

dython: A Set of Analysis and Visualization Tools for Data and Variables in Python

Shaked Zychlinski  ¹

¹ Independent Researcher, Israel

DOI: [10.21105/joss.09174](https://doi.org/10.21105/joss.09174)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: Richard Littauer 

Reviewers:

- [@sayantikabanik](#)
- [@sepandhaghghi](#)

Submitted: 04 October 2025

Published: 17 December 2025

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

Summary

Exploratory data analysis (EDA) frequently requires quantifying and visualizing associations between variables. In datasets with mixed variable types (continuous and categorical), analysts must manually choose suitable metrics (Pearson's R, correlation ratio, Cramér's V, Theil's U, etc.), compute them, then assemble the results and plot, often with custom logic. The **dython** package automates much of this workflow: it inspects variable types, computes suitable association measures, and returns a clean tabular result.

As **dython** was designed to be used for research, it puts an emphasis on the visual plots generated by its core methods, providing highly-readable and customizable visualizations of the output, treating these as a core component rather than a byproduct.

In short, **dython** lowers the friction for inter-variable association analysis in mixed-type datasets and improves the reproducibility of EDA workflows.

First version release date: February 2018.

Statement of Need

While there are many statistical and visualization libraries in Python (e.g., pandas ([team, 2025](#)), scipy ([Virtanen et al., 2020](#)), scikit-learn ([Pedregosa et al., 2011](#)), seaborn ([Waskom, 2021](#))), they treat continuous data, categorical data, and the overall visualization separately. Users often resort to custom glue code to:

1. determine which columns are categorical vs. numeric,
2. choose an appropriate association statistic (e.g., Pearson's R for numeric–numeric, correlation ratio for numeric–categorical, Cramér's V or Theil's U for categorical–categorical),
3. compute those pairwise,
4. assemble a matrix or graph, and
5. annotate, visualize, and interpret the results.

This fragmentation results in excessive copying and pasting of boilerplate code, inconsistencies, or risk of mistakes, especially in exploratory settings or pipelines.

dython addresses this gap by providing a unified, high-level API that:

- **infers variable types,**
- **automatically selects appropriate measures,**
- **returns structured and annotated output,**

- **offers visualization** (heatmaps, annotation) integrations, and
- **offers model evaluation tools** (ROC, AUC, thresholding) for classification tasks

Therefore, `dython` helps data scientists, statisticians, and researchers spend less time writing glue code and more time focusing on insights.

Functionality

Below is a summary of existing methods of `dython`, per module.

`nominal`

Method	Description
<code>associations</code>	Computes associations between mixed-type features
<code>cluster_correlations</code>	Applies clustering to reorder a correlation matrix
<code>compute_associations</code>	Deprecated; replaced by <code>associations(compute_only=True)</code>
<code>conditional_entropy</code>	Computes conditional entropy of X given Y
<code>correlation_ratio</code>	Computes correlation between categorical and numeric variables
<code>cramers_v</code>	Computes Cramér's V between categorical variables
<code>identify_nominal_columns</code>	Detects nominal (categorical) columns
<code>identify_numeric_columns</code>	Detects numeric columns
<code>numerical_encoding</code>	Encodes a mixed dataset into numeric format
<code>replot_last_associations</code>	Replots the last association heatmap
<code>theils_u</code>	Computes Theil's U (uncertainty coefficient)

`model_utils`

Method	Description
<code>ks_abc</code>	Computes KS statistic, ABC, and optional plot
<code>metric_graph</code>	Plots ROC/PR curves for classifiers
<code>random_forest_feature_importance</code>	Plots feature importance for Random Forest models

`sampling`

Method	Description
<code>boltzmann_sampling</code>	Samples values under Boltzmann distribution
<code>weighted_sampling</code>	Samples values using weighted probabilities

`data_utils`

Method	Description
<code>identify_columns_with_na</code>	Returns dataset columns containing NA values
<code>identify_columns_by_type</code>	Identifies columns of requested data types
<code>one_hot_encode</code>	Converts a 1D array of integers into a one-hot matrix

Method	Description
split_hist	Plots a histogram split by categories

Code Examples

Associations

- `dython.nominal.associations(df, theil_u=False, plot=False, return_results=False, **kwargs)`

This computes pairwise associations across all columns in a pandas DataFrame `df`. Internally, for each pair, it selects a measure appropriate to the variable types:

- continuous–continuous → Pearson correlation (or Spearman, if configured)
- continuous–categorical → correlation ratio
- categorical–categorical → Cramér's V or Theil's U

It outputs a pandas DataFrame (square matrix) of association values and optionally produces a heatmap (with annotations).

Example usage:

```
from dython.nominal import associations
assoc_df = associations(my_df, theil_u=True, plot=True)
```

Model evaluation

- `dython.model_utils.metric_graph(y_true, y_pred, metric='roc', **kwargs)`

This utility helps visualize classification performance. For a given true-label array `y_true` and predicted scores `y_pred`, it can plot ROC curves, compute AUC for each class (in multiclass settings), and show threshold recommendations.

Example:

```
from dython.model_utils import metric_graph
metric_graph(y_true, y_pred_probs, metric='roc')
```

- `dython.model_utils.ks_abc(y_true, y_pred, **kwargs)` This performs the Kolmogorov–Smirnov test over the positive and negative distributions of a binary classifier, and compute the area between curves.

Example:

```
from dython.model_utils import ks_abc
ks_abc(y_true, y_pred_probs)
```

Related work

Several libraries provide components somewhat overlapping `dython`'s functionality:

- `scipy.stats` ([Virtanen et al., 2020](#)), `statsmodels` ([Seabold & Perktold, 2010](#)) — full support for continuous correlations and tests, but limited categorical association tools
- `scikit-learn` ([Pedregosa et al., 2011](#)) — mutual information, label encoding, classification metrics, but lacks seamless cross-type association matrices
- `pingouin` ([Vallat, 2018](#)) — a statistical package including correlation and effect sizes, but does not integrate categorical–categorical measures like Theil's U or automatic visualization

Installation

You can install the released version via:

- **pip**: pip install dython
- **conda**: conda install -c conda-forge dython
- **Source**: pip install git+https://github.com/shakedzy/dython.git

Dependencies include standard scientific Python packages such as numpy ([Harris et al., 2020](#)), pandas ([team, 2025](#)), scipy ([Virtanen et al., 2020](#)), scikit-learn ([Pedregosa et al., 2011](#)), matplotlib ([Hunter, 2007](#)), and seaborn ([Waskom, 2021](#)).

Usage Mention

Throughout its lifetime, dython has been used in many projects, including:

- [Official implementation of TabDDPM](#) ([Kotelnikov et al., 2023](#)) by [Yandex Research](#)
- [gretel-synthetics](#) by [Gretel.ai](#)
- [torchmetrics](#) by [Lightning AI](#)
- [ydata-quiality](#) by [YData](#)

Acknowledgements

The author thanks users and contributors who have filed issues, submitted pull requests, or suggested enhancements, as well as the authors and communities of the foundational packages on which dython builds.

Appendix: Code Examples

A minimal example using associations:

```
import pandas as pd
from sklearn import datasets
from dython.nominal import associations

# Load dataset
iris = datasets.load_iris()

# Convert int classes to strings to allow associations method
# to automatically recognize categorical columns
target = ["C{}".format(i) for i in iris.target]

# Prepare data
X = pd.DataFrame(data=iris.data, columns=iris.feature_names)
y = pd.DataFrame(data=target, columns=["target"])
df = pd.concat([X, y], axis=1)

# Plot features associations
associations(df)
```

This would output:

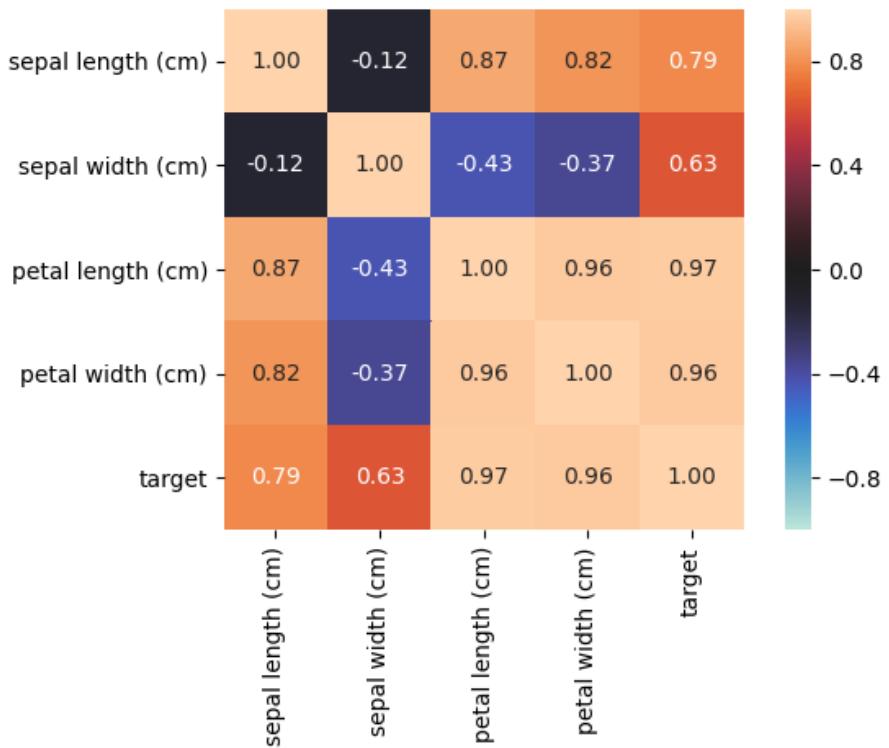


Figure 1: Example of an associations heatmap plotted over the Iris Dataset

A minimal example using `metric_graph`:

```

from sklearn import datasets, svm
from sklearn.preprocessing import label_binarize
from sklearn.model_selection import train_test_split
from sklearn.multiclass import OneVsRestClassifier
from dython.model_utils import metric_graph

# Load data
iris = datasets.load_iris()
X = iris.data
y = label_binarize(iris.target, classes=[0, 1, 2])

# Add noisy features
random_state = np.random.RandomState(4)
n_samples, n_features = X.shape
X = np.c_[X, random_state.randn(n_samples, 200 * n_features)]

# Train a model
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.5, random_state=0
)
classifier = OneVsRestClassifier(
    svm.SVC(kernel="linear", probability=True, random_state=0)
)

# Predict

```

```

y_score = classifier.fit(X_train, y_train).predict_proba(X_test)

# Plot ROC graphs
metric_graph(
    y_test, y_score, "roc", class_names_list=iris.target_names
)

```

This would output:

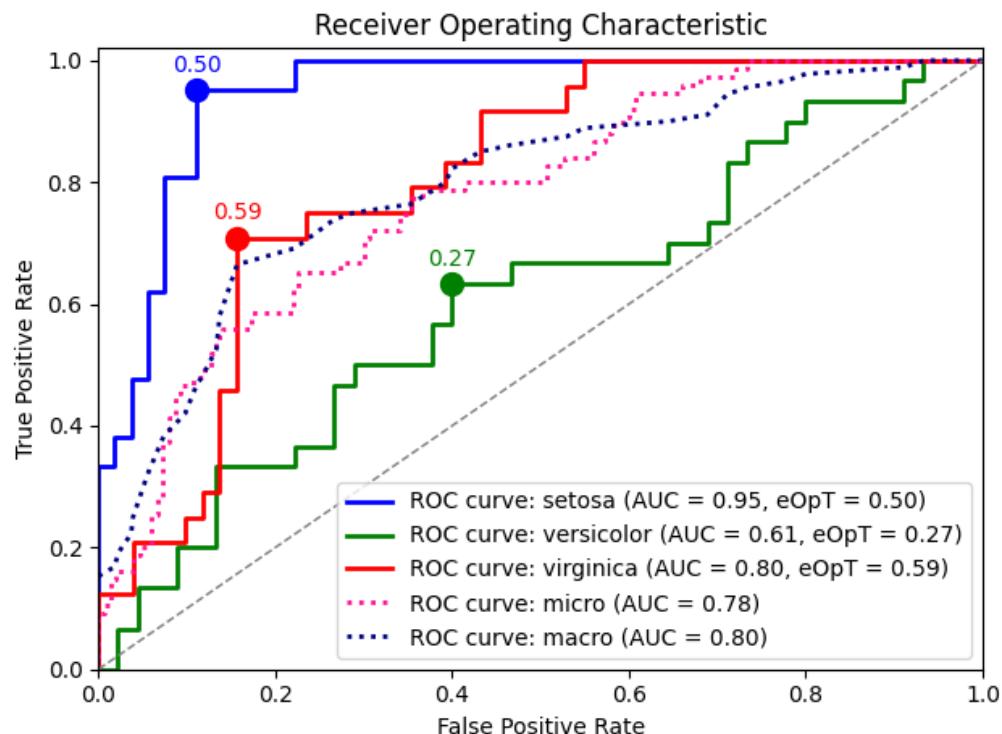


Figure 2: Example of a ROC graph plotted over the Iris Dataset

References

- Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., ... Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825), 357–362. <https://doi.org/10.1038/s41586-020-2649-2>
- Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3), 90–95. <https://doi.org/10.1109/MCSE.2007.55>
- Kotelnikov, A., Baranchuk, D., Rubachev, I., & Babenko, A. (2023). TabDDPM: Modelling tabular data with diffusion models. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu, & S. Sabato (Eds.), *Proceedings of the 40th international conference on machine learning (ICML)* (Vol. 202, pp. 17708–17728). PMLR. <https://proceedings.mlr.press/v202/kotelnikov23a.html>
- 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., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python.

- Journal of Machine Learning Research*, 12, 2825–2830.
- Seabold, S., & Perktold, J. (2010). Statsmodels: Econometric and statistical modeling with Python. In Stéfan van der Walt & Jarrod Millman (Eds.), *Proceedings of the 9th Python in Science Conference* (pp. 92–96). <https://doi.org/10.25080/Majora-92bf1922-011>
- team, P. development. (2025). *Pandas-dev/pandas: Pandas* (Version v2.3.3). Zenodo. <https://doi.org/10.5281/zenodo.17229934>
- Vallat, R. (2018). Pingouin: Statistics in Python. *Journal of Open Source Software*, 3(31), 1026. <https://doi.org/10.21105/joss.01026>
- Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., ... SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 17, 261–272. <https://doi.org/10.1038/s41592-019-0686-2>
- Waskom, M. L. (2021). Seaborn: Statistical data visualization. *Journal of Open Source Software*, 6(60), 3021. <https://doi.org/10.21105/joss.03021>