Mini-Project (ML for Time Series) - MVA 2024/2025 The ROCKET algorithm: from classification to prediction

Eva Robillard eva.robillard@ens-paris-saclay.fr
Lila Roig lilaroig66@gmail.com

January 8, 2025

1 Introduction

The ROCKET algorithm [Dempster et al., 2020] was originally introduced as a fast and accurate method for time series classification. While classification tasks for time series require high accuracy, existing state-of-the-art methods often achieve top performance but are computationally expensive, like HIVE-COTE [Middlehurst et al., 2021] or InceptionTime [Ismail Fawaz et al., 2020].

ROCKET addresses this challenge by drawing inspiration from Convolutional Neural Networks (CNNs). While CNNs have shown remarkable success in processing structured data like images, ROCKET adapts this concept to time series. ROCKET applies simplified convolutions that resemble those in CNNs, making it computationally efficient while retaining high performance. It uses random convolutional kernels to extract features from univariate time series. By simplifying the convolutional process and employing a linear classifier on the extracted features, ROCKET achieves a better balance between speed and accuracy.

ROCKET was initially designed for univariate time series classification and has not been extensively explored in other contexts or tasks beyond classification. Our work addresses these gaps by extending ROCKET to new tasks and testing its adaptability to multivariate time series. The objectives of this study are twofold:

- **Verification:** Evaluate the performance of the ROCKET algorithm on its original task – time series classification – using a new dataset.
- **Adaptation:** Extend the ROCKET algorithm to multivariate data and to a novel task – time series prediction – and evaluate its performance in this context.

While Lila focused on the implementation of the code and methods, Eva performed the tests and wrote this report. The code for ROCKET was made available in the original paper [Dempster et al., 2020] and the code to load datasets from .tsf format to a Pandas dataframe format for Python was available in the Monash Dataset paper [Godahewa et al., 2021] [1]. Except those two methods, all the code was implemented from scratch. The improvement of our work resides in the ability to apply ROCKET for prediction of univariate and multivariate time series.

[1]https://github.com/rakshitha123/TSForecasting?tab=readme-ov-file

2 Method

2.1 The ROCKET algorithm for classification

The ROCKET (*RandOm Convolutional KErnel Transform*) method is a fast and accurate approach to time series classification. It achieves state-of-the-art accuracy by combining random convolutional kernels and simple, scalable classifiers. Its use of a large variety of random kernels and a novel feature extraction approach (PPV and max pooling) enables it to handle diverse time series datasets efficiently while maintaining computational simplicity. Its steps are explained in the following as well as on Figure 1.

Step 1: Kernel Definition Each kernel is different and is defined by its length, weights, bias, dilation, and padding, with all parameters randomly initialized. The length l_k is selected uniformly from $\{7, 9, 11\}$. Weights $\mathbf{W}$ are sampled as $w \sim \mathcal{N}(0, 1)$, followed by mean-centering: $\omega = \mathbf{W} - \overline{\mathbf{W}}$. The bias b is drawn from a uniform distribution $b \sim \mathcal{U}(-1, 1)$. Dilation d is sampled on an exponential scale: $d = \lfloor 2^x \rfloor,\ x \sim \mathcal{U}(0, \log_2((l_t - 1)/(l_k - 1)))$, where l_t is the time series length. Dilation introduces spacing between sampled elements, enabling kernels to capture patterns at varying frequencies. Padding is applied at random (with equal probability) for each kernel, ensuring alignment with boundary elements when necessary. Stride is always set to one. Nonlinearities such as ReLU are not applied, ensuring the resulting feature maps are agnostic to specific transformations. Input time series are assumed to be normalized with a mean of zero and a standard deviation of one. These kernel parameters were determined to optimize classification accuracy on development datasets and generalize well to unseen data.

Step 2: Convolution The randomly generated kernel k is convolved with each input time series X following the convolutional transform: $X[i] * k = \sum_{j=0}^{l_k - 1} X[i + j \cdot d] \cdot w[j] + b$.

Step 3: Feature extraction For each kernel, two features are extracted from the obtained feature map, which means that k kernels actually result in $2k$ features per time series. The extracted features are the **Maximum Value**, which captures the strongest presence of the kernel's pattern within the time series, and the **Proportion of Positive Values (PPV)**, which measures the prevalence of the pattern, calculated as the fraction of positive elements in the feature map.

Step 4: Classification Once the features are extracted from the time series, they serve as input to a linear classifier. In our case, we use a ridge regression classifier.

2.2 Adapting ROCKET for Multivariate Time Series

Recall that for univariate time series, the ROCKET algorithm applies a **different** set of convolutional kernels for each univariate series. As described in the MiniROCKET paper [Dempster et al., 2021], ROCKET can be adapted for multivariate time series by applying the **same** convolutional kernels independently to each variable in the series. This approach processes the entire multivariate series at once: the same convolutional kernels are generated once and then applied simultaneously and independently to all variables in the multivariate series. This method extracts features from each variable separately without directly leveraging correlations between them, making it particularly suitable for heterogeneous data. We experimented with this method by modifying the ROCKET algorithm directly for multivariate series. However, it resulted in significant memory issues on large datasets due to the linear increase in the number of calculated features: Total feature size $=$ num_kernels $\times$ 2 $\times$ num_variables. To address this, we propose an alternative approach that involves applying ROCKET independently to each variable in the multivariate series, treating each dimension as a separate univariate time series. Each variable is analyzed separately, decoupling the processing of different dimensions. However, unlike the standard univariate case where different kernels are generated for each series, here, all variables share the same set of convolutional kernels, which are generated once and reused for every dimension. This approach, conceptually equivalent to the previous method, offers advantages in terms of memory efficiency and modularity while maintaining the separation of variables. Alternatively, in order to capture dependencies between variables, the MultiROCKET algorithm [Tan et al., 2022] could be implemented, although it may also impose significant memory constraints on large multivariate time series.

2.3 Adapting ROCKET to Time Series Forecasting

To adapt ROCKET for forecasting, we apply the algorithm in the same way as for classification by extracting features from time series. However, instead of using a classifier, we replace it with a **regression model** that uses the features computed by ROCKET to make predictions for the chosen forecast horizon, denoted H.

Step 1: Sliding Window Transformation ROCKET, originally designed for classification, takes input time series in the format: (`num_examples`, `input_length`), where `num_examples` is the number of time series in the classification dataset and `input_length` is the length of the time series. However, in the case of univariate forecasting, we have only a single time series of size (`input_length`). Thus, we transform our univariate time series of size (`input_length`) into a time series of size (`num_examples`, `input_length`) = (`num_windows`, `window_length`) using sliding windows applied to the series. For each series, the sliding window method is used to segment the training series (`train`) into usable sub-series. This method involves sliding a fixed-size window (`window_size = W`) over the series with a defined step (stride). At each window position, two sets are created: `X_train`, which contains past observations (values in the input window), and `y_train`, which contains future values to predict (values in the output window). This approach generates multiple examples of time sequences from a single series, thereby increasing the dataset size for model training. Specifically, the new dataset is, for `X_train`, a matrix of size `num_windows` $\times$ W, where each row is an input window and, for `y_train`, a matrix of size `num_windows` $\times$ H, where each row contains future values to predict. The sliding windows are computed using a vectorized implementation without explicit loops, leveraging indices to simultaneously generate all input and output windows. This approach optimizes efficiency and reduces computation time for large time series. We implement a custom sliding window function, drawing inspiration from the ideas of different sources.[2] [3] The sliding window method allows us to transform a time series into a set of input-output pairs X and y, ready for use in a forecasting model. Moreover, the filling of incomplete windows ensures that all generated windows have a constant size, which is crucial for training models requiring fixed-dimension tensors. Before applying sliding windows, we ensure that there are enough points in the series to form at least one sliding window, apply a forecasting horizon, and include a stride between successive windows. The size of the sliding window must

[2]https://github.com/timeseriesAI/tsai/blob/main/tsai/data/preparation.py

[3]https://towardsdatascience.com/fast-and-robust-sliding-window-vectorization-with-numpy-3ad950ed62f5

not exceed half the total length of the time series ($l/2$). For an example illustrating this process, refer to the Appendix.

Step 2: Feature Extraction with ROCKET Once the training windows are generated, ROCKET is applied to the input windows (`X_train`). A set of random convolutional kernels is generated, and these kernels are applied vectorially across all input windows, producing a new matrix `X_train_features` containing extracted temporal features. Each kernel generates two features: the maximum value of the convolution and the proportion of positive values. If the total number of convolutional kernels is denoted as `num_kernels`, the resulting feature matrix `X_train_features` has dimensions (`num_windows`, `num_kernels` × 2). For the testing phase, the last input window from the training series is extracted and used as `X_test`. This window is transformed in the same way as the training data: the same convolutional kernels are applied, generating a matrix `X_test_features` of size (1, `num_kernels` × 2).

Step 3: Regression Model Integration Once this matrix is generated, a regression model, previously trained on `X_train_features` and `y_train`, is applied to `X_test_features` to produce a prediction. The results represent the forecasted future values for the specified horizon.

3 Data

3.1 ROCKET for classification: OliveOil dataset

The OliveOil dataset[4] is composed of spectra extracted via Fourier transform infrared spectroscopy (FTIR) of 60 oils from four European countries. It was used in several papers as a classification dataset. The dataset thus contains 60 time series, all univariate (only one variable per time serie) and each serie contains 570 time points. As recommended in the ROCKET paper [Dempster et al., 2020], we normalize each time series. We visualize the series and notice they are all indistinguishable at first glance as observed in Figure 2 in Appendix.

3.2 ROCKET for prediction: Monash Archive

The Monash archive [Godahewa et al., 2021] is a time series forecasting archive, which includes a total of 58 datasets. These datasets span various sampling rates (yearly, quarterly, monthly, . . .). Some series have missing values. The archive contains 2 versions of each of these, one with and one without missing values, replaced by using an appropriate imputation technique (this is the case we will consider). A summary is provided in Table 1. We did not consider all datasets, particularly those that were too computationally expensive to process. In the Monash archive, univariate time series are stored in files where each file contains multiple univariate time series. Each row (n) represents a complete time series, while the columns (T) correspond to different time steps. The time series within the same file can have variable lengths (T). Multivariate time series, on the other hand, are stored in files where each file contains a single multivariate time series. The rows (T) represent successive time steps, and the columns (n) correspond to the different variables observed at each time step.

The forecast horizon H of each time series in the Monash archive is determined based on the frequency of the series provided by the archive. To determine the forecast horizon, we rely on the

[4]https://www.timeseriesclassification.com/description.php?Dataset=OliveOil

recommendations from the Monash paper. The forecast horizon allows us to split our time series of length l into a test set of size forecast_horizon $= H$ and a training set of size $l - H$ to perform the forecasting task. We also need to adapt the size of the sliding windows (W) automatically for each time series. To determine the size of the sliding windows for each series, we calculate the number of lags (past values). Lags represent the past observations that the model uses to predict future values. Thus, the number of lags corresponds to the size of the sliding windows used by the model for forecasting. Some papers recommend that the number of lags is determined based on the seasonality multiplied by 1.25 [Godahewa et al., 2021, Hewamalage et al., 2021]. For short time series, lags are calculated as the forecast horizon multiplied by 1.25. Adjustments are made for specific datasets, such as hourly data, where daily seasonality can be used to reduce memory requirements. Finally, the number of lags is adjusted to be at least equal to the forecasting horizon, ensuring the model has sufficient past data to make reliable predictions.

The application of ROCKET to the forecasting task begins with loading and normalizing the time series, as recommended in the ROCKET paper. Each series is then split into a training set (`train`) of size $l - H$ and a test set (`test`) of size H, based on the forecast horizon H, following the Monash paper's recommendations [Godahewa et al., 2021].

4 Results

4.1 ROCKET for classification

We validated ROCKET's abilities in classification as we observed that the algorithm performs well on the task for the OliveOil dataset, achieving an accuracy of **88.889%**. This result demonstrates the algorithm's ability to extract meaningful patterns and features from the time series data, even in cases where distinguishing between classes is challenging at first glance.

4.2 ROCKET for prediction

In order to evaluate the performance of the prediction task on the Monash Archive datasets, as highlighted in Table 1, we calculated several evaluation metrics, including MASE, MSE, RMSE, and sMAPE (as detailed in the Appendix), to assess the quality of the forecasts. All results, including processing time, were recorded in an Excel file for comparative analysis across datasets. Additionally, we examined the distributions of the metrics across different frequencies (Figure 3), regressors (Figure 4), and dataset types, as univariate or multivariate (Figure 5).

The MASE compares our model's predictions to a naive model that uses the last seasonality as a prediction. Most results show a MASE > 1, indicating the model isn't competitive. The weekly-frequency datasets performed best, with scores consistently under 1, suggesting the models are most effective at capturing weekly patterns. The ridge regressor outperformed XGBoost and Random Forest across all metrics but exhibited more spread in its predictions, indicating less consistency. Multivariate datasets showed higher absolute errors but more consistent predictions, while univariate datasets had more variability in performance.

Overall, the ROCKET-based prediction model often fails to outperform naive seasonal predictions, indicating room for improvement. The models struggle with high-frequency data (hourly, daily) but show more stable predictions for weekly patterns, suggesting the need for refinement for real-time forecasting applications.

References

[Dempster et al., 2020] Dempster, A., Petitjean, F., and Webb, G. I. (2020). Rocket: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34(5):1454–1495.

[Dempster et al., 2021] Dempster, A., Schmidt, D. F., and Webb, G. I. (2021). Minirocket: A very fast (almost) deterministic transform for time series classification. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, pages 248–257.

[Godahewa et al., 2021] Godahewa, R., Bergmeir, C., Webb, G. I., Hyndman, R. J., and Montero-Manso, P. (2021). Monash time series forecasting archive. In *Neural Information Processing Systems Track on Datasets and Benchmarks*.

[Hewamalage et al., 2021] Hewamalage, H., Bergmeir, C., and Bandara, K. (2021). Recurrent neural networks for time series forecasting: Current status and future directions. *International Journal of Forecasting*, 37(1):388–427.

[Ismail Fawaz et al., 2020] Ismail Fawaz, H., Lucas, B., Forestier, G., Pelletier, C., Schmidt, D. F., Weber, J., Webb, G. I., Idoumghar, L., Muller, P.-A., and Petitjean, F. (2020). Inceptiontime: Finding alexnet for time series classification. *Data Mining and Knowledge Discovery*, 34(6):1936–1962.

[Middlehurst et al., 2021] Middlehurst, M., Large, J., Flynn, M., Lines, J., Bostrom, A., and Bagnall, A. (2021). Hive-cote 2.0: a new meta ensemble for time series classification. *Machine Learning*, 110(11):3211–3243.

[Tan et al., 2022] Tan, C. W., Dempster, A., Bergmeir, C., and Webb, G. I. (2022). Multirocket: multiple pooling operators and transformations for fast and effective time series classification. *Data Mining and Knowledge Discovery*, 36(5):1623–1646.

Appendix

A. The ROCKET Algorithm

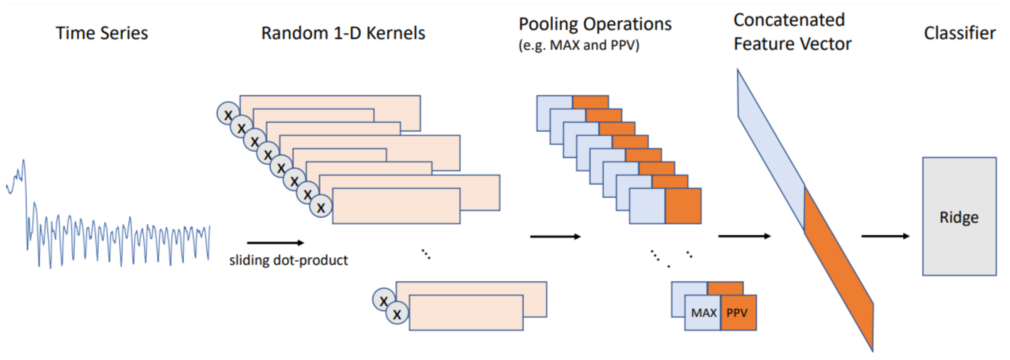


Figure 1: The steps of the ROCKET algorithm

A note on the existing implementation ROCKET is implemented in Python, utilizing just-in-time (JIT) compilation via Numba for efficient computation of the convolutional transform. The design allows for natural parallelism, enabling the use of multiple CPU cores or GPUs to accelerate the process further.

A note on computational complexity The convolution operation in the ROCKET algorithm has a complexity of $O(k \cdot n \cdot l_t)$, where k is the number of kernels, n the number of time series, and l_t the time series length. The small, fixed kernel size (maximum $l_k = 11$) ensures computational efficiency. The ridge regression has a complexity of $O(n^2 \cdot f)$ or $O(n \cdot f^2)$, depending on whether $f \gg n$ or $f \ll n$, where $f = 2k$ is the feature count.

B. OliveOil Dataset

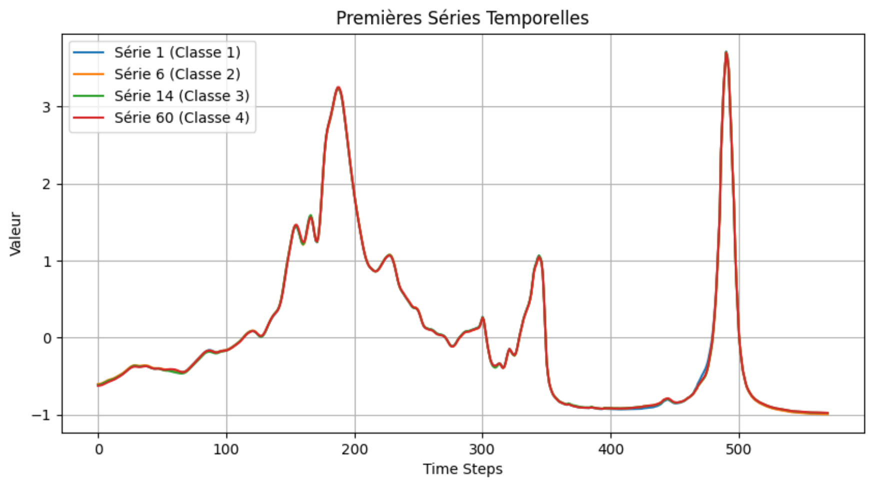


Figure 2: Plot of multiple series from the dataset: they are undistinguishable

C. Sliding Window Example

We will illustrate how sliding windows are generated using an example. Consider a time series input of length $T = 12$:

$$X = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]$$

And consider:

- Window size: $W = 5$
- Forecasting horizon: $H = 3$
- Stride: $s = 2$

First, we calculate the number of sliding windows:

$$n_{\text{windows}} = \frac{T - W - H}{s} + 1 = 3$$

Then, we generate the windows:

$$X = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 6 & 7 \\ 5 & 6 & 7 & 8 & 9 \end{bmatrix} \quad y = \begin{bmatrix} 6 & 7 & 8 \\ 8 & 9 & 10 \\ 10 & 11 & 12 \end{bmatrix}$$

Here, we note that the series is too short for the next output window to be complete, as there are not enough points remaining in the series after the last input window to cover the forecasting horizon $H = 3$. In this case, the last available value is used to fill the window, and we obtain:

$$X = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 6 & 7 \\ 5 & 6 & 7 & 8 & 9 \\ 7 & 8 & 9 & 10 & 11 \end{bmatrix} \quad y = \begin{bmatrix} 6 & 7 & 8 \\ 8 & 9 & 10 \\ 10 & 11 & 12 \\ 12 & 12 & 12 \end{bmatrix}$$

D. Monash Archive datasets

Dataset	Domain	Nb. Series	Min. Len.	Max. Len.	Nb. Freq.	Multivariate	Used ?
M1	Multiple	1001	15	150	3	No	YES
M3	Multiple	3003	20	144	4	No	YES
M4	Multiple	100000	19	9933	6	No	YES
Tourism	Multiple	1311	11	333	3	No	YES
CIF 2016	Banking	72	34	120	1	Yes	YES
LSM	Energy	5560	288	39648	1	Yes	-
Aus. Elec. Dem.	Energy	5	230736	232272	1	Yes	-
Wind Farms	Energy	339	6345	527040	1	Yes	-
Dominick	Sales	115704	28	393	1	No	-
Bitcoin	Economic	18	2659	4581	1	No	YES
Pedestrian Cts	Transport	66	576	96424	1	No	-
Vehicle Trips	Transport	329	70	243	1	Yes	-
KDD Cup 2018	Nature	270	9504	10920	1	No	YES
Weather	Nature	3010	1332	65981	1	No	-
NN5	Banking	111	791	791	2	Yes	YES
Web Traffic	Web	145063	803	803	2	Yes	-
Solar	Energy	137	52560	52560	1	No	-
Electricity	Energy	321	26304	26304	2	Yes	YES
Car Parts	Sales	2674	51	51	1	No	YES
FRED-MD	Economic	107	728	728	2	Yes	YES
SF Traffic	Transport	862	17544	17544	2	Yes	-
Rideshare	Transport	2304	541	541	1	Yes	YES
Hospital	Health	767	84	84	2	Yes	YES
COVID Deaths	Nature	266	212	212	1	No	YES
Temp. Rain	Nature	32072	725	725	1	Yes	-
Sunspot	Nature	1	73931	73931	1	No	-
S. River Flow	Nature	1	23741	23741	1	No	-
US Births	Nature	1	7305	7305	1	No	-
Solar Power	Energy	1	7397147	7397147	1	No	-
Wind Power	Energy	1	7397147	7397147	1	No	-

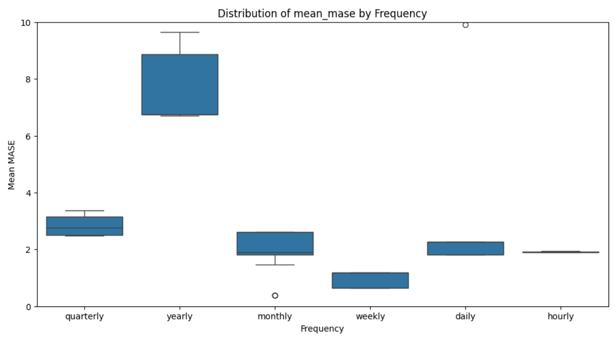
Table 1: Datasets in the Current Time Series Forecasting Archive

D. Forecasting Evaluation metrics

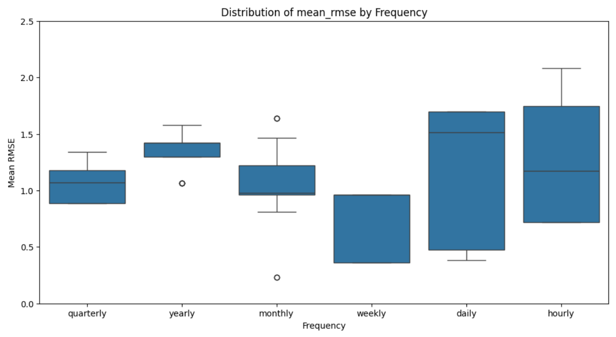
The forecasting results are evaluated using the following metrics:

- **MASE (Mean Absolute Scaled Error):** Accounts for scale invariance and seasonality.
- **MSE (Mean Squared Error):** Penalizes large deviations.
- **RMSE (Root Mean Squared Error):** Provides interpretability in the original scale.
- **sMAPE (Symmetric Mean Absolute Percentage Error):** Measures relative error.

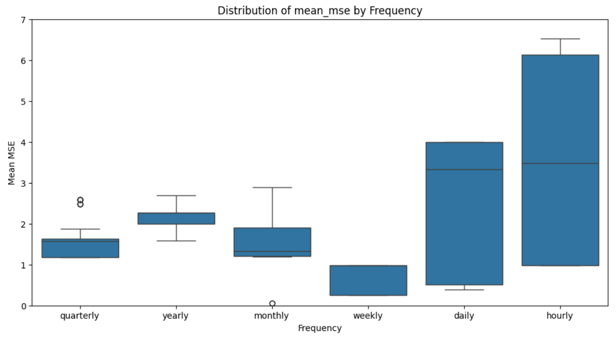
E. Results



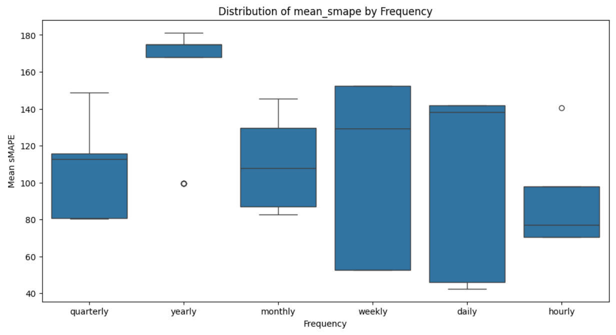
(a) MASE mean by frequency



(b) RMSE mean by frequency

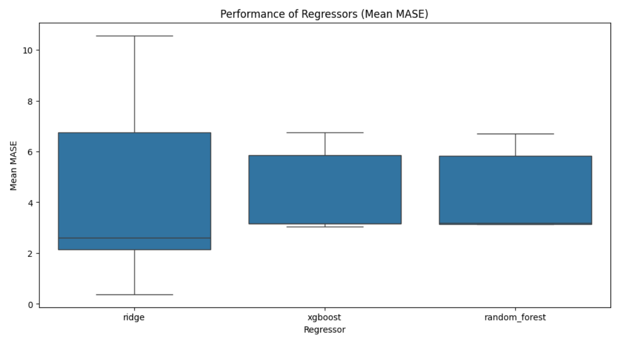


(c) MSE mean by frequency

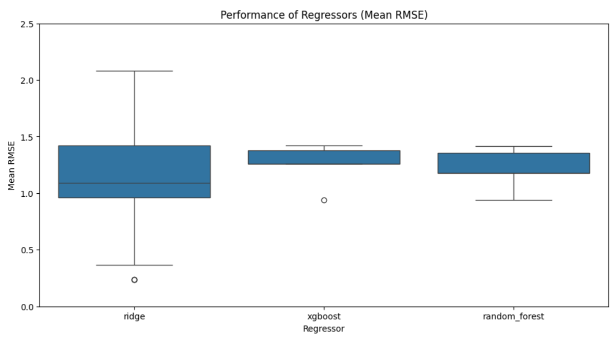


(d) sMAPE mean by frequency

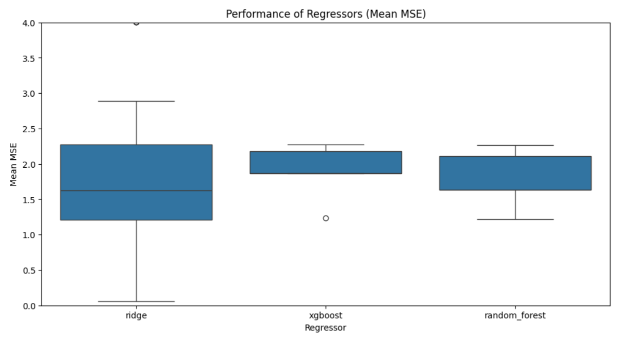
Figure 3: Metrics distribution by frequency



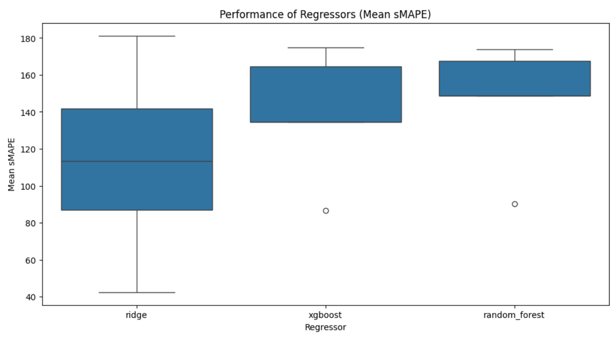
(a) MASE mean by regressor



(b) RMSE mean by regressor

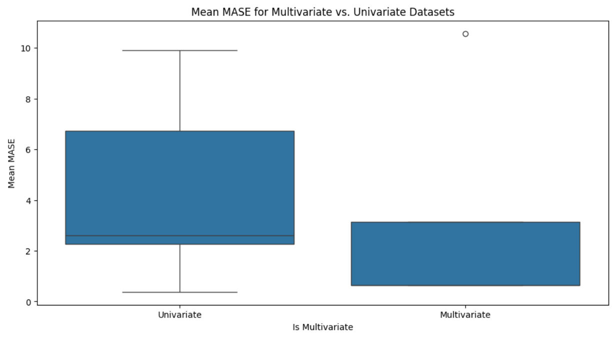


(c) MSE mean by regressor

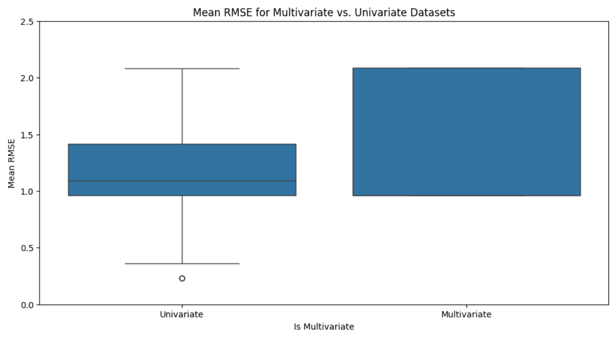


(d) sMAPE mean by regressor

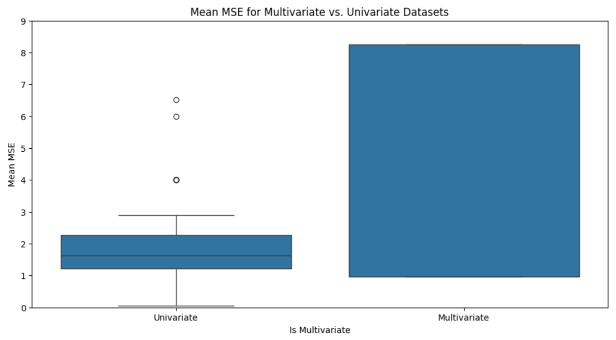
Figure 4: Metrics distribution by regressor



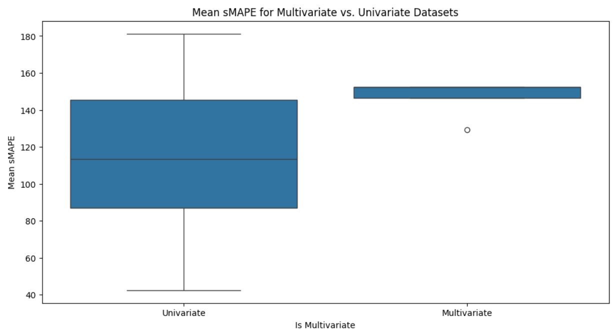
(a) MASE mean by dataset type



(b) RMSE mean by dataset type



(c) MSE mean by dataset type



(d) sMAPE mean by dataset type

Figure 5: Metrics distribution for univariate and multivariate datasets