Qualidade de código

Por que estamos piorando? E como reverter isso?

Prof. Fabio Kon
Instituto de Matemática, Estatística e Ciência da Computação
USP

Python Brasil 2025 - 24/10/2025

Qualidade de Código

Sempre foi algo que nos preocupou

Má qualidade leva a bugs e má experiência do usuário

Todos nós aqui sabemos muito bem o que é isso

A qualidade melhorou muito ao longo da história

Melhores linguagens

Melhores ferramentas

Melhores processos e técnicas

Mas, recentemente, voltou a piorar

Popularização de Python com não especialistas

Uso massivo em Ciência de Dados e IA

LLMs, que geram código correto de vez em quando

O que é qualidade de software?

Corretude
Eficiência no desempenho
Manutenibilidade

Usabilidade
Segurança

Interoperabilidade
Confiabilidade
Portabilidade

Qualidade Interna de software

Todas as dimensões anteriores podem ser mapeadas aqui

Mas com foco em:

Design
Arquitetura de software
Código-fonte
Documentação

Qualidade de código

Clareza

Concisão

Uso de bons nomes

Modularização

Uso correto do idioma

Uso de padrões

Testabilidade

Clareza

Concisão

Bons Nomes

Modularização

Idioma

Padrões

Testabilidade

Como medir a qualidade do código?

Métricas objetivas

Média de:
- Linhas por método
- Métodos por classe
- Complexidade ciclomática

- Atributos por classe
- Parâmetros por método

Acoplamento
Coesão
Duplicação de Código

Cobertura de Testes

Exemplo: SonarQube

Automatiza a coleta dessas métricas (e várias outras)

Detector de maus cheiros

Open Core
(i.e., núcleo é software livre, *community edition*, mas funcionalidades avançadas são fechadas)

Complexo mas muito poderoso

Alternativas:

Linters e outras ferramentas

Pylint e Flake8 (Python)

ESLint (JavaScript/TypeScript)

PMD (para Java e outros)

Checkstyle (convenções de estilo e deteção de maus cheiros em Java)

Clang Static Analyzer: bom para encontrar bugs em código C/C++ e Objective-C.

semgrep (semantic grep, open core, bom para segurança)

A modern tool for analyzing and visualizing code quality in Git projects. Visualize your codebase through metrics and graphs.

Get Started Learn More

By Yesman Mamani
(BCC-IME/USP)

Key Features

Discover how our platform can help you improve your code quality

Quality Analysis

Monitor your code quality with detailed metrics

Intuitive Visualization

Explore your projects through graphs and other visual resources

Git Integration

Easily import your projects from GitLab or Github

Estudo da literatura e análise de 10 grandes projetos de software livre de Ciência de Dados e Aprendizado de Máquina Aplicado

Catálogo de Antipadrões

https://gitlab.com/interscity/dsmlcodingpatterns

10 Anti-padrões (em código DSML)

- High Cyclomatic Complexity
- Unnecessary Iterations
- Duplicated Code
- Unclear Names
- Too Many Arguments
- Too Many Local Variables
- Too Many Instances Attributes
- Too Many Statements
- Low Class Cohesion
- High Class Coupling

https://gitlab.com/interscity/dsmlcodingpattern

UNNECESSARY ITERATIONS

DESCRIPTION

Iterating manually through a Pandas DataFrame or Series to process data is generally not recommended when there are more efficient built-in methods available to perform the same task.

PROBLEMATIC CODE

```
import pandas as pd

def main():
        data = {'Date': pd.date_range(start='2023-01-01',
                                      period=5,
                                      freq='D'),
            'Close': [100, 102, 101, 105, 107]}
        df = pd.DataFrame(data)
        calculate_percentage_changes(df)


def calculate_percentage_changes(df):
        percentage_changes = []
        for i in range(1, len(df)):
                previous_close = df['Close'].iloc[i - 1]
                current_close = df['Close'].iloc[i]
                pct_change = ((current_close - previous_close) / previos_close) * 100
                percentage_changes.append(pct_change)
```

SOLUTIONS

1. Utilize efficient built-in methods in Pandas to perform the same operation.

REFACTORED CODE

```
import pandas as pd

def main():
        data = {'Date': pd.date_range(start='2023-01-01',
                                      period=5,
                                      freq='D'),
            'Close': [100, 102, 101, 105, 107]}

        df = pd.DataFrame(data)
        calculate_percentage_changes(df)


def calculate_percentage_changes(df):
        df['Pct_Change'] = df['Close'].pct_change() * 100

# pct_change() is a buil-in method from DataFrame class to provide the fractional change from
# the immediatly previous row (by default). To get the percentage change, just multiply the value by 100.
```

CONCLUSION

Iterating through a Pandas object can lead to performance issues when working with large datasets. Instead, always look for efficient solutions that are already implemented in Pandas.

Projects

Manage and monitor your development projects

+ Add Project

EconML

No description available

View Project >

SocialHubAPI

No description available

View Project >

FinRL

No description available

View Project >

OVERVIEW MODULES QUALITY SETTINGS

EconML - Overview

Comprehensive analysis of code quality, structure, and metrics

85% Quality Score

9.51% Duplicated Code

20 Avg Lines/Function

141 Avg Lines/Class

Most Recurrent Code Smells

1	Too Many Arguments	253
2	Too Many Statements	198
3	Too Many Locals	186
4	Too Many Instance Attributes	53
5	Too Complex	51

Most Problematic Modules

1	_dr.py econml/iv/dr/_dr.py	48
2	_ortho_forest.py econml/orf/_ortho_forest.py	29
3	linear_model.py econml/sklearn_extensions/linear_model.py	25
4	test_statsmodels.py econml/tests/test_statsmodels.py	24
5	_drlearner.py	

Project Totals

43,884 Total Lines

741 Total Issues

2144 Total Functions

312 Total Classes

Module Analysis

Detailed breakdown of code structure and quality metrics per module

Module Distribution

Analysis

131 Total Modules

335 Avg Lines/Module

5.7 Avg Issues/Module

Quality Distribution

Health

- Excellent (115)
- Good (16)
- Needs Work (0)

FILE NAME	CODE SMELLS	CLASSES	FUNCTIONS	LINES	QUALITY SCORE
_dr.py econml/iv/dr/_dr.py	48	15	100	2,544	83%
_ortho_forest.py econml/orf/_ortho_forest.py	29	6	49	1,164	82%
linear_model.py econml/sklearn_extensions/linear...	25	16	69	1,704	88%
test_statsmodels.py econml/tests/test_statsmodels.py	24	14	56	1,110	90%

Quality Score Settings for EconML

Customize how quality scores are calculated for EconML. These settings determine how code metrics affect the overall quality score calculation.

Weight Factors

Weight factors determine the importance of each metric in the quality score calculation. The sum of all weights must equal 1.0 for proper calculation.

Code Issues Weight

0.6

Importance of code issues in the quality score

Class Size Weight

0.25

Importance of class size in the quality score

Function Size Weight

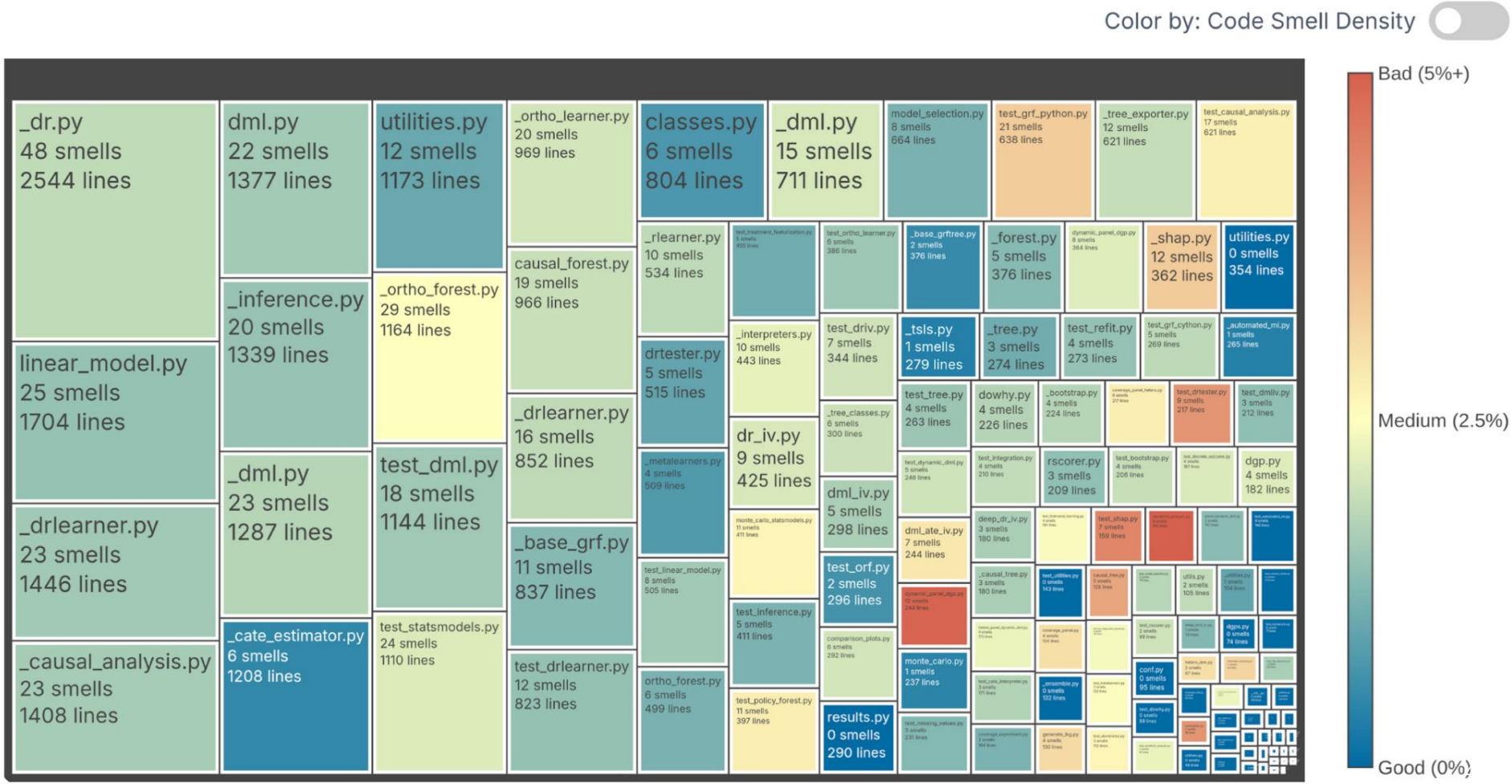
0.15

Importance of function size in the quality score

Weight Sum: 1.00

Normalization Factors

Normalization factors balance the relative impact of each metric in the quality score calculation, ensuring that different measurement scales contribute proportionally to the final assessment.


Color by: Code Smell Density
Bad (5%+)
Medium (2.5%)
Good (0%)
_dr.py
48 smells
2544 lines
dml.py
22 smells
1377 lines
utilities.py
12 smells
1173 lines
_ortho_learner.py
20 smells
969 lines
classes.py
6 smells
804 lines
_dml.py
15 smells
711 lines
model_selection.py
8 smells
664 lines
test_grf_python.py
21 smells
638 lines
_tree_exporter.py
12 smells
621 lines
test_causal_analysis.py
17 smells
621 lines
linear_model.py
25 smells
1704 lines
_inference.py
20 smells
1339 lines
_ortho_forest.py
29 smells
1164 lines
causal_forest.py
19 smells
966 lines
_rlearner.py
10 smells
534 lines
drtester.py
5 smells
515 lines
_drlearner.py
16 smells
852 lines
_metalearners.py
4 smells
509 lines
_dml.py
23 smells
1287 lines
test_dml.py
18 smells
1144 lines
_base_grf.py
11 smells
837 lines
test_linear_model.py
8 smells
505 lines
_drlearner.py
23 smells
1446 lines
_cate_estimator.py
6 smells
1208 lines
test_statsmodels.py
24 smells
1110 lines
test_drlearner.py
12 smells
823 lines
ortho_forest.py
6 smells
499 lines
_causal_analysis.py
23 smells
1408 lines
test_treatment_featurization.py
5 smells
455 lines
test_ortho_learner.py
6 smells
386 lines
_base_grftree.py
2 smells
376 lines
_forest.py
5 smells
376 lines
dynamic_panel_dgp.py
8 smells
364 lines
_shap.py
12 smells
362 lines
utilities.py
0 smells
354 lines
_interpreters.py
10 smells
443 lines
test_driv.py
7 smells
344 lines
_tsls.py
1 smells
279 lines
_tree.py
3 smells
274 lines
test_refit.py
4 smells
273 lines
test_grf_cython.py
5 smells
269 lines
_automated_ml.py
1 smells
265 lines
_tree_classes.py
6 smells
300 lines
test_tree.py
4 smells
263 lines
dowhy.py
4 smells
226 lines
_bootstrap.py
4 smells
224 lines
test_drtester.py
9 smells
217 lines
test_dmliv.py
3 smells
212 lines
dr_iv.py
9 smells
425 lines
dml_iv.py
5 smells
298 lines
test_dynamic_dml.py
5 smells
246 lines
test_integration.py
4 smells
210 lines
rscorer.py
3 smells
209 lines
test_bootstrap.py
4 smells
206 lines
dgp.py
4 smells
182 lines
monte_carlo_statsmodels.py
11 smells
411 lines
dml_ate_iv.py
7 smells
244 lines
deep_dr_iv.py
3 smells
180 lines
test_shap.py
7 smells
159 lines
test_orf.py
2 smells
296 lines
_causal_tree.py
3 smells
180 lines
test_utilities.py
0 smells
143 lines
utils.py
2 smells
105 lines
test_inference.py
5 smells
411 lines
dynamic_panel_dgp.py
12 smells
244 lines
comparison_plots.py
6 smells
292 lines
monte_carlo.py
1 smells
237 lines
conf.py
0 smells
95 lines
test_policy_forest.py
11 smells
397 lines
results.py
0 smells
290 lines
_ensemble.py
0 smells
133 lines
test_dmliv.py
0 smells
88 lines

prototypes/orthogonal_forests/comparison_plots.py

Line 174

```
142     def generic_joint_plots(file_key, dfs, labels, file_name_prefix):
172         plt.cla()
173
174     def metrics_subfig(dfs, ax, metric, c_scheme=0):
175         if c_scheme == 0:
176             palette = plt.get_cmap('Set1')
177         else:
178             palette = plt.get_cmap('tab20b')
179         if metric == "bias":
180             biases = np.zeros((len(dfs[0]), len(dfs)))
181             for i, df in enumerate(dfs):
182                 treatment_effects = df[[c for c in df.columns if bool(re.search('TE_[0-9]+', c))]]
183                 bias = np.abs(np.mean(treatment_effects, axis=1) - df["TE_hat"])
184                 biases[:, i] = np.abs(np.mean(treatment_effects, axis=1) - df["TE_hat"])
185             vparts = ax.violinplot(biases, showmedians=True)
186             ax.set_title("Bias")
187         elif metric=="variance":
188             variances = np.zeros((len(dfs[0]), len(dfs)))
189             for i, df in enumerate(dfs):
190                 treatment_effects = df[[c for c in df.columns if bool(re.search('TE_[0-9]+', c))]]
191                 variance = np.std(treatment_effects, axis=1)
192                 variances[:, i] = np.std(treatment_effects, axis=1)
```

Occurrences: dml_ate_iv.py ↔ dr_iv.py

prototypes/dml_iv/dml_at... Lines 59-69

```
16  class DMLATEIV:
50      def fit(self, y, T, X, Z):
55          T : treatment (single dimensional)
56          X : features/controls
57          Z : instrument (single dimensional)
58          """
59          if len(Z.shape) > 1 and Z.shape[1] > 1:
60              raise AssertionError("Can only accept single
61          if len(T.shape) > 1 and T.shape[1] > 1:
62              raise AssertionError("Can only accept single
63          if len(y.shape) > 1 and y.shape[1] > 1:
64              raise AssertionError("Can only accept single
65          Z = Z.flatten()
66          T = T.flatten()
67          y = y.flatten()
68
69          n_samples = y.shape[0]
70          res_t = np.zeros(n_samples)
```

DUPLICATE

prototypes/dml_iv/dr_iv.py Lines 236-247

```
188 class DRIV(_BaseDRIV):
193     def __init__(self, model_Y_X, model_T_X, modelst
228         super(DRIV, self).__init__(nuisance_models, nt
233         return
234
235     def _check_inputs(self, y, T, X, Z):
236         if len(Z.shape) > 1 and Z.shape[1] > 1:
237             raise AssertionError(
238                 "Can only accept single dimensional instr
239         if len(T.shape) > 1 and T.shape[1] > 1:
240             raise AssertionError(
241                 "Can only accept single dimensional treat
242         if len(y.shape) > 1 and y.shape[1] > 1:
243             raise AssertionError("Can only accept single
244         Z = Z.flatten()
245         T = T.flatten()
246         y = y.flatten()
247         return y, T, X, Z
```

Próximos passos

Disponibilizar DeSmell num servidor web

Uso de IA (LLMs) para avaliação semi-automática de qualidade de código em parceria com humano

Montar um curso online (Coursera) sobre qualidade de código para não-especialistas

Prof. Fabio Kon
IME-USP
www.ime.usp.br/~kon

https://www.coursera.org/learn/ciencia-computacao-python-conceitos
https://www.coursera.org/learn/ciencia-computacao-python-conceitos-2

https://www.coursera.org/learn/lab-poo-parte-1
https://www.coursera.org/learn/lab-poo-parte-2