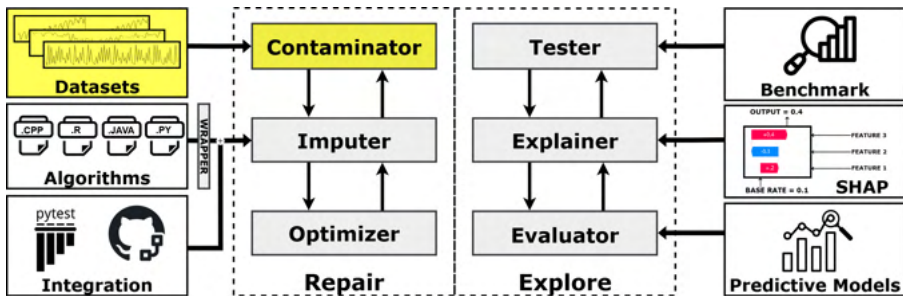
ImputeGAP Features

Pipeline Synthesis

Downstream Analysis

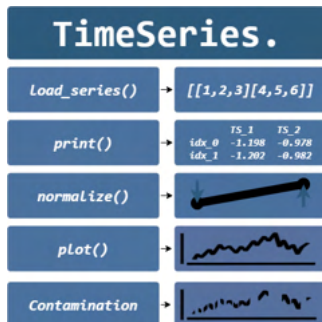
Explainability

Library Extension



One Object to Rule Them All

- ▶ A single object to manage the entire time series lifecycle, from loading to evaluation.



Note

- Input: `file`, `str`, `pathlib.Path`, `list`
- Output: `numpy.ndarray`
- Instance variable: `TimeSeries.data`

```
1 from imputeGap.recovery.manager import TimeSeries
2 ts = TimeSeries()
```

Data Loader

Loads time series data from a file or predefined dataset.

```
load_series(data, nbr_series=None, nbr_val=None, header=False, replace_nan=False,  
            verbose=True) \[source\]
```

```
1 # load from built-in datasets  
2 ts.load_series(utils.search_path("eeg-alcohol"))  
3  
4 # load your dataset  
5 ts.load_series("../impute-gap/datasets/kdd.txt")  
6  
7 # load as a matrix  
8 ts.import_matrix([[1,2,3],[4,5,6]])  
9  
10 # print your ts  
11 ts.print(nbr_series=3, nbr_val=20)
```

TimeSeries.

load_series() → `[[1,2,3][4,5,6]]`

print() →

	TS_1	TS_2
ts_0	1.158	0.978
ts_1	1.202	0.982

normalize() → 

plot() → 

Contamination → 

Note

Please ensure that your input data satisfies the following format:

- Columns are the series' values
- Column separator: empty space
- Row separator: newline
- Missing values are NaNs

Normalization

Normalize the time series data with one of the supported techniques, such as Z-Score or Min-Max.

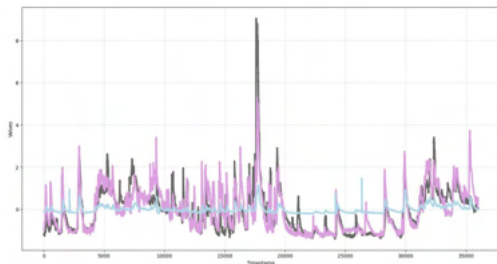
```
normalize(normalizer='z_score', verbose=True) ¶
```

[\[source\]](#)

TimeSeries.	
load_series()	→ [[1, 2, 3], [4, 5, 6]]
print()	→ TS_1 TS_2 TS_1 TS_2 TS_1 TS_2
normalize()	→ 
plot()	→ 
Continuation	→ 

► Z-Score

```
1 ts.normalize(normalizer="z_score")
```



► Min-Max

```
1 ts.normalize(normalizer="min_max")
```



Plotting

- Plot the time series: original, contaminated, and imputed.

```
plot(input_data, incomp_data=None, recov_data=None, nbr_series=None, nbr_val=None,  
     series_range=None, subplot=False, size=(16, 8), algorithm=None,  
     save_path='./impute/assets', cont_rate=None, display=True, verbose=True) \[source\]
```

Parameters

input_data: *numpy.ndarray*

The original time series data without contamination.

incomp_data: *numpy.ndarray*, optional

The contaminated time series data.

recov_data: *numpy.ndarray*, optional

The imputed time series data.

nbr_series: *int*, optional

The maximum number of series to plot.

nbr_val: *int*, optional

The maximum number of values per series to plot.

series_range: *int*, optional

The index of a specific series to plot. If set, only this series will be plotted.

subplot: *bool*, optional

Print one time series by subplot or all in the same plot.

size: *tuple*, optional

Size of the plot in inches. Default is (16, 8).

algorithm: *str*, optional

Name of the algorithm used for imputation.

save_path: *str*, optional

Path to save the plot locally.

cont_rate: *str*, optional

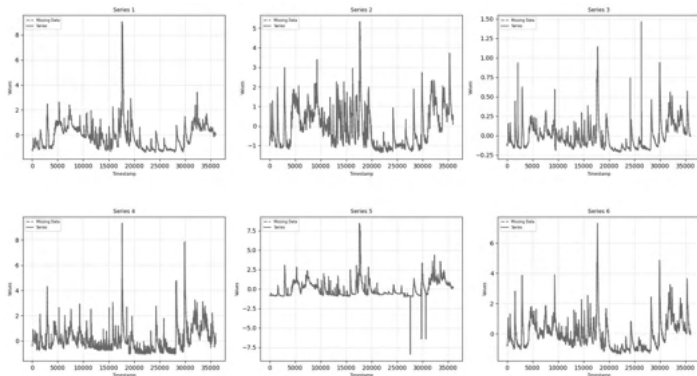
Percentage of contamination in each series to plot.

display: *bool*, optional

Whether to display the plot. Default is True.

verbose: *bool*, optional

Whether to display the plot information. Default is True.



```
1 ts.plot(ts.data, ts_m, nbr_series=6, nbr_val=36000, subplot=True)
```

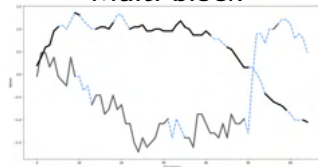
TimeSeries.	
<code>load_series()</code>	<code>[1, 5, 6, 9, 5, 6]</code>
<code>print()</code>	<pre>TS: 1 TS: 2 len: 6 len: 6 min: 1 min: 1 max: 6 max: 6</pre>
<code>normal_line()</code>	
<code>plot()</code>	
<code>ExportAndLoad</code>	

Missingness Patterns

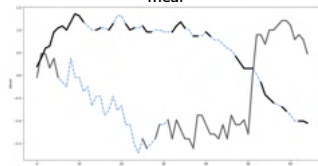
- Two families to replicate a wide range of scenarios, Multi-Block and Mono-Block.

TimeSeries.	
<code>load_series()</code>	<code>[1, 5, 6] [0, 5, 6]</code>
<code>print()</code>	<pre>10.0 10.0 100.0 100.0 100.0 100.0</pre>
<code>normalizer()</code>	
<code>plot()</code>	
<code>Contamination</code>	

Multi-block

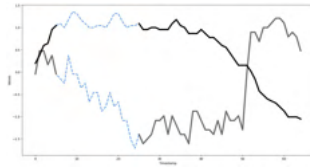


mcar

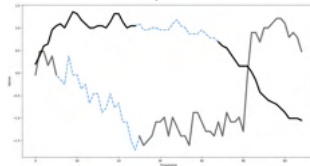


distribution

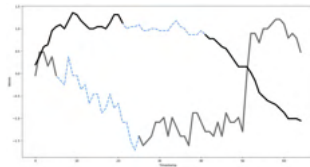
Mono-block



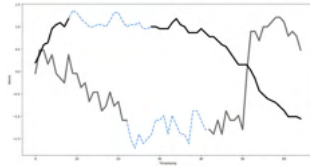
aligned



disjoint



scatter

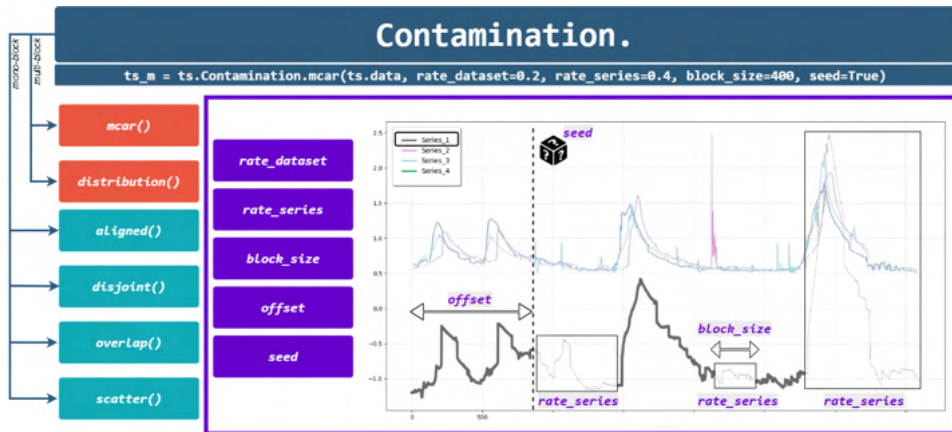


overlap

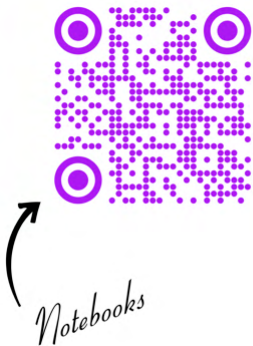
Pattern Customization

- ▶ The patterns can be used with default or customizable parameters.

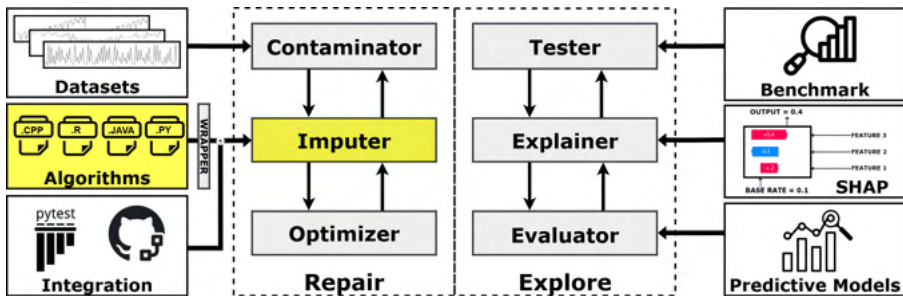
TimeSeries.	
<code>load_series()</code>	<code>[1, 5, 6, 0, 5, 5]</code>
<code>print()</code>	<pre>TS: 1 len: 6 min: 0 max: 6</pre>
<code>normalise()</code>	
<code>plot()</code>	
Contamination	



Demo N°1 - Loading and contamination



https://imputegap-tutorials.github.io/KDD-2025/html/slides_codes.html



Data Splitting Problem

- ▶ In conventional ML pipelines, datasets are partitioned into training and testing subsets using a predefined ratio.
- ▶ In the context of imputation, NaN values might appear in the training set.
- ▶ The model cannot be used to impute data seen during training.

s1	1.0	NaN	3.1	4.0	5.2
s2	1.2	3.2	5.1	2.3	0.0
s3	0.0	1.5	2.8	1.2	3.3
s4	4.3	2.2	3.3	3.3	1.1
s5	1.1	2.0	3.0	4.0	5.0
s6	5.5	NaN	3.3	2.2	1.1
s7	7.7	6.6	5.5	4.4	3.3
s8	2.1	0.0	1.2	2.4	3.6
s9	NaN	9.9	NaN	7.7	NaN
s10	1.6	1.4	8.8	1.1	1.2

Construction of Masks

s1	1.0	NaN	NaN	NaN	5.2	TEST
s2	1.2	3.2	5.1	2.3	0.0	
s3	0.0	1.5	2.8	1.2	3.3	TRAIN
s4	4.3	2.2	3.3	3.3	1.1	
s5	1.1	2.0	3.0	4.0	5.0	
s6	5.5	NaN	3.3	2.2	1.1	
s7	7.7	6.6	5.5	4.4	3.3	
s8	2.1	0.0	1.2	2.4	3.6	
s9	NaN	9.9	NaN	7.7	NaN	
s10	1.6	1.4	8.8	1.1	1.2	

- ▶ Control the overall proportion of missing values by artificially insert missing entries.
- ▶ Create a training mask that excludes all missing values with complete time series.
- ▶ Create a testing mask for imputation, maintaining a consistent missing ratio across all deep learning models (e.g., 60%).

```
1 original_missing_ratio = get_missing_ratio(cont_data_matrix)
```

```
1 cont_data_matrix, new_mask = prepare_testing_set(cont_data_matrix, original_missing_ratio,  
    block_selection=True, tr_ratio=0.8)
```

Data Leakage

s1	1.0	-99	-99	-99	5.2	TEST
s2	1.2	3.2	5.1	2.3	0.0	
s3	0.0	1.5	2.8	1.2	3.3	TRAIN
s4	4.3	2.2	3.3	3.3	1.1	
s5	1.1	2.0	3.0	4.0	5.0	
s6	5.5	-99	3.3	2.2	1.1	
s7	7.7	6.6	5.5	4.4	3.3	
s8	2.1	0.0	1.2	2.4	3.6	
s9	-99	9.9	-99	7.7	-99	
s10	1.6	1.4	8.8	1.1	1.2	

- ▶ Avoid data leakage by replacing all values that must be hidden from the model with large negative constants.
- ▶ The model will reveal potential data leakage during evaluation.

```
1 cont_data_matrix = prevent_leakage(cont_data_matrix, new_mask, nan_val=-99)
```

Final Masking

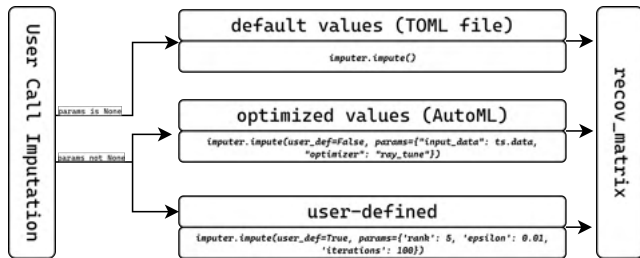
- ▶ Split the training set into internal training and testing subsets for the model.
- ▶ Artificially mask the internal testing set to evaluate model performance during training.

s1	1.0	-99	-99	-99	5.2	TEST
s2	1.2	3.2	5.1	2.3	0.0	
s3	0.0	1.5	2.8	1.2	3.3	TRAIN
s4	4.3	2.2	3.3	3.3	1.1	
s5	1.1	2.0	3.0	4.0	5.0	
s6	5.5	-99	3.3	2.2	1.1	TEST
s7	7.7	6.6	5.5	4.4	3.3	
s8	2.1	0.0	1.2	2.4	3.6	TRAIN
s9	-99	9.9	-99	7.7	-99	
s10	1.6	1.4	8.8	1.1	1.2	

```
1 mask_test, mask_valid, nbr_nans = split_mask_bwt_test_valid(cont_data_matrix, test_rate=1, valid_rate=0, nan_val=-99, seed=42)
2 mask_train = generate_random_mask(cont_data_matrix, mask_test, mask_valid, droprate=0.6, offset=0.1, seed=42)
```

Build Imputation

```
1 imputer = Imputation.MatrixCompletion.CDRec(ts_m)
2 imputer.impute()
```



```
1 def impute(self, user_def=True, params=None):
2     if params is not None:
3         rank, epsilon, iters = self._check_params(user_def, params)
4     else:
5         rank, epsilon, iters = utils.load_parameters(query="default", algorithm=self.algorithm)
6
7     self.recov_data = cdrec(incomp_data=self.incomp_data, rank=rank, iters=iters, epsilon=epsilon)
8
9     return self
```

Imputation Evaluation

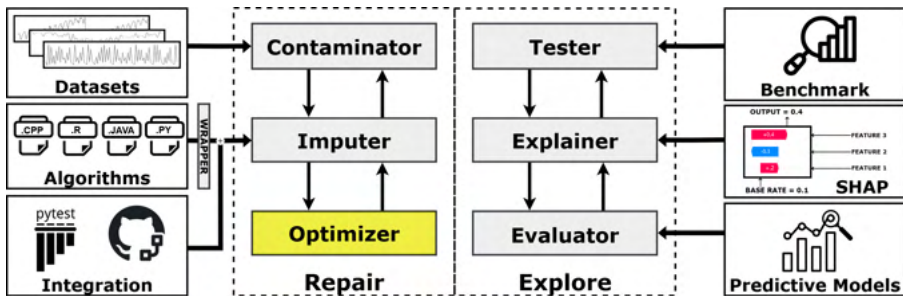
```
class imputegap.recovery.evaluation.Evaluation(input_data, recov_data, incomp_data,  
        algorithm='', verbose=True) \[source\]
```

► Imputation scoring:

```
1 imputer.score(ts.data, imputer.recov_data)  
2 ts.print_results(imputer.metrics)
```

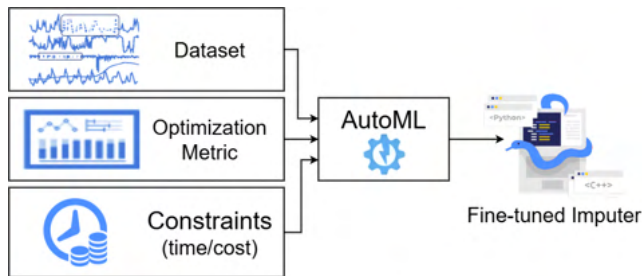
► Dictionary of metrics:

```
1 recov_vals = self.recov_data[np.isnan(self.incomp_data)]  
2 return {"RMSE": self.compute_rmse(), "MAE": self.compute_mae(), "MI": self.compute_mi(), "  
        CORRELATION": self.compute_correlation()}
```



Optimization Methods for Imputation

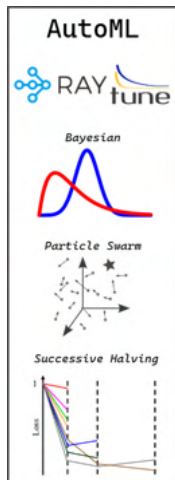
- ▶ Domain-specific imputation algorithms.
- ▶ Manually tuning each parameter and evaluating it is time-consuming.



<https://medium.com/@haimantikamra/azure-automated-ml-the-journey-from-zero-to-ai-hero-b37335963a1f>

```
1 cdrec_imputer.impute(params={'rank': 5, 'epsilon': 0.01, 'iterations': 100}) # RMSE+0.1
2 cdrec_imputer.impute(params={'rank': 10, 'epsilon': 0.001, 'iterations': 50}) # RMSE+0.15
3 cdrec_imputer.impute(params={'rank': 7, 'epsilon': 0.1, 'iterations': 200}) # RMSE-0.1
```

Optimization Methods for Imputation

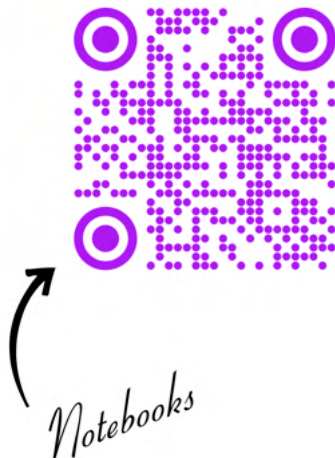


- ▶ The objective function is shared across all optimizers.
- ▶ Parameter ranges are defined for each algorithm. The wider the range, the more computationally expensive the search becomes.

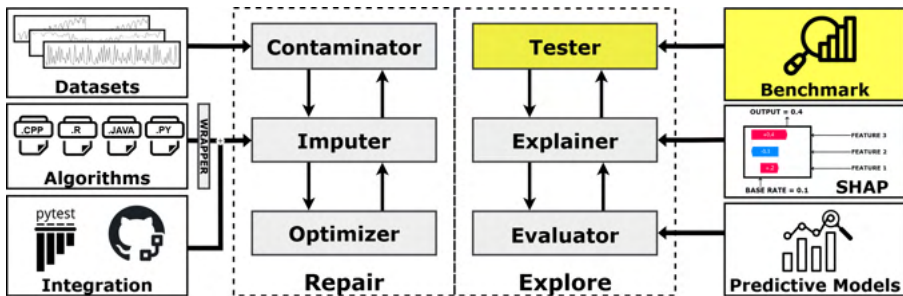
```
1 RAYTUNE_PARAMS = {  
2     "cdrec": {  
3         "rank" : tune.grid_search([i for i in range(2,16,1)]),  
4         "eps"  : tune.loguniform(1e-6, 1),  
5         "iters": tune.grid_search([i*50 for i in range(1,4)])  
6     }  
7 }
```

```
1 imputer.impute(user_def=False, params={"input_data": ts.data, "optimizer": "ray_tune"})
```

Demo N°2 - Imputation

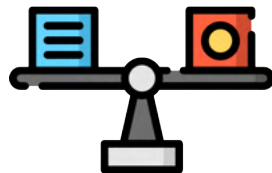


https://imputegap-tutorials.github.io/KDD-2025/html/slides_codes.html



Benchmarking

- ▶ “Fairness” is key in benchmarking.
 - ▶ same code-base
 - ▶ both efficacy and efficiency
 - ▶ multiple metrics and datasets
 - ▶ algorithms should be properly calibrated
- ▶ Existing tools
 - ▶ lack of comprehensive end-to-end support
 - ▶ system failures due to heavy workloads
 - ▶ lack of customization, transparency, and generalization



Workload Execution



18 datasets



7 missingness patterns



34 algorithms



5 optimizers



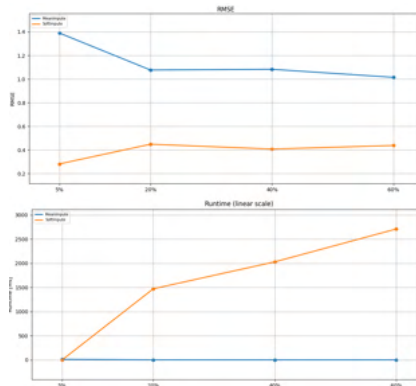
fully customizable
missing rate

- ▶ Over 100k possible combinations.
- ▶ On-demand materialization of results.
- ▶ GPU accelerators are used when available.

Benchmarking

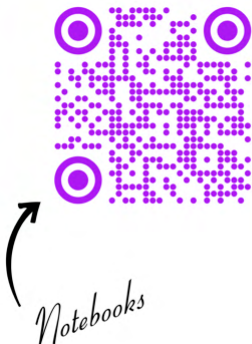
- ▶ No silver bullet imputation algorithm¹.
- ▶ The optimizer derives optimal parameters based on an average missing rate.

```
1 bench.eval(datasets=["bafu", "chlorine"],  
2 patterns=["mcar", "aligned"],  
3 algorithms=["MeanImpute", "SoftImpute"],  
4 optimizers=["default_params"],  
5 x_axis=[0.05, 0.2, 0.4, 0.6],  
6 metrics=["rmse", "runtime"])
```



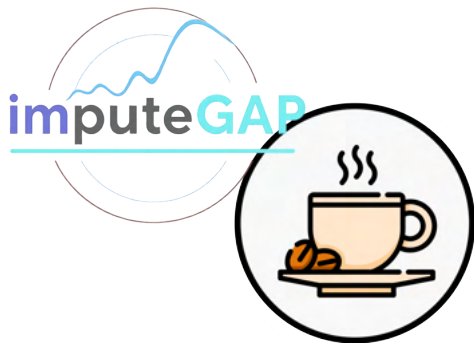
¹Khayati M., Lerner A., Tymchenko Z., and Cudré-Mauroux P.: "Mind the Gap: An Experimental Evaluation of Imputation of Missing Values Techniques in Time Series", PVLDB'20 [Best Experiments and Analysis Award]

Demo N°3 - Benchmark



https://imputegap-tutorials.github.io/KDD-2025/html/slides_codes.html

Coffee Break



<https://imputegap.readthedocs.io/>

Current Section

Introduction and Overview

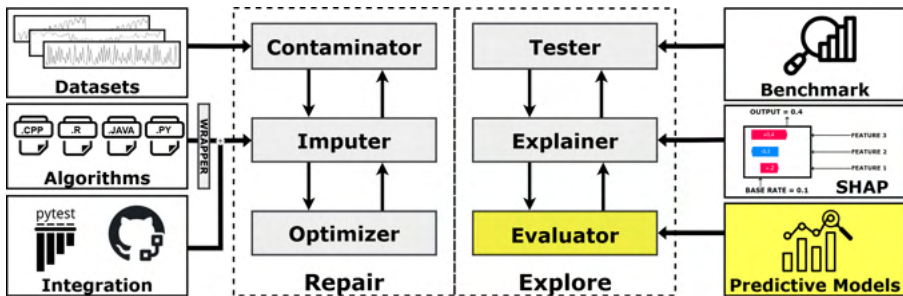
ImputeGAP Features

Pipeline Synthesis

Downstream Analysis

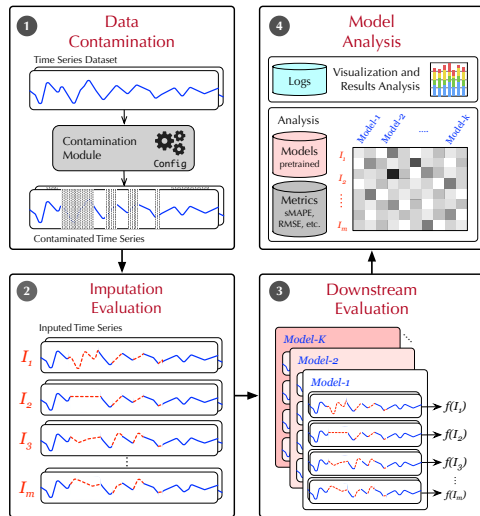
Explainability

Library Extension



Upstream vs. Downstream

- ▶ Downstream analysis requires complete data.
- ▶ Does imputation efficacy map to downstream results?
- ▶ Do upstream and downstream metrics align?
- ▶ Do downstream tasks behave differently w/t imputation?



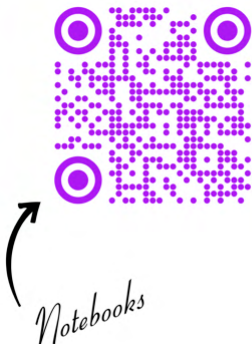
Downstream Implementation

- ▶ Current version implements forecasting task (classification in progress)
- ▶ 16 parameterizable forecasters from sktime and darts APIs.

```
1 ImputeGAP downstream models for forecasting : ['arima', 'bats', 'croston', 'deepar', 'ets', 'exp-  
    smoothing', 'hw-add', 'lightgbm', 'lstm', 'naive', 'nbeats', 'prophet', 'sf-arima', 'theta', '  
    transformer', 'unobs', 'xgboost']
```

```
1 def downstream_analysis(self):  
2     task = self.downstream.get("task", "forecast")  
3     model = self.downstream.get("model", "naive")  
4     params = self.downstream.get("params", None)  
5     plots = self.downstream.get("plots", True)  
6     baseline = self.downstream.get("baseline", "zero-impute")
```

Demo N°4 - Downstream



https://imputegap-tutorials.github.io/KDD-2025/html/slides_codes.html

Current Section

Introduction and Overview

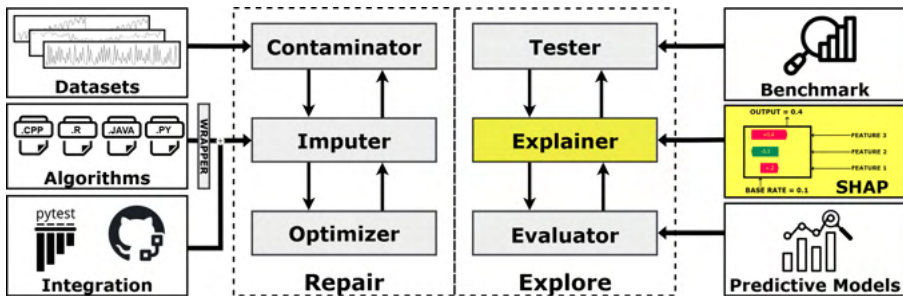
ImputeGAP Features

Pipeline Synthesis

Downstream Analysis

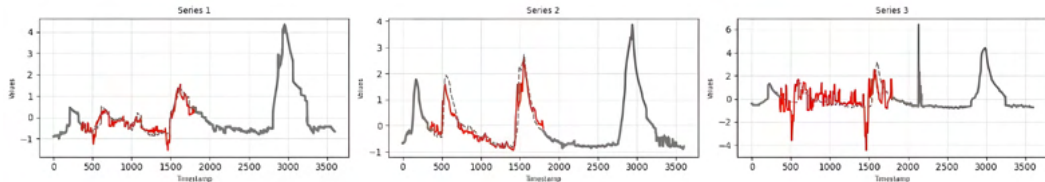
Explainability

Library Extension



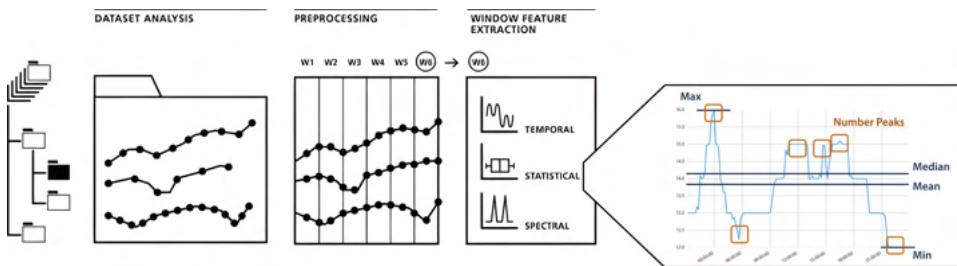
Imputation Inconsistencies

- ▶ The evaluation metrics are reliable, but can we interpret the imputation results?
- ▶ Is the performance of imputation algorithms consistent?
- ▶ Which factors contribute to it?



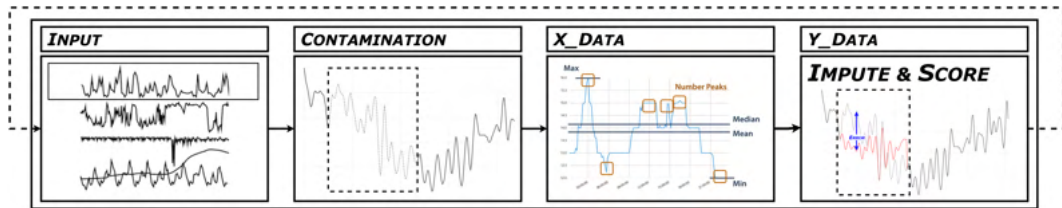
Time Series Characterization

- ▶ Interpret imputation results and diagnose the factors influencing their quality.
- ▶ Using adaptable extractors, we assess imputation quality by analyzing feature-level characteristics of the contaminated data.



Explainer Dataset

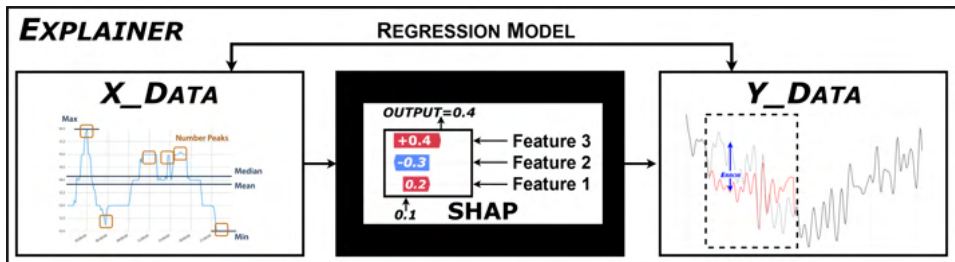
- To leverage the feature information, a loop is executed to extract features for each series contamination.



```
1 for si in range(0, limit): # 1) series iterator
2     tmp = TimeSeries.import_matrix(data)
3     ts_m = utils.config_contamination(tmp, si, pattern="aligned", series_rate=0.6) # 2) contaminate si
4
5     extracted_features = self.extractor(ts_m, categories, features) # 3) extract features
6
7     imputer.impute(user_def=True, params=params) # 4) impute
8     imputer.score(input_data) # 4) score
9
10    x_data.append(extracted_features) # store data
11    y_data.append(imputer.metrics) # store labels
```

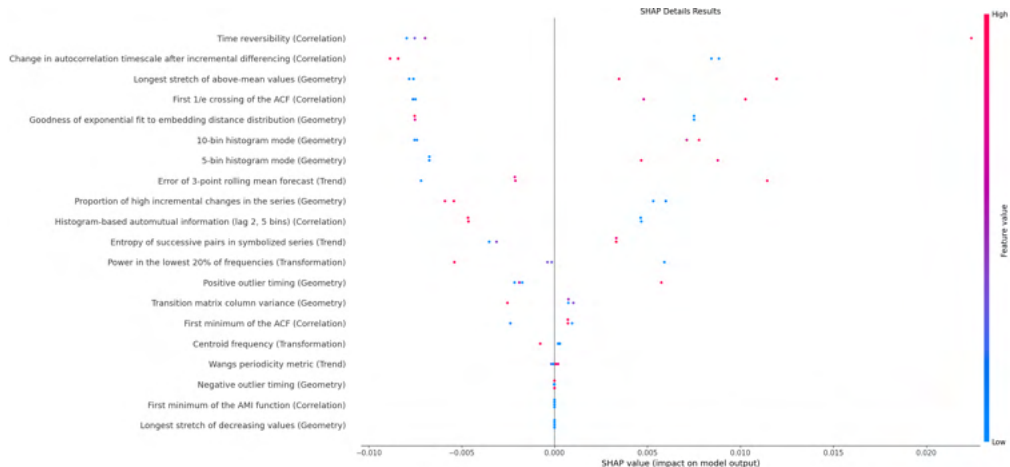
Regression Model

- ▶ The regression model learns the relationship between feature values and the imputation score, guided by the SHAP black-box explainer.
- ▶ SHAP assigns a positive or negative value indicating its contribution to the output.



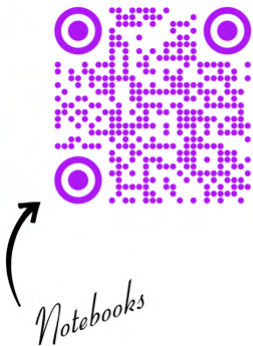
```
1 model = RandomForestRegressor()  
2 model.fit(x_train, y_train)  
3 exp = shap.KernelExplainer(model.predict, x_test)  
4 shval = exp.shap_values(x_test)
```

Feature contribution



- | | | | | | |
|---|---------|------------|-------------|-------------|--|
| 1 | Feat:5 | CDRec/Bafu | score=13.21 | Correlation | Time reversibility |
| 2 | Feat:12 | CDRec/Bafu | score=10.17 | Correlation | Change of autocorrelation timescale after incr. diff |
| 3 | Feat:7 | CDRec/Bafu | score=9.07 | Geometry | Longest stretch of above-mean values |

Demo N°5 - Explainer



https://imputegap-tutorials.github.io/KDD-2025/html/slides_codes.html

Current Section

Introduction and Overview

ImputeGAP Features

Pipeline Synthesis

Downstream Analysis

Explainability

Library Extension

Integration

- ▶ ImputeGAP is fully extensible and customizable
- ▶ Supported programming languages, Python, C++, MATALAB, JAVA, R. (***)
NQU: Mourad fix ***)



add your dataset



add your missingness pattern



add your algorithm



add your optimizer

Integration Python

- Add your custom algorithm logic.

your_llm.py

```
1 def your_llm(ts_m, par_1, par_2, tr_ratio, logs, verbose):
2     start_time = time.time()
3     recov_data = your_algo(ts_m, par_1, par_2, tr_ratio, logs, verbose)
4     end_time = time.time()
5     return recov_data
```

- Organize it within the appropriate algorithm family by creating a dedicated class.

imputation.py

```
1 class Imputation:
2     class LLMs:
3         class YourLLM(BaseImputer):
4             algorithm = "your_llm"
5             def impute(self, user_def=True, params=None, tr_ratio=0.9):
6                 from imputegap.algorithms.your_llm import your_llm
7
8                 self.recov_data = your_llm(ts_m=self.incomp_data, par_1, par_2, tr_ratio, self.logs,
9                                             self.verbose)
9
10            return self
```

Integration Toolbox

- Define default values the hyperparameters.

utils.py

```
1 config = toml.load(filepath)
2 if algorithm == "your_llm":
3     par_1 = int(config[algorithm]['par_1'])
4     par_2 = str(config[algorithm]['par_2'])
5     return (par_1, par_2)
```

default_values.toml

```
1 [your_algo]
2 par_1 = 42
3 par_2 = "val"
```

- Register the algorithm in the toolkit interface for easier access.

utils.py

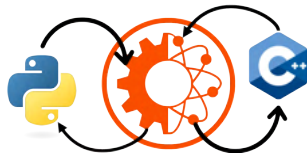
```
1 alg = algorithm.lower().replace('_', '').replace('-', '')
2
3 if alg == "your_algo":
4     imputer = Imputation.YourClass.YourAlgo(incomp_data)
5
6 return imputer
```

Wrappers Support

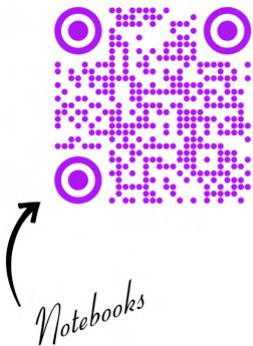
- ▶ ImputeGAP uses wrappers to seamlessly convert input and output variables between Python and C++.
- ▶ C++ code is compiled into shared objects (.so files)

```
1 shared_lib = utils.load_share_lib("lib_cdrec.so")
2 __ctype_matrix = utils.__marshal_as_native_column(__py_matrix);
3 shared_lib.cdrec_imputation(__ctype_matrix);
4
5 py_matrix = utils.__marshal_as_numpy_column(__ctype_matrix);
```

```
1 void
2 cdrec_imputation(double *matrixNative)
3 {
4     arma::mat input = marshal_as_arma(matrixNative, dimN, dimM);
5     Algorithms::CDMissingValueRecovery rmv(input);
6 }
```



Demo N°6 - Integration



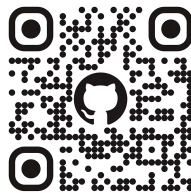
https://imputegap-tutorials.github.io/KDD-2025/html/slides_codes.html

A Hands-on Tutorial on Time Series Imputation with ImputeGAP

Thank you!

quentin.nater@unifr.ch
mourad.khayati@unifr.ch

Questions?



<https://imputegap.readthedocs.io>