I feel like writing software documentation is like doing my taxes - HELP!

PyCon Italia 2025

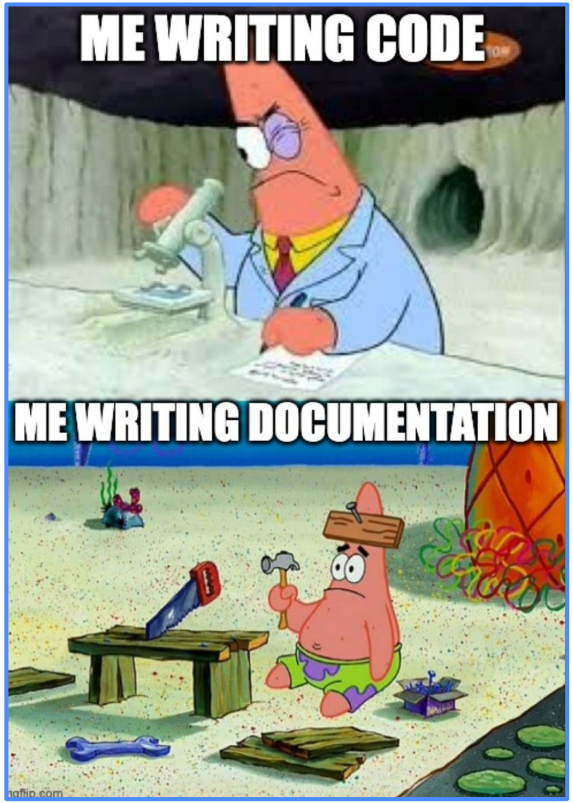
Dr Maren Westermann

Background

“Could you create a documentation on how to write documentation? I'm always in awe with the docs you create.”

— Contributor on Narwhals Discord

Background


ME WRITING CODE
ME WRITING DOCUMENTATION


ME EARNING MONEY
ME DOING MY TAXES
imgflip.com

Table of contents

Documentation - why?

For yourself

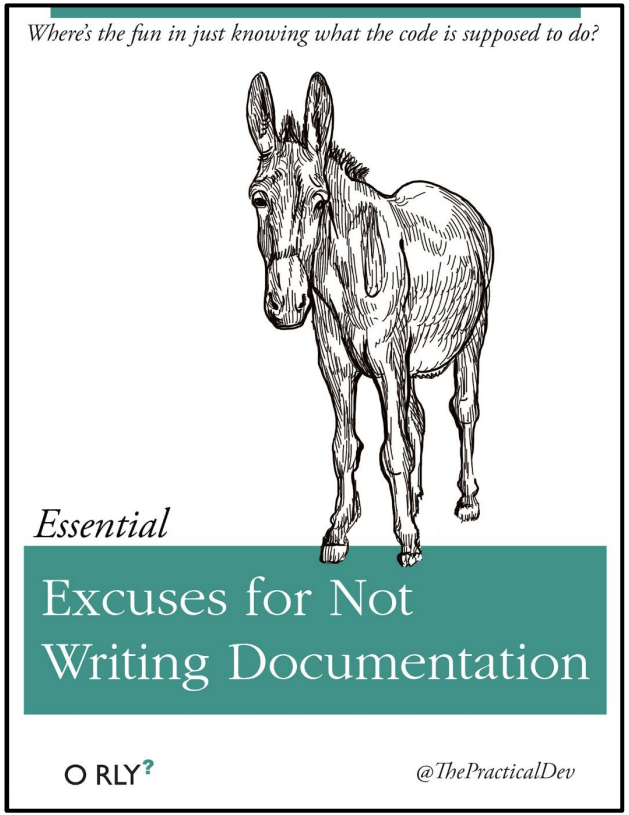
- Make your future self understand your own code
- Improve your technical writing skills

For others

- Explains to users
  - The WHY behind the code
  - Which problem your project solves
  - How to use the code

In general

- Clarifies intentions behind the code
- Good starting point for new contributors



Documentation - how to

Target audience

- Users
- Developers

Overview

- **What** does your code do?
- **Who** is this project for?
- **Why** is this project needed?

Documentation - how to

Home | Search | GitHub

Narwhals

- Home
- Why
- Installation and quick start
- Intro tutorial
- Concepts
- Overhead
- Perfect backwards compatibility policy
- Supported libraries and extending Narwhals
- How it works
- Ecosystem
- Security
- Resources
- API Completeness
- API Reference
- This

Extremely lightweight and extensible compatibility layer between dataframe libraries!

- **Full API support**: cuDF, Modin, pandas, Polars, PyArrow.
- **Lazy-only support**: Dask, DuckDB, Ibis, PySpark, SQLFrame. Work in progress: Daft.

Seamlessly support all, without depending on any!

- ✅ **Just use** a subset of **the Polars API**, no need to learn anything new
- ✅ **Zero dependencies**, Narwhals only uses what the user passes in so your library can stay lightweight
- ✅ Separate **lazy** and eager APIs, use **expressions**
- ✅ Support pandas' complicated type system and index, without either getting in the way
- ✅ **100% branch coverage**, tested against pandas and Polars nightly builds
- ✅ **Negligible overhead**, see overhead
- ✅ Let your IDE help you thanks to **full static typing**, see `narwhals.typing`
- ✅ **Perfect backwards compatibility policy**, see stable api for how to opt-in

Who's this for?

Anyone wishing to write a library/application/service which consumes dataframes, and wishing to make it completely dataframe-agnostic.

Let's get started!

Documentation - how to

Write small reproducible examples

Random Forest Regressor

Examples

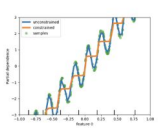
```
>>> from sklearn.ensemble import RandomForestRegressor
>>> from sklearn.datasets import make_regression
>>> X, y = make_regression(n_features=4, n_informative=2,
...                        random_state=0, shuffle=False)
>>> regr = RandomForestRegressor(max_depth=2, random_state=0)
>>> regr.fit(X, y)
RandomForestRegressor(...)
>>> print(regr.predict([[0, 0, 0, 0]]))
[-8.32987858]
```

Documentation - how to

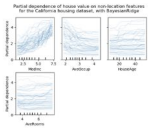
Put more in depth examples in a separate place

Detailed examples about the Random Forest Regressor

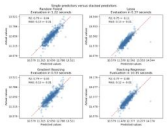
Gallery examples



Release Highlights for scikit-learn 1.4

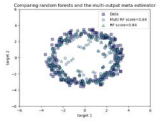


Release Highlights for scikit-learn 0.24

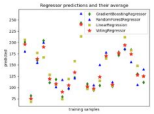


Combine predictors using stacking

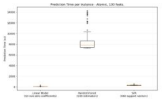
Comparing Random Forests and Histogram Gradient Boosting models



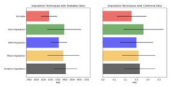
Comparing random forests and the multi-output meta estimator



Plot individual and voting regression predictions



Prediction Latency



Imputing missing values before building an estimator

Documentation - how to

Processes

- Contributing guidelines
- Installation instructions
- Project license

Support

- Issue tracker
- Interaction with the community,
  e.g. mailing list, Discord server

Documentation - how to

Section Navigation

Contributing

This project is a community effort, and everyone is welcome to contribute. It is hosted on scikit-learn/scikit-learn. The decision making process and governance structure of scikit-learn is laid out in Scikit-learn governance and decision-making.

Scikit-learn is somewhat selective when it comes to adding new algorithms, and the best way to contribute and to help the project is to start working on known issues. See Issues for New Contributors to get started.

Our community, our values

We are a community based on openness and friendly, didactic, discussions.

We aspire to treat everybody equally, and value their contributions. We are particularly seeking people from underrepresented backgrounds in Open Source Software and scikit-learn in particular to participate and contribute their expertise and experience.

Decisions are made based on technical merit and consensus.

Code is not the only way to help the project. Reviewing pull requests, answering questions to help others on mailing lists or issues, organizing and teaching tutorials, working on the website, improving the documentation, are all priceless contributions.

We abide by the principles of openness, respect, and consideration of others of the Python Software Foundation: https://www.python.org/psf/codeofconduct/

In case you experience issues using this package, do not hesitate to submit a ticket to the GitHub issue tracker. You are also welcome to post feature requests or pull requests.

On this page

This Page

- Show Source

Documentation - how to

Narwhals | Search | GitHub

Installation and quick start

Installation

UV | Python's venv

First, ensure you have installed UV, and make sure you have created and activated a Python 3.8+ virtual environment.

If you haven't, you can follow our *setting up your environment* guide. Then, run:

```
uv pip install narwhals
```

Verifying the Installation

To verify the installation, start the Python REPL and execute:

```
import narwhals

print(narwhals.__version__)
```

```
1.39.1
```

If you see the version number, then the installation was successful!

Documentation and AI tools

Can AI tools write documentation?

Documentation and AI tools

Original example

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import make_regression
X, y = make_regression(n_features=4, n_informative=2,
                       random_state=0, shuffle=False)
regr = RandomForestRegressor(max_depth=2, random_state=0)
regr.fit(X, y)
print(regr.predict([[0, 0, 0, 0]]))
```

AI generated example

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split

# Generate a synthetic regression dataset
X, y = make_regression(n_samples=200, n_features=4, noise=0.1, random_state=42)

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

# Create and fit the regressor
regr = RandomForestRegressor(n_estimators=100, max_depth=3, random_state=42)
regr.fit(X_train, y_train)

# Predict on the test set
y_pred = regr.predict(X_test)

# Print the R^2 score
print("Test R^2 score:", regr.score(X_test, y_test))
```

Documentation and AI tools

Original param docs

```
n_estimators : int, default=100
    The number of trees in the forest.

    .. versionchanged:: 0.22
       The default value of ``n_estimators`` changed from 10 to 100
       in 0.22.
```

AI generated param docs

```
n_estimators : int, default=100
    The number of trees in the forest.

    Increasing the number of estimators can improve model performance,
    but also increases training time and memory usage. For small datasets,
    a lower value may suffice, while larger values are often beneficial
    for more complex datasets. The default value is 100.

    .. versionchanged:: 0.22
       The default value of ``n_estimators`` changed from 10 to 100
       in 0.22.
```

Documentation and AI tools

Original docstrings

```python
def get_params(self, deep=True):
    """
    Get parameters for this estimator.

    Parameters
    ----------
    deep : bool, default=True
        If True, will return the parameters for this estimator and
        contained subobjects that are estimators.

    Returns
    -------
    params : dict
        Parameter names mapped to their values.
    """
    out = dict()
    for key in self._get_param_names():
        value = getattr(self, key)
        if deep and hasattr(value, "get_params") and not isinstance(value, type):
            deep_items = value.get_params().items()
            out.update((key + "__" + k, val) for k, val in deep_items)
        out[key] = value
    return out
```

AI generated docstrings

```python
def get_params(self, deep=True):
    """
    Get parameters for this estimator.

    Parameters
    ----------
    deep : bool, default=True
        If True, will return the parameters for this estimator and
        contained subobjects that are estimators.

    Returns
    -------
    params : dict
        Parameter names mapped to their values.
        If `deep=True`, will return parameters of nested estimators as well,
        with keys of the form ``<component>__<parameter>``.
    """
    out = dict()
    for key in self._get_param_names():
        value = getattr(self, key)
        if deep and hasattr(value, "get_params") and not isinstance(value, type):
            deep_items = value.get_params().items()
            out.update((key + "__" + k, val) for k, val in deep_items)
        out[key] = value
    return out
```

Documentation and AI tools

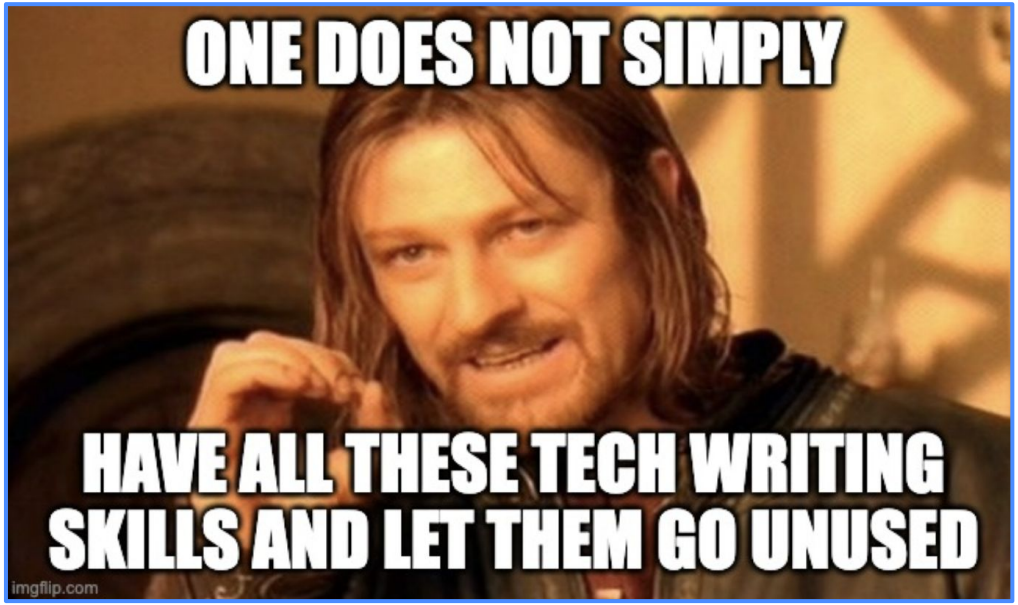
Can AI tools write documentation?

- In clearly defined environments: yes
- Can not go beyond training data and context
  - Can **not be creative** and create new ideas
- AI tools tend to produce a lot of output with sparse information density
  - Don't blindly trust the AI output
- Use AI tools for **assistance**

Documentation and AI tools

Will technical writers die out due to AI?

NO



Have fun!

The Zen of Python

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>>
```

Have fun!

The Zen of Narwhals

```
>>> import narwhals.this
THE ZEN OF NARWHALS
Keep it simple
Move slowly and keep things working
A good API is an honest one
Yes, that needs documenting
People learn better from examples
  than from explanations
If in doubt, better to say 'no'
  than to risk causing a commotion
Yes, we need a test for that
If you want users
  you need good docs
Our code is not irreplaceable
```

Have fun!

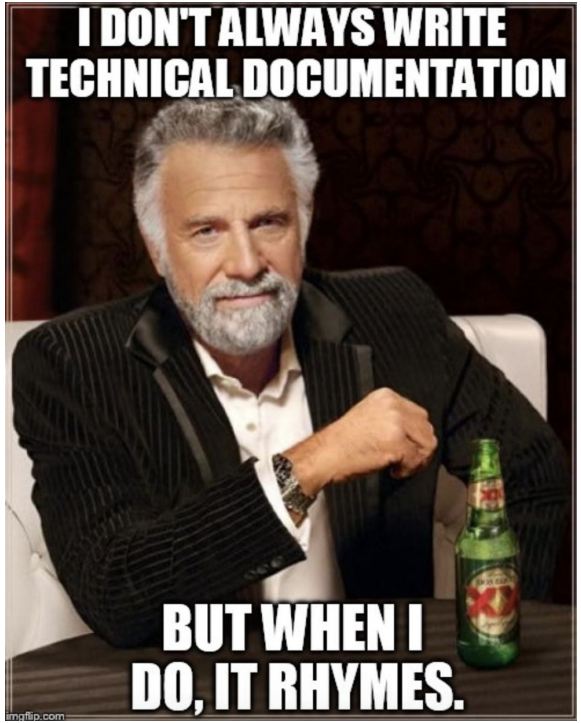
scikit-learn issue #30088

import this

Idea:

Care about the code, every single bit.
Designing a clear API makes it much easier to fit.
When things can break, you want to be strict.
Keep your mind open, the future is very hard to predict.

Suggestions welcome! :)



Further information

Write the Docs

writethedocs.org

Write the Docs conference
27-28 Oct 2025, Berlin

Read the Docs

docs.readthedocs.com

Further information

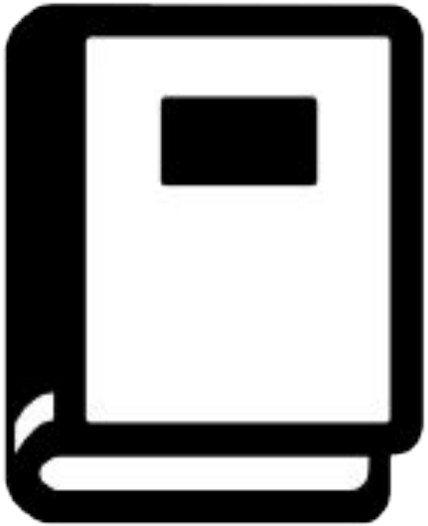
Sphinx

sphinx-doc.org

MkDocs

mkdocs.org

On YouTube:



Thank you

Let's connect :)