UNIVERSITY OF AMSTERDAM

UvA-DARE (Digital Academic Repository)

Programmed differently? Testing for gender differences in Python programming style and quality on GitHub

Brooke, Siân

DOI
10.1093/jcmc/zmad049

Publication date
2024

Document Version
Final published version

Published in
Journal of Computer-Mediated Communication

License
CC BY

Link to publication

Citation for published version (APA):
Brooke, S. (2024). Programmed differently? Testing for gender differences in Python programming style and quality on GitHub. *Journal of Computer-Mediated Communication*, *29*(1), Article zmad049. https://doi.org/10.1093/jcmc/zmad049

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: https://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, P.O. Box 19185, 1000 GD Amsterdam, The Netherlands. You will be contacted as soon as possible.

UvA-DARE is a service provided by the library of the University of Amsterdam (https://dare.uva.nl)

Download date:05 Sept 2026

Journal of Computer-Mediated Communication, 2024, zmad049
https://doi.org/10.1093/jcmc/zmad049
Full-length Research Article
Special Issue: Gender Gaps in Digital Spaces

Programmed differently? Testing for gender differences in Python programming style and quality on GitHub

Siân Brooke [1,*]

[1]Department of Methodology, London School of Economics, Houghton Street, London, WC2A 2AE, United Kingdom
*Corresponding author: Siân Brooke. Email: s.j.brooke@lse.ac.uk

Abstract

The underrepresentation of women in open-source software is frequently attributed to women's lack of innate aptitude compared to men: natural gender differences in technical ability (Trinkenreich et al., 2021). Approaching code as a form of communication, I conduct a novel empirical study of gender differences in Python programming on GitHub. Based on 1,728 open-source projects, I ask if there is a gender difference in the quality and style of Python code measured in adherence to PEP-8 guidelines. I found significant gender differences in structure and how Python files are organized. While there is gendered variation in programming style, there is no evidence of gender difference in code quality. Using a Random Forest model, I show that the gender of a programmer can be predicted from the style of their Python code. The study concludes that gender differences in Python code are a matter of style, not quality.

Lay Summary

This study examines whether there is a difference in Python programming styles between gender groups. I examine available code on GitHub, a cloud-based hosting platform for collaboration known as version control, often used in open-source software development. First, I infer the gender of users from their usernames and the information provided on their profiles, labeling users as feminine, masculine, ambiguous, and anonymous. Anonymous users had no gender-based markers on their profiles, while ambiguous users had feminine and masculine characteristics. I then collect the publicly available projects of these users written in Python. Next, I analyze and generate statistics on Python files' adherence to style guidelines using a linter, an automated checking of source code for programmatic and stylistic errors. My findings reveal a gendered difference in the structure and components of Python files. However, I also discovered no gender difference regarding violations of Python style guidelines and code quality. This study shows gender difference in Python programming styles but not in the standard or quality of the code.

Keywords: gender, open-source software, computational methods, intersectional and feminist approaches, HCI.

Despite its title, Open-Source Software (OSS) has remained essentially closed to women, who comprise less than 10% of its contributors. Even in communities with few or no barriers to entry, such as GitHub, women are severely underrepresented across programming languages. Women represent 4.8% of core developers for projects based on Python, 4.5% for C++, and 4.2% for Java (Trinkenreich et al., 2021). Yet, the positive effects of gender diversity have drawn attention from researchers and practitioners. Greater representation of women benefits productivity, leading to improved outcomes, increased problem-solving capacity, greater creativity, and a healthier work environment (Vedres & Vásárhelyi, 2022). The evidence for the benefits of greater gender inclusion in OSS is overwhelmingly positive. So why are women still so underrepresented?

The absence of women in OSS is frequently attributed to a lack of innate aptitude compared to men: a natural "gender difference" in technical ability. This sexist belief perpetuates women's exclusion as it frames their absence in computing as inevitable (Frieze & Quesenberry, 2019). In education, the gender difference approach presumes that the computer science curriculum should be changed ("made pink") to suit women's computing deficiencies (Frieze & Quesenberry, 2019). Moreover, stereotypes of gender and technical expertise can extend beyond metrics of knowledge, where it is assumed that even the code itself is inscribed with gendered norms. Carter and Jenkins (2002) conducted preliminary research into coding differences, finding that many teachers believed they could guess a student's gender from their code. I draw on and expand this study to empirically test if a gender difference is present in Python code in OSS development on GitHub.

Writing readable code is fundamental to OSS. Python is a popular programming language that allows complex applications while prioritizing concise and readable syntax. Such *high-level programming languages* use natural-language elements designed to be easily understood by human readers and computers. For example, in Python, the command *print()* is used to output data. While programming is built on mathematical logic, its writing, reading, and interpretation involve linguistic frameworks vulnerable to gendered evaluation, as high-level languages resemble familiar text structures. Programming in such languages requires syntax, definitions, statements, and arguments, which include scope for stylistic variation. Like natural-language, elements of high-level programming languages can be susceptible to interpretation and judgment.

In natural-language, preconceptions about gender affect the interpretation of specific linguistic features and intentions (Lindvall-Östling et al., 2020). In written communication, for

Associate Editors: Sandra González-Bailón and Emőke-Ágnes Horvát
Received: 3 April 2023. **Revised:** 13 November 2023. **Accepted:** 20 November 2023
© The Author(s) 2024. Published by Oxford University Press on behalf of International Communication Association.
This is an Open Access article distributed under the terms of the Creative Commons Attribution License (https://creativecommons.org/licenses/by/4.0/), which permits unrestricted reuse, distribution, and reproduction in any medium, provided the original work is properly cited.

Downloaded from https://academic.oup.com/jcmc/article/29/1/zmad049/7596747 by Universiteit van Amsterdam user on 24 March 2026

instance, a firm statement of disagreement may be seen as strong if the author is a man. In contrast, the same statement could be interpreted as cold and cutting from a woman. Scholars have speculated that the same is true for code. Marino (2020) considers how the gender of a programmer influences the assumption of "strength" and code quality. I hypothesize that code is a form of computer-mediated communication (CMC) and is as gendered as written and spoken language. Informed by computational sociolinguistics, this project investigates whether gendered differences exist in the material code. I ask: Is there a gendered difference in Python programming style?

The structure of this article is as follows. I first outline the related literature, beginning with how automated recognition can reinforce stereotypes. Building from the literature, I present the specific research questions and associated hypotheses. Next, I summarize the novel methodology of this study, detailing how I analyzed Python code by identifying subtle programming errors and unconventional coding practices through *linting* and assessing adherence to style guidelines. The analysis also measures gender differences in *modular programming* by focusing on the building blocks of code in the content of lines in Python files. The study's findings empirically challenge the justification for women's exclusion from OSS based on gendered differences in programming quality. I conclude that assumptions on women's lack of ability in programming are not empirically grounded.

Related literature

Extensive research has been conducted into gender differences in the linguistic style of CMC (Liu et al., 2023). Flanagin (2020) suggests that CMC researchers should focus beyond new features of specific technologies, instead emphasizing the underlying, exceptional, mechanisms of mediation. Open-source development is one such mechanism, a form of online collaboration that requires reading and reviewing computer code produced by others (Brock, 2019; Holohan & Garg, 2005). In illustrating how code is communication, Marino (2020) considers a scenario where a man and a woman are presented with a popular programming challenge: to write a program that computes an anagram of a *string* (a sequence of characters) as part of a job interview. Given two technically different answers, one simple and one complex, the "stronger" solution is based on interpretation by the reader. The complex answer may be understood as sophisticated or needlessly idiosyncratic, the simple as more concise or reflecting limited knowledge. The gender of the assumed author would meaningfully impact such interpretation and assessment of quality (Marino, 2020). As a communication system, cultural norms extend beyond a programming task's specifics, and cultural norms mediate code interpretation.

Gender and text

Like human readers, machines also assume authors' gender based on text interpretation. In practicing computational inference of social categories, it is crucial to understand the politics of identity. Keyes (2018) surmises that automatic gender recognition treats gender as binary, immutable, and physiological, which has long been considered inaccurate. They show how, in everyday interactions offline, individuals infer gender based on posture, dress, and vocal cues and "then justify it with physiological cues after the fact" (2018, p. 3). In such interactions, gender is not *recognized* or *identified* but *inferred* (Keyes, 2018). If programmers are not vigilant to gender biases, programmed tools can inherit and recreate gender stereotypes. In her examination of gender data bias, Criado-Perez (2019) shows how the search term "computer programmer" can direct views to a male programmer's website simply because a search algorithm has inferred that masculine pronouns are more closely associated with the term "computer programmer,' thus websites with masculine pronouns are more relevant to the search query. The same association was not true for feminine pronouns (pp. 166–167). Therefore, much work on computational language models is focused on undoing such learning or *debiasing* algorithms, meaning removing potentially discriminatory practices or power structures embedded in prejudiced language that automated systems learn from. In their seminal paper on debiasing word embeddings in natural-language processing, Bolukbasi et al. (2016) show that models trained on various sources exhibit strong gender stereotypes, which the increasing use of automated tools can amplify. Therefore, researchers on the inference of gender need to be aware of its limitations and be careful not to claim to discover truths or to reinforce gender stereotypes.

Gender diversity in programming teams is essential to challenging biases coded into automated tools (Brooke, 2021). Despite plentiful and valuable research on the absence of women in programming, empirical evidence for gender differences in code is mainly anecdotal. Yet examining gender differences in natural-language is a robust field, as a prominent aim of sociolinguistics is to understand the relationship between social variables and the use of language (Nguyen et al., 2016). Stylistic variations in writing are a key focus of gender inference of authors or users of online platforms (Nguyen et al., 2016). Language is a resource for individuals to author their own flexible gendered identity (Bucholtz & Hall, 2005). An individual's gender identity is constructed using linguistic features associated with masculine or feminine speech and writing. Scholars have identified elements associated with feminine speech that reflect a submissive social position, such as uncertainty verbs, hedging, minimizers, and the use of specific punctuation (e.g., "!") (Doughman et al., 2021). These features reflect the subordinate position of feminine identities in broader social structures, mediated by role adoption in particular settings such as the workplace (Bucholtz & Hall, 2005). It is plausible that similar linguistic patterns are apparent in code, where granular details reveal a gendered difference in writing.

Until now, the focus of gender inference has been on written or spoken language. Natural-language is often defined in opposition to computer code. Nonetheless, Mackenzie (2005) argues for a communicative analysis of code itself, extending a speech-based notion of code as instruction to include the mediated practices of programming, configuring, and running a computational project. Here, code is an "objectification of a linguistic praxis" (p. 76), where code is understood not just in terms of explicit meaning but also as self-reflexive and mediated by the individuals, interactions, and culture that created it. From this perspective, code is written communication with meaning and interpretation, where one may make assumptions about the identity of the author based on the form of code itself.

Gender stereotypes associated with natural-language may be applicable to code. Carter and Jenkins (2002, p. 7) saw

Downloaded from https://academic.oup.com/jcmc/article/29/1/zmad049/7596747 by Universiteit van Amsterdam user on 24 March 2026

that despite the prevalence of sexism in computing, code written by "female" students was labeled by educators as "neater" and well-organized with consistent formatting. Such descriptions imply a gendered ideology, where feminine stereotypes are gentle, sympathetic, and *neat* compared to dominant, loud masculinity (Koenig, 2018). While programming is built on logic, its writing and interpretation involve linguistic frameworks, words, and syntax. As a high-level language, Python can resemble familiar text structures. Programming in such languages requires syntax, definitions, statements, including scope for stylistic variation. As a language, programming thus has structural markers comparable to natural-language that may make it susceptible to stereotypes if evidence is found of gendered variation.

Coding, linting, interacting

OSS emphasizes that computers and humans should easily read and understand code. The basis of OSS is that code is made freely available and is open to modification and redistribution (Trinkenreich et al., 2021). If code is not *readable*, it discourages use, erecting barriers to participation in OSS (Viafore, 2021). *Linters* are a shortcut to facilitate readability, analogous to advanced spelling and grammar checkers in natural writing and word processing. The term *linting* refers to the use of static analysis tools ("linters") to detect bugs and other issues ("lint") in software programs (Viafore, 2021). The term initially referred to a Unix utility, where the command *lint* would examine C source programs, focusing on broad compatibilities (Johnson, 1978). While modern C and C++ compilers have lint-like functions, lint-like tools are useful for languages like JavaScript and Python, which are dynamically typed (without needing a compiler). A compiler translates a programming language into machine code or another low-level programming language a computer can understand. Because compilers typically do not enforce strict rules prior to execution, linters can be used as simple debuggers for finding common errors (e.g., *syntactic discrepancies*) as well as hard-to-find errors such as *heisenbugs*, drawing attention to suspicious code as possible errors (Viafore, 2021). Linters can be as pedantic as spell checkers, highlighting issues with style and potential mistakes and facilitating the reading and sharing of code in OSS.

Beyond eliminating errors, defining *readability* in OSS can be difficult, given that it depends on the individual capacity to read a given text or section of code. To enhance understanding, linters rely on standards, representing endless collaboration to develop agreed style conventions for a particular language (Viafore, 2021). They are collaboratively authored by the community involved in developing a language. In Python, the most consensual writing style is defined by the PEP-8 standard, which the popular linting tool *Pylint* enforces (Rother, 2017). PEP-8 evolves over time as additional conventions are identified, and past approaches are rendered obsolete by changes in the language itself. Also crucial within the convention is the notion that many projects have their own coding style guidelines, and in the case of conflict, such project-specific guides take precedence for that project (Rother, 2017). Since being written in 2001, PEP-8 is intended to improve the readability of code and make it consistent across the broad spectrum of Python applications (van Rossum, 2023). As the Python principles state: "readability counts" (Peters, 2004). Therefore, conformity to style guidelines is not a matter of finesse but is necessary for code to be read, reused, and remixed.

Several recent social studies have explicitly included linting in their research design. There is a particular trend in the literature to examine the possible benefits of linting as a pedagogical tool. Obermüller et al. (2021) study how the criticism produced by linting can be complemented by positive feedback. They introduce the concept of *code perfumes* as the counterpart to code smells (or warnings). Perfumes indicate the correct application of programming practices considered exemplary rather than exclusively highlighting issues and potential bugs. Similarly, Farah et al. (2022) examine chatbots (*Lint Bot*) as a teaching instrument in lessons on enforcing coding standards in JavaScript. Bart et al. (2021) also developed a tool in Python that can semi-automate student feedback on coding assignments. In these studies, the *linter* is implicitly positioned as an agent in the interactive process of producing and evaluating code. Social interactions are fundamental to success when students are learning to program. However, with the feedback provided by linting, the programmer/user interacts and is in conversation with the linter, giving feedback. The work of Obermüller et al. (2021) highlights this, as they implicitly position the feedback from the linter as a social agent. The feedback provided by the linter to the programmer during writing has been included as a research instrument in pedological studies.

Gender, programming, and GitHub

Like linting, *version control* is an essential tool for programmers in OSS. Also known as source control, version control is the practice of tracking and managing changes to software code. Git is a distributed version control system; it can be complex and intimidating, so GitHub is frequently used for uploading and managing copies of a Git Repository (or "repo"). Purchased by Microsoft in 2018, GitHub is a remote collaboration platform that allows users to work together or independently on technical projects, expanding the possibilities of programming by creating libraries and technological tools. It can track changes made to files, launch software, and host websites. There are over 100 million developers on GitHub, 20.5 million of which joined in 2022 alone (Dohmke, 2023). GitHub has remained at the forefront of OSS and technological development, hosting Interactive Developer Environments, including the most popular source code editor, Microsoft's VS Code. Despite the growth and popularity of GitHub, only 24.5% of programmers on the site are women.

Valuable studies have documented the nuances of gender discrimination in GitHub as a collaborative coding platform. Terrell et al.'s (2017) study of gender effects in proposed changes to a software project's code, documentation, or other resources on GitHub found that women's contributions are accepted more often than men's if their identity is obscured. However, when gender is made visible, women's contributions are 15% less likely to be accepted (Terrell et al., 2017). Similarly, Vedres and Vásárhelyi (2019, p. 1) found that "disadvantage is a function of gendered behaviour" on the open-source development platform GitHub. "Femaleness" was qualified by variables such as professional ties, level of activity (push/pull requests), and areas of specialization (Vedres & Vásárhelyi, 2019). The study argues that measures of reputation ("success"—as starred repositories) and survival ("time account active") on the platform were adversely

Downloaded from https://academic.oup.com/jcmc/article/29/1/zmad049/7596747 by Universiteit van Amsterdam user on 24 March 2026

affected by femaleness rather than by categorical discrimination. They saw that not only was this true for women, but men and users with unidentifiable gender are also likely to suffer for exhibiting behavior that demonstrates "femaleness." Vedres and Vásárhelyi (2019) conclude that implicit sexism punishes feminine behavior that adversely affects one's status and collaboration prospects, as opposed to explicit sexism penalizing femininity in GitHub usernames. This study's gendered analysis of the code hosted on GitHub will contribute to the existing work on gender differences in platform interactions, as highlighted by Terrell et al. (2017) and Vedres and Vásárhelyi (2019).

Expectations of gender difference in code impact the programmer's skill assessment. As outlined by Marino (2020), the reviewing of code quality is mediated by context-specific factors. The assumed gender of programmers in an anonymous setting can affect how a technical solution is evaluated: basic or concise, excessive or complex (Marino, 2020). As highlighted earlier, Carter and Jenkins (2002) show that computing educators often believe they can guess a student's gender from their code. However, this too is open to the same conflicting gendered interpretation that Marino (2020) outlines. For instance, they show that for one code snippet half the educators assumed it had been written by a woman, with its neatness and lack of confidence in "thinking aloud" comments being widely cited as reasons (Carter & Jenkins, 2002). The same justification was used by other educators, emphasizing the content and extent of the comments as the reason they believed the author was a man (Carter & Jenkins, 2002). Given those rationalizations for gender differences in code are contradictory and grounded in cultural sexism that discriminates against women, there is little theoretical reasoning in the literature as to why specific code features would be tied to gender expression. Expanding Carter and Jenkins's (2002) research to include contemporary OSS tools, I expect to find no significant gender difference in adherence to PEP-8 guidelines.

Building from the literature, I test if variation in Python code can be attributed to gender. This project complements studies on gender bias in activity and interactions in OSS by showing that exclusion does not result from gender difference in code quality. Therefore, I chose to research questions that are relevant to the style and structure of Python files rather than differences in activity and recognition. For technical accuracy, I use the language of Python *modules*, referring to files with the ".py" extension containing Python code that performs a specific task. I ask: (RQ1) Is there a gender difference in module structure? (RQ2) Is there a gender difference in the style of Python modules? (RQ3) Can the gender of a module's author be predicted from Python programming structure and style? This study empirically tests if there is gender difference in Python code, a rhetoric that is often used to rationalize women's exclusion from OSS development.

Methods

This project leverages large-scale data through public coding repositories in Python on GitHub. Python is the second most popular programming language on GitHub, growing 22.5% in popularity in the year 2022 (Dohmke, 2023). Figure 1 shows the data collection, processing, and analysis strategy. It also includes the relevant files for each stage of the analysis, as contained in this article's GitHub repository. I collected all GitHub repositories that meet the criteria below, suggesting an initial sample size of 1.1 million users.

- *The owner is the only contributor*. To avoid complexity in unpicking specific contributions in collaborative work, I only collected repositories where the owner was the sole contributor.
- *Written in Python3*. Restricting to one version of a programming language permits greater validity.
- *The repository was created on or after January 1, 2019*. Python2 was deprecated on January 1, 2020. Python3 was already in circulation, so the collection was backdated a year.
- *Not marked as private or archived and publicly visible*. I did not consider it ethical to include repositories that the owner had listed as private.
- *Not a fork*. Repositories are not forks or initialized as copies of other repositories.
- *The repository is between 0.01 and 1 gigabyte in size*. As of 2022, GitHub documentation recommends that repositories remain less than 1GB.
- *Owned by a User*. Repositories with the owner as a sole user rather than an "Organisation" as a shared account.

Interaction is a crucial feature of open source and as such, software is defined by the potential to share. Therefore, I narrowed the selection of repositories to include those with 10–250 *forks* (a new repository that shares code with the original) and 10–150 *stars* (like bookmarking or favoriting). While I focus on repositories with only one author, a minimum of 10 forks and stars represents a reasonable engagement with OSS communities. For example, if a repository has 10 forks on GitHub, it has been copied and shared 10 times by other users. These selection criteria thus reflect that the Python code sampled was shared and engaged with, including an upper limit that excluded outliers in popularity that may skew the results. This range demonstrates repositories that fit our criteria for inclusion (specified above) and are sufficiently interacted with to be included in an analysis of gender differences in programming in open source. Nonetheless, GitHub does not make statistics on the overall distribution of forks and stars publicly available, which means that the specific generalizability of the sample is difficult to obtain. In total, I collected 1,728 repositories consisting of 30,198 modules or ".py" files.

Gender inference procedure

As GitHub users do not list gender on their profile, I inferred each person's gender using their location and name markers. I followed Vedres and Vásárhelyi's (2019) selection of data points, using full names (provided names), nicknames (usernames), and email addresses (prefixes) offered by users on their profiles. I also incorporated Brooke's (2021) expansion of Vasilescu et al.'s (2015) *genderComputer*, growing the library of names it uses to infer gender. I also expanded genderComputer's pre-processing of usernames to include "leet speak" or substituting a word's letters with numbers or special characters. I used the Python library *GeoPy* to clean users' location data to assist with gender inference, as noted by Vasilescu et al. (2015). Working with the GitHub API, I aimed for an approximately even distribution of masculine and feminine Python modules to assist simplicity in statistical tests for gender differences.

Downloaded from https://academic.oup.com/jcmc/article/29/1/zmad049/7596747 by Universiteit van Amsterdam user on 24 March 2026

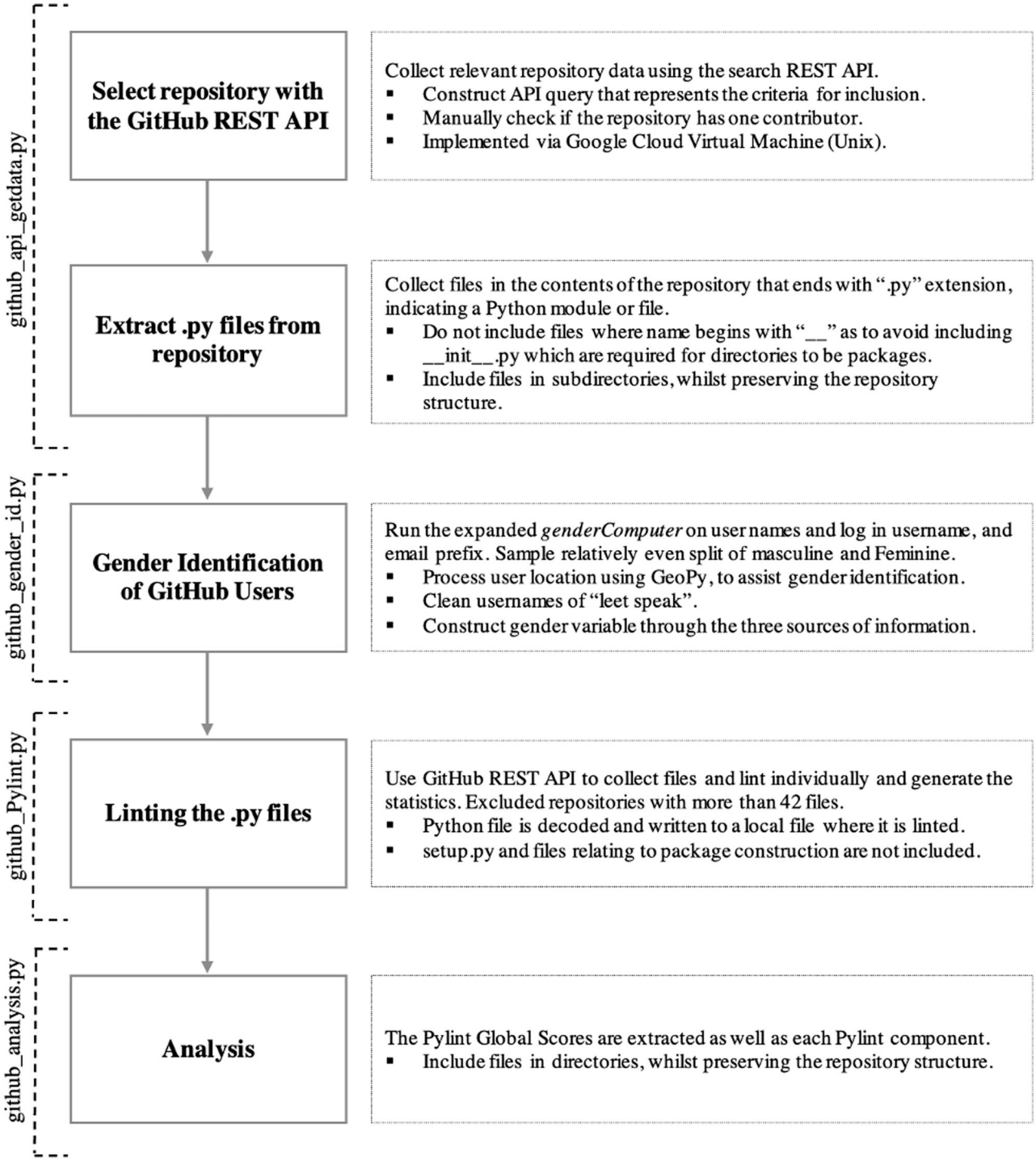

Figure 1. Research design and analysis strategy.

The results of the gender inference procedure are displayed in Table 1. I initially considered including biographical ("about me") data provided on user profiles. However, there were little to no explicit markers of gender here. Users generally included their job title ("Developer," "Student," "Researcher") and specific technical interests ("Machine Learning," "AI," "Engineering"). Following Vedres and Vásárhelyi (2019), I assess the accuracy of the inferences compared to a manually coded baseline using the same features (username, real name, email). I took a random sample of 850 users from the dataset. There were 105 cases where I disagreed with the gender inference, meaning an 87% agreement. I utilized Krippendorff's alpha, a quantification of intercoder reliability, using statistics of agreement.[1] Considering four gender categories—*feminine, masculine, ambiguous*, and *anonymous*—the alpha was 0.72. Considering feminine and masculine users only, the alpha was 0.87. These statistics demonstrate good reliability.

I labeled users as masculine, feminine, ambiguous, and anonymous. The "ambiguous" label was given to users with significant gender markers, but it was unable to statistically infer the masculine or feminine categories with sufficient

Downloaded from https://academic.oup.com/jcmc/article/29/1/zmad049/7596747 by Universiteit van Amsterdam user on 24 March 2026

Table 1. Count of repositories and files

Gender	Repositories/users	.py Files/modules
Feminine	774	10,122
Masculine	483	9,991
Ambiguous	248	5,050
Anonymous	223	5,035
Total	1,728	30,198

confidence (0.4-0.8).[2] Considering the insights of Keyes (2018), I intend to understand gender in a nuanced way, with an analysis that includes the inference of gender beyond a binary. I differentiate between "ambiguous" users, where gender markers were present, and "anonymous" users, where they were not.[3] Further, I use the language of "masculine" and "feminine" to represent how gender is enacted and maintained through *performance* in language. Butler (1999) emphasizes the importance of gender in interaction, meaning how gender is recognized in the specific context in which it is performed. Gender is ultimately an issue of a broader culture, not specifically an issue for women (Frieze & Quesenberry, 2019). By incorporating multiple sources of profile data, I am not expanding my gender inference sources beyond what information is readily available to other GitHub users. For an interactive site illustrating how these measures of programming style work on Python modules, see the link in the Data Availability section.

Style and code linting

The operationalization of communication in empirical work is a challenge to researchers. A focus on the structure of language in specific features provided by *linters* can appear simplistic or superficial. However, the features outputted from linters represent conforming to normative syntax and semantics, which ultimately helps others work with the code and see errors more easily. Work on popular programming forums shows that women face more severe social sanctions than men for minor technical infractions, such as adherence to style guidelines (Brooke, 2021). Furthermore, linting is fundamental in the mechanism of open-source communication and collaboration (Obermüller et al., 2021). Pylint is often employed in developing OSS, as code that follows convention is easier to understand and therefore, other programmers can repair, maintain, or build on the work. By focusing on gender differences in adherence to PEP-8, this study focuses on the initial assumptions and tests of quality in OSS.

As Pylint analyses code without running it, it is particularly suited to the large-scale task of this study. Note that *refactor* and *convention* will not cause the code files or modules to stop running. As stated, *modules* are files with the ".py" extension containing Python code that can be imported (or used) inside another Python program. Table 2 shows examples of the Pylint checkers, module components, and associated examples (Pylint, 2023). This is not an exhaustive list, as Pylint version 3.0.0a6 can currently produce 452 different "messages," which are collated into style checker components for this study.[4]

Global Pylint Score

Pylint enforces the standard, producing the metric *Global Pylint Score*. The maximum score for adhering to PEP-8 is a Pylint score of 10, but there is no lower bound, and the score can be negative if there is a heavy infringement of the Python style standards (Pylint, 2023). The size of the given Python code weights the errors through the number of statements in each Python file. Statements are the smallest standalone element of code that produces an outcome. The formula for the score is:

$$\textit{Global Pylint Score} = 10 - \left(\frac{5 \times \textit{error} + \textit{warning} + \textit{refactor} + \textit{convention}}{\textit{statement}} \times 10 \right)$$

where:

- ***error:*** The total number of errors in a module. An error is an issue in a program that prevents the program from completing its task. For example, E0401 or "import-error" is used when Pylint has been unable to import a module.
- ***warning:*** The total number of warning style checks generated. Warnings indicate potential future errors; they are non-critical and do not terminate the program.
- ***refactor:*** The total number of refactoring errors. Refactoring is restructuring code intended to improve the software's design while preserving its functionality. For example, if several lines of code are repeated in a file, they can be *refactored* into a *user-defined function* where a custom-made function can be called instead of copying and pasting code. Pylint would produce style check R0801 (duplicate-code).
- ***convention:*** The total number of format-checking style checkers where the code is in violation of PEP-8. For instance, in its current implementation, PEP-8 suggests lines should be limited to 79 characters to allow you to easily open multiple files next to each other. In Pylint, this will produce the C0301 (line-too-long) style check.
- ***statement:*** The total number of lines of Python code in the module. These are *unwrapped* (not made to fit a viewable window), without *docstrings* (used to document the purpose of code without detailing the implementation; not runnable, beginning with:""""), *comments* (used to enhance readability; not runnable, beginning with #), or blank lines.

The formulation of Pylint scores to include the number of lines of code permits comparison across different-sized *modules* as they are relative to each file weighted by size. According to Python style guidelines, project-specific guidelines take precedence over PEP-8, represented as *convention* (van Rossum, 2023). If the specific style check convention was present across more than half of the .py files in the GitHub repository, it was not included as a violation in the computing of the global Pylint score. I identified outliers with a Pylint score below −100, which were not included in the data.

Testing for gender differences

I define *gender difference* as statistically significant variation between gender groupings. Table 3 outlines the operationalization of metrics used to test for gendered difference. Overall,

Downloaded from https://academic.oup.com/jcmc/article/29/1/zmad049/7596747 by Universiteit van Amsterdam user on 24 March 2026

Table 2. Example of Pylint style checker components

Pylint checker	Style checker component	Example style checker message[1]
Basic checker	Error	E0104 (return-outside-function): Used when a return statement is found outside a function or method
	Convention	C0103 (invalid-name): Used when the name does not conform to naming rules associated with its type (constant, variable, class, etc.)
	Warning	W0101 (unreachable): Used when some code is behind a return or raise statement, which will never be accessed
Design checker	Refactor	R1260 (too-complex): Used when a method or function is too complex based on McCabe Complexity Cyclomatic
Format checker	Convention	C0303 (trailing-whitespace): Used when there is a whitespace between the end of a line and the new line
	Warning	W0311 (bad-indentation): Used when an unexpected number of indentations, tabulations, or spaces has been found
Refactoring checker	Refactor	R1702 (too-many-nested-blocks): Used when a function or a method has too many nested blocks. This makes the code less understandable and maintainable. The maximum number of nested blocks for the function/method body is five by default
	Convention	C0201 (consider-iterating-dictionary): Emitted when a dictionary's keys are iterated through the .keys() method. It is enough to just iterate through the dictionary, as in for key in the dictionary
[...]		

[1] Rationale from https://pylint.readthedocs.io/en/stable/index.html

I opted for simplicity in testing. Given specific technical terminology and the potentially broad implications of findings, I prioritized the interpretability of the results. I tested using Welsch's ANOVA and Chi-squared tests. If the test is significant, the test statistic tells us about the ratio of between-group variation to within-group variation. A considerable value means the between-group variation is larger than the within-group variation. In my study, this means there is more discrepancy between gender groups than within each gender group, providing evidence for a *gendered difference*.

Using statistical inference methods, I am testing against a hypothesis of no effect or relationship based on gender. Testing for gender differences in module structure (RQ1), the null hypotheses are $H0_1$: *There are no gender differences in module organization* and $H0_2$: *There are no gender differences in module constituents*, referring to the building blocks of Python such as functions. Next, I examine if there is a gender difference in the style of Python modules (RQ2), with the associated null hypotheses, $H0_3$: *There is no gender difference in global Pylint scores*, and $H0_4$: *There is no relationship between gender and style checker components*. To conduct significant testing with the large sample size, I used bootstrapping, a statistical procedure that resamples a single dataset to create many simulated samples. Following the advice of Davidson and MacKinnon (2000), I set the confidence level at 0.01 and drew samples of 3,000 with replacement.

Addressing RQ3, I examine if the gender of the author can be predicted from Python coding style. Significance testing is not suitable for this task. Instead, I used a Random Forest model with cross-validation ($k = 10$) to predict gender identity using the module structure and Pylint style checker components. I also considered the inclusion of additional features, such as the longevity of an account and user-level activity metrics. Statistical analysis based on information retrieved from websites and platforms via APIs can often include such metrics by default due to their ease of access. However, I opted to retain the focus on code as communication.

Care is taken to ensure that the findings are not obscured by specific technical terms related to Python as a programming language, so relevant definitions are provided. These definitions may be superfluous for the more technically inclined. Nonetheless, definitions are included to assist in the interpretation of the results as presented. Additionally, it is crucial that attempts to infer user gender from code are not misappropriated for purposes of discrimination and that inference of femininity does not incur negative judgment. To the best of my knowledge, the methodology of this study is unique in applying computational text analysis to the material code itself. As with any new approach to inferring identity from online and technical data not intended for the purpose, ethical implications must be considered. This study is fundamentally a work of feminist activism that challenges assumptions of gendered differences used to maintain a sexist structure in OSS.

Table 3. Metrics to test for gender difference

Theme	Concept	Operationalization
Module structure	Organization	Lines of *code*, *docstrings*, *comments*, and *empty* lines in a module
	Constituents	Frequency of *classes*, *functions*, and *methods* in a module
Module style	Pylint Score	The *global Pylint score* is a measure of PEP-8 adherence
	Style Checker Components	Style checker messages are collated into the components of *information*, *error*, *refactor*, *warning*, and *information*

Results

Module structure

RQ1 asks if there is a gender difference in module structure, focusing on the *organization* (lines of code, code, documentation) and *constituents* (methods, functions, classes) of Python modules. I tested for significant gendered differences using ANOVA, which enabled the comparison of more than two groups simultaneously to determine whether a relationship exists between them. I tested for departures from

Downloaded from https://academic.oup.com/jcmc/article/29/1/zmad049/7596747 by Universiteit van Amsterdam user on 24 March 2026

assumptions of normality using Bartlett's test. As the test was significant, I rejected the null hypothesis and concluded that not all gender groups have the same variance, and therefore, I used Welch's ANOVA. Table 4 shows the mean (μ) and standard deviation of each gender grouping and the *f*-value and significance. Note that these values are rounded, as a fraction of a line of code is not meaningful.

First, I look at differences in organization by the average number of lines in each module by gender grouping. *Total lines* here refer to the complete length of the module, including empty lines. Table 4 shows that masculine users have the largest average module size by a reasonable margin of 43 lines. I found a significant gender difference in the total number of lines between gender groups. Next, I look at the average count of lines of Python code within each module. The metric is typically used to indicate a given file's size and predict the effort required to modify it to prevent future problems, otherwise called *maintainability*. I find that masculine GitHub users have more lines of code on average than other gender groups, followed by feminine users. This finding is logical, as masculine users also have the greatest total lines. Therefore, I reject the null hypothesis and find a significant gender difference in the average number of lines of code by gender grouping.

I also look at organizational differences in the number of lines that refer to the *documentation* of code in comments and docstrings (Table 4). Commenting involves placing human-readable descriptions inside modules that detail what the code is doing. In Python, this is achieved with the "#" symbol. Proper commenting can simplify code maintenance and help find errors and issues faster. I then look at docstrings, a short form for "documentation string," a type of multi-line comments frequently used in Python. They are accessible when the block of code they relate to is used elsewhere, an essential part of PEP-8. The tests show a significant gender difference in the count of comments and docstrings lines between gender groupings, with masculine users having more lines of docstring on average. Masculine users have the same frequency of comment lines as anonymous and feminine users despite having significantly longer files by total line count. This suggests that masculine users are more likely to document their code with docstrings rather than comments.

Like docstrings, blank or empty lines are essential to PEP-8. Like blank lines to indicate paragraphs in written text, in Python, blank lines indicate logical sections and improve readability. I also find a gendered difference in the count of empty lines between gender groupings. I see significant gender differences in total lines, lines of code, documentation, and blank lines. I reject the null hypothesis $H0_1$: *There are no gender differences in module organization.* Compared to other gender groups, masculine users have longer files and prefer docstrings to comments. The *f-stat* shows the greatest variation in module organization between all genders is docstrings, which, given the importance of documenting to PEP-8, suggests that there may be a gender difference in Python code styles.

I also tested for gender differences in module constituents ($H0_2$), the portable Python code blocks that make up the module. In Python, a *function* is a labeled block of code that performs a specific task. A *method* is like a function but for a particular type of information. For example, *.lower()* makes text (known as a string) lowercase but does not work on numeric data. *Classes* are more complex but can be understood as a blueprint for organizing information and its manipulation. Table 5 indicates the mean and standard deviation in the count of functions, modules, and classes between gender groupings and Welch ANOVA for significance. Table 5 shows a similar average number of functions per module between gender groupings. Welch ANOVA found that the between-gender variation was larger than the within-gender variation, meaning there is a significant gender difference in function count. I also found a significant difference in gender groupings between the average number of methods and classes in Python modules. Consequently, I reject the null hypothesis $H0_2$: *There is no gender difference in the module constituents*.

Table 4. Gender differences in module organization by lines

Gender	Total lines		Code		Docstring		Comment		Empty	
	μ	Std	μ	Std	μ	Std	μ	Std	μ	Std
Feminine	281	334	194	218	19	49	24	40	45	41
Masculine	324	296	212	215	32	53	24	36	56	80
Ambiguous	253	421	169	290	24	68	20	40	41	72
Anonymous	279	333	190	237	25	66	24	51	41	47
Welch ANOVA F-stat	49.72**		45.09**		62.23**		34.63**		64.05**	

* $p < .01$. **$p < .001$.

Table 5. Gender differences in module constituents

Gender	Function		Method		Class	
	μ	Std	μ	Std	μ	Std
Feminine	3.82	5.71	5.41	9.26	1.39	2.46
Masculine	3.92	8.08	8.34	12.94	1.26	1.96
Ambiguous	3.64	3.64	6.47	15.81	1.51	3.28
Anonymous	3.62	6.39	6.46	11.13	1.43	2.78
Welch ANOVA F-stat	2.89*		64.82**		14.68**	

* $p < 0.01$. **$p < 0.001$.

The largest gender difference is in the use of methods, with masculine users utilizing methods the most frequently and feminine the least. There is less variation by gender for functions and classes, but anonymous and ambiguous users employ classes more regularly than gender-identified (masculine, feminine) users. The most minor variation between gender groups is with functions, with masculine and feminine users having a relatively higher average count. These findings could suggest that users whose gender is known take a more functional approach to programming in comparison to anonymous and ambiguous users who use more classes. These findings merely indicate such a difference as the analysis did not directly assess functional approaches to programming. Therefore, there is a significant difference in total lines, module organization (docstring, comment, and empty lines), and module constituents (functions, methods, classes). Addressing RQ1, I find a significant gendered difference in the structure of Python modules in terms of organization and constituents.

Global Pylint score and style checker components

The second research question asks if there is a gender difference in Pylint scores and style checker components. While the

Downloaded from https://academic.oup.com/jcmc/article/29/1/zmad049/7596747 by Universiteit van Amsterdam user on 24 March 2026

Table 6. Description of Pylint scores by gender (repository)

Gender	Count	μ	Std	Min	25%	50%	75%	Max
Feminine	774	1.20	10.77	−70.00	0.31	4.18	6.22	10.00
Masculine	483	0.46	9.27	−70.00	−1.17	3.44	5.83	10.00
Ambiguous	248	1.28	8.18	−78.67	−0.61	3.34	5.75	10.00
Anonymous	223	1.27	8.82	−80.00	−0.57	3.54	5.92	10.00

previous section analyzed the module or .py file level, the Pylint score is analyzed at the repository level to control for project-specific style conventions. Table 6 shows the descriptive statistics of the global Pylint scores by each gender, summarizing the central tendency and dispersion.

The mean shows that masculine-identified users have the lowest average Pylint scores (0.46), while feminine (1.20), ambiguous (1.28), and anonymous (1.27) users have similar scores. This suggests that, on average, masculine users violate the PEP-8 guidelines more frequently than other users. The maximum Pylint score of 10 out of 10 was achieved across all gender groupings. The standard deviation indicates a reasonable spread of Pylint scores for each gender group. I conducted a Welch's ANOVA and found no significant difference in global Pylint scores by gender groupings (*f-stat* = 0.51, $p = .69$). I thus accept the hypothesis that *$H0_3$: There is no gender difference in Pylint scores*, and gender does not explain variation in adherence to PEP-8.

I ran Welch's t-test on the Pylint scores for differences between (1) masculine and feminine users, (2) gender-labeled, and (3) non-identified users. First, I found no significant gendered difference in Pylint scores between masculine and feminine users (*t-stat* = −0.48, $p = .65$). Second, I found no significant difference in Pylint scores (*t-stat* = 1.17, $p = .23$) between gender-identified (feminine, masculine) and non-gender-identified users (ambiguous, anonymous). I surmise that there is no significant gendered difference in global Pylint scores. Consequently, although gender differences in organization and code constituents suggested gender differences in style, gender does not explain variation that contravenes the PEP-8 guidelines.

Next, I examined the granular gendered differences in the Pylint style, testing *$H0_4$*. First, I focus on the individual *messages* grouped into the *style checker components*. These messages are the specific issues raised by Pylint when analyzing the code in the dataset. Of the possible 452 style checkers, 192 were present. Style checker components indicate potential problems with the code, including errors where the script would not execute. The five most frequent style checkers were the same for each gender group, and their rationale is shown in Table 7. This frequency is at the level of the .py file and does not include cumulative counts of an error within a module.

Expanding the analysis to the 10 most frequent style checker messages, the single discrepancy between gender groupings was that feminine users had *consider-using-from-import* rather than *trailing-whitespace* as violations of PEP-8. This suggests that there will not be a significant gender difference in programming style related to style checker components. As outlined in Table 2, the style checker components (collection of messages) are *error*, *convention*, *warning*, *refactor*, and *information*. Mindful of the potential for gender differences to emerge in subcategories of Pylint, I also tested for gender-based variations in style checker components. The information checker is not included in the global Pylint score and refers to how the code in the module handles information, such as explicitly suppressing warning messages. I did not include the information checker in the chi-squared analysis due to insufficient data points, as shown with feminine users in the Information style checker component.

Table 7. Most frequent topics

Style check	Rationale
missing-module-docstring	At least one .py file in the repository does not have text documenting its purpose at the module's start (first few lines)
invalid-name	Used when a name does not fit the naming convention associated with its type (constant, variable, class …) and/or is inconsistent across the repository
missing-function-doctstring	A function or method has no docstring. A docstring is a text so programmers can understand what it does without reading the implementation details
line-too-long	PEP 8 suggests lines should be limited to 79 characters. This is because it allows you to open multiple files next to one another while avoiding line wrapping
missing-class-docstring	A class has no docstring. A docstring is a text so programmers can understand what it does without reading the implementation details

Table 8 shows the frequency, mean, and standard deviation of style checker components by gender category. First, comparing gender-identified users, I find that masculine users have more warnings and violations of PEP-8 conventions than other gender groups. Masculine users also have the second most errors on average, after anonymous users. This is logical as masculine users had the lowest global Pylint score, though gender difference in Pylint was insignificant. I conducted a chi-squared test for association between the style checker components and gender grouping. The resulting test was significant, meaning I rejected *$H0_4$* and found a significant gendered difference in style checker components.

I conducted multiple 2×2 Chi-square tests using the Bonferroni-adjusted p-value ($a = 0.008$). Post-hoc testing revealed significant gender differences for the style checker components for all the pairwise comparisons, excluding one. The exception was the comparison of feminine and anonymous users. This being the only insignificant comparison indicates a comparable coding style between feminine and anonymous users. The similarity in programming styles suggests that women may purposefully obscure their gender and choose to be anonymous. This proposition is supported by the relatively large and significant difference in style between masculine and anonymous users.

The most considerable variation is between ambiguous and anonymous users. This implies meaningful gender differentiation between anonymous and ambiguous users, supporting the methodological choice to distinguish between the presence and absence of features in gender identification.

The outlined results offer a nuanced response to RQ2. The absence of significant variation between global Pylint scores shows no significant gender difference in the overall *quality*

Downloaded from https://academic.oup.com/jcmc/article/29/1/zmad049/7596747 by Universiteit van Amsterdam user on 24 March 2026

Table 8. Style checker components contingency table

Gender	Style checker components				
	Convention	Error	Information	Refactor	Warning
Feminine	38,570 ($\mu = 5$, std = 12)	7,310 ($\mu = 4$, std = 5)	4 ($\mu = 1$, std = 0)	6,498 ($\mu = 2$, std = 2)	24,866 ($\mu = 8$, std = 17)
Masculine	35,420 ($\mu = 8$, std = 27)	5,730 ($\mu = 8$, std = 15)	388 ($\mu = 8$, std = 28)	4,698 ($\mu = 2$, std = 3)	27,633 ($\mu = 15$, std = 27)
Ambiguous	50,824 ($\mu = 5$, std = 12)	8,387 ($\mu = 4$, std = 5)	73 ($\mu = 1$, std = 0)	7,519 ($\mu = 1$, std = 1)	44,222 ($\mu = 11$, std = 65)
Anonymous	52,979 ($\mu = 5$, std = 12)	24,877 ($\mu = 10$, std = 71)	210 ($\mu = 4$, std = 5)	8,088 ($\mu = 2$, std. = 2)	31,767 ($\mu = 8$, std = 36)

X^2 :12,957.53**
Degrees of freedom: 12

* $p < 0.01$. ** $p < 0.001$.

of Python code. However, there is evidence of a gender difference in *style*. The style checker components of the Pylint score differ significantly by gender. Such a finding is evidence of Simpson's paradox, whereby an association between gender and programming disappears when the style checker components are combined into the global Pylint score. I find that while there is no gender difference in code quality, there is evidence of gendered styles in Python. These results indicate that it is feasible to predict the gender of users from the style of their code.

Testing for gendered coding identification

In the final element of the analysis, I addressed RQ3 and tested if structure and style could be used to classify users' inferred gender. I used a Random Forest model to predict gender identity, using structural constituents and style checker components of Python modules. When performing the classification, each feature vector element was used as a separate input for the classifier. I did not include the global Pylint score as it is a composite measure of the included style checker components. Randomized hyperparameter search selected a maximum depth of 14 (longest path from the root node to leaf node) and 393 estimators (the number of trees to build). Visualizing an entire tree with a depth of 14 is impractical. Therefore, Figure 2 represents a small tree with a depth of three.

For comparison with existing literature, I restricted the prediction to the labels of masculine and feminine. I then used this model to predict the gender of the anonymous users in my dataset. As ambiguous users were defined by the presence of gender markers, including them in the classification model was not sensible.

To evaluate my classification model, I first established a reference level to assess the Random Forest model. I used a Zero Rate Classifier (ZRC), which always classifies to the largest class. The ZRC has a baseline of 0.62 based on gender inference of users. I measured the performance of the classification models using the F1 Score, a function of Precision and Recall. *Precision* is how accurately the model performs regarding predicted positives, meaning the model correctly classifies modules as feminine and masculine. *Recall* calculates how many actual feminine data points that model captures (True Positives). The F1 score seeks a balance between these two metrics. The F1-score achieved by my model was 0.98, successfully outperforming the base model (0.62) in classifying the gender inferred from module authors. Addressing RQ3, I find that the gender of Python modules can be predicted based on programming style.

Feature importance

The principal advantage of building the classification model is that I can compare the relative importance of module organization, constituents, and style checker components. The Random Forest model allows us to ascertain how attributes of a Python module contribute to the model prediction. Table 9 indicates the total 13 input variables used as features to infer gender. The overarching categories here are (1) *module organization*, the total count of lines in a .py file and those that are docstrings, comments, code, and empty; (2) module *constituents*, the count of functions, methods, and classes in each .py file; and (3) style checker, the *components* of the Pylint style checkers including error, warning, refactor, convention, and information.

Table 9 shows that module organizational features are the most important in differentiating between masculine and feminine authors of Python code. The number of lines of code and docstrings in a module is particularly important, representing 17% and 14% of the classification decision. Empty lines are also noteworthy, accounting for 16%. The importance of these structural elements (75%) shows that how a module is organized is a crucial marker of gender difference in Python programming. Similarly, the use of module constituents is an essential contributor to gender difference (24%), but the style checker at (<1%) leads to the conclusion that gender differences in coding styles are manifested in how code is organized within a module. Given structure is considerably more critical to the classification decision than the style checker components, gender difference does not indicate a violation of PEP-8.

Discussion

This study has argued that code is a form of communication in OSS. Code in OSS shares features with written text: It has authors and is read, understood, and shared. Contributing code is fundamental to open source, yet code interpretation is mediated by cultural norms and context that extend beyond the specifics of a particular programming task. Code is embedded in a system of communication, meaning that assessments of quality are intrinsically tied to the perceptions of the author. Gender and masculinity have been powerfully associated with quality in programming, suggesting that notions of gender difference have become widespread without empirical evidence.

This study asked whether there is a gender difference in module structure and style in Python code and whether gender can be predicted from programming style. The first element of module structure I examined was organization. I

Downloaded from https://academic.oup.com/jcmc/article/29/1/zmad049/7596747 by Universiteit van Amsterdam user on 24 March 2026

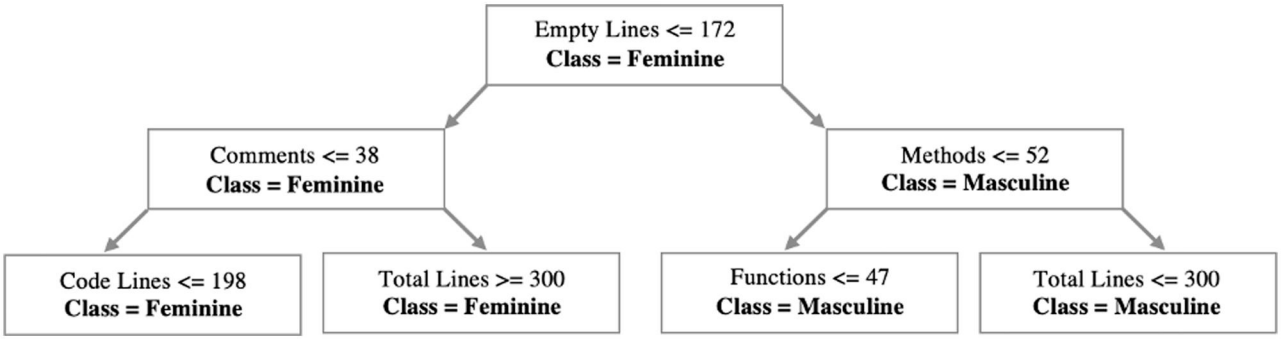

Figure 2. Representation of a decision tree with a maximum depth of 3.

Table 9. Feature importance for gender classification of Python modules

Variable		Importance
Module organization	Code lines	0.171
	Empty lines	0.160
	Total lines	0.159
	Docstring lines	0.138
	Comment lines	0.124
Module constituents	Method	0.094
	Class	0.058
	Function	0.083
Style checker components	Information	0.001

found a significant gender difference in the length of a file and empty lines, documentation, and code. The test statistic revealed that the greatest variation in module organization by gender was docstrings, a crucial part of PEP-8. The second part of the module structure was constituents, referring to methods, functions, and classes. I again found significant gender differences between all groups, with functions used more by gender-identified users than anonymous and ambiguous users. Therefore, I see significant gender differences regarding organization and constituents in module structure.

The second research question turns to gender differences in Pylint scores and style checker components, which align with code quality captured in PEP-8. Although there is no significant gender difference in the global Pylint score, there is a difference in the style checker components of Python repositories. This shows that while there is no gendered variation in overall quality, there is variation in module style. Post-hoc testing revealed that feminine and anonymous users have comparable coding styles, implying that feminine users may choose to be anonymous on technical platforms. This suggestion is maintained by the relatively large and significant difference in quantified style between masculine and anonymous users. Additionally, the largest variation in style is between ambiguous and anonymous users, supporting the design decision to distinguish between the presence and absence of features in gender identification (Keyes, 2018). In summary, while there is no gender difference in overall code quality, there is in checker components and thus Python programming style.

Finally, I test if structural and stylistic features of Python code can be used to predict if a user is masculine or feminine. I show that variation in coding style is distinct enough to predict users' gender. Crucially, the predictive model also allows comparison between the importance of module structure and style checker components as elements of gendered programming style. The structure of Python modules, consisting of the organization of lines and module constituents, is the most critical predictor of user gender (0.752). The style checker components of Python code, related to the PEP-8 guidelines, are relevant (0.235) but pointedly less important than the structure of files. I find that gender differences in coding style can predict the gender of an author of a Python module.

The results reveal a somewhat complex picture of gender differences in Python. I find a gender difference in the *style* of Python code, as evidenced by significant gender-based variation in the organization and constituents of modules. However, the non-significant difference in Pylint scores indicates no gender-based differences in code *quality*, as represented in the global Pylint evaluation. The significant gender difference in style checker components, but not the global Pylint score, suggests that different gender groups vary in their specific violations of PEP-8 but not in the overall quality of the code. This supposition is supported by the predictive model, which shows that the structure of Python modules is a more important predictor of user gender than style violations. I find empirical evidence of gender differences in Python code in relation to style but no substantial evidence of a gender difference concerning code quality. I conclude that assumptions on women's lack of ability in programming are not empirically grounded.

There are three key limitations of this study. First, the sample criteria of the study do not encompass the fullness of practice in OSS development. As I restricted the sample to GitHub repositories with a singular author, I omitted the normative OSS practice of multiple authorship of modules and technical projects (Vedres & Vásárhelyi, 2019). Second, an equally noteworthy omission is that I do not include the mechanisms of reviewing code on GitHub, a vital platform functionality. Third, the methodology also fails to acknowledge the heterogeneity of purpose and function in repositories that makes Python such a popular programming language. This study has provided a broad and general analysis of gender differences in Python modules. Future scholarship should incorporate a refined picture sensitive to implicit gender biases in the collaboration and functioning of OSS.

My findings complement work on gender biases in activity and interaction on GitHub by demonstrating no significant gender difference in code quality (Terrell et al., 2017; Vedres & Vásárhelyi, 2019). The results have important consequences for computing education and policy as well as for interventions in gender inequality in OSS development. In conjunction with the literature, my findings show that educators and employers must actively challenge specific gender stereotypes in context, including mistaken assumptions about

Downloaded from https://academic.oup.com/jcmc/article/29/1/zmad049/7596747 by Universiteit van Amsterdam user on 24 March 2026

ability inferred from differences in style (Brock, 2019; Carter & Jenkins, 2002). Further, there is potential for measures of gender difference to impact generative coding models (Bart et al., 2021; Brooke, 2021). As we take care to de-bias natural-language processing and large language models like GPT-3, similar care should be taken to ensure that generative coding does not suffer from the same or similar issues. As automated systems become increasingly dominant and have the potential to write code, we must consider what value systems and implicit identity features are being learned. Future research should explore the linkages and relationships between natural-language and programming code in gendered terms. Computational tools are uniquely positioned to document such large-scale gendered differences and account for how they can shape and mold everyday technical tools. For Marino (2020, p. 33), "the walls of a computer do not remove code from the world but encode the world and human biases." This study has illustrated the potential of computational research to challenge gender bias and sexist reasoning for women's exclusion from programming.

Data availability

The data used in this study are available upon reasonable request to the author. The source code is available here: https://github.com/SianJMBrooke/ProgrammedDifferently. An interactive site to test the gendered style of your own Python code is available at https://www.sianbrooke.com/programmed-differently.

Funding

This work was supported by the Leverhulme Trust Early Career Fellowship, grant number ECF-2021-272.

Conflict of interest: None declared.

Open science framework badges

Open Materials

The components of the research methodology needed to reproduce the reported procedure and analysis are publicly available for this article.

Notes

1. https://pypi.org/project/krippendorff/.
2. Vasilescu et al. (2015) refer to this category as "unisex," but such language is not appropriate in discussions of people or users.
3. The label "Non-Binary" was initially considered instead of "Ambiguous." However, the Non-Binary gender identity requires more careful consideration in computational research than reflecting mixed or undetermined gender expression.
4. https://Pylint.pycqa.org/en/latest/user_guide/messages/messages_overview.html

Acknowledgements

I am grateful to the Leverhulme Trust for supporting this research project. I thank Theodora Sutton, Maximilian Steimel, and Marion Lieutaud for their unwavering support. Finally, I would like to thank the editors and three anonymous reviewers for their constructive comments, which have greatly strengthened this article.

References

Bart, A. C., Gusukuma, L., & Kafura, D. (2021). We are authoring semi-automated feedback for Python Code with Pedal. *SIGCSE '21: Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. 1378. New York, USA, Online Conference. https://doi.org/10.1145/3408877.3439535

Bolukbasi, T., Chang, K.-W., Zou, J. Y., Saligrama, V., & Kalai, A. T. (2016). Man is to computer programmer as woman is to homemaker? Debiasing word embeddings. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, & R. Garnett (Eds.), *Advances in neural information processing systems 29 (NIPS 2016)*. Curran Associates, Inc. https://hdl.handle.net/2144/37516

Brock, K. (2019). *Rhetorical code studies: Discovering arguments in and around code*. University of Michigan Press. https://doi.org/10.3998/mpub.10019291

Brooke, S. J. (2021). Trouble in programmer's paradise: Gender biases in sharing and recognising technical knowledge on Stack Overflow. *Information Communication and Society*, *24*(14), 2091–2112. https://doi.org/10.1080/1369118X.2021.1962943/FORMAT/EPUB

Bucholtz, M., & Hall, K. (2005). Identity and interaction: A sociocultural linguistic approach. *Discourse Studies*, *7*(5), 585–614. https://doi.org/10.1177/1461445605054407

Butler, J. (1999). *Gender trouble: Feminism and the subversion of identity*. Routledge.

Carter, J., & Jenkins, T. (2002). Spot the difference: Are there gender differences in coding style? In *Proceedings of the 3rd Annual LTSN-ICS Conference*. https://kar.kent.ac.uk/id/eprint/13757

Criado-Perez, C. (2019). *Invisible women: Exposing data bias in a world designed for men*. Random House.

Davidson, R., & MacKinnon, J. G. (2000). Bootstrap tests: How many bootstraps? *Econometric Reviews*, *19*(1), 55–68. https://doi.org/10.1080/07474930008800459

Dohmke, T. (2023). *The GitHub Blog: 100 million developers and counting*. GitHub. https://github.blog/2023-01-25-100-million-developers-and-counting/

Doughman, J., Khreich, W., El Gharib, M., Wiss, M., & Berjawi, Z. (2021). Gender bias in text: Origin, taxonomy, and implications. *Proceedings of the 3rd Workshop on Gender Bias in Natural Language Processing* (pp. 34–44). Association for Computational Linguistics, Online Conference. https://doi.org/10.18653/v1/2021.gebnlp-1.5

Farah, J. C., Spaenlehauer, B., Sharma, V., Rodriguez-Triana, M. J., Ingram, S., & Gillet, D. (2022). Impersonating chatbots in a code review exercise to teach software engineering best practices. *2022 IEEE Global Engineering Education Conference (EDUCON)* (pp. 1634–1642). IEEE. https://doi.org/10.1109/EDUCON52537.2022.9766793

Flanagin, A. J. (2020). The conduct and consequence of research on digital communication. *Journal of Computer-Mediated Communication*, *25*(1), 23–31. https://doi.org/10.1093/jcmc/zmz019

Frieze, C., & Quesenberry, J. L. (2019). How computer science at CMU is attracting and retaining women. *Communications of the ACM*, *62*(2), 23–26. https://doi.org/10.1145/3300226

Holohan, A., & Garg, A. (2005). Collaboration online: The example of distributed computing. *Journal of Computer-Mediated Communication*, *10*(4). https://doi.org/10.1111/j.1083-6101.2005.tb00279.x

Johnson, S. C. (1978). *Lint, a C program checker*. Wolfram Schneider. https://wolfram.schneider.org/bsd/7thEdManVol2/lint/lint.pdf

Keyes, O. (2018, November). The misgendering machines: Trans/HCI implications of automatic gender recognition. In *Proceedings ACM Human-Computer. Interacion. 2, Community Supported Cooperative Work*, Article 88, 22 pages. New York, USA. https://doi.org/10.1145/3274357

Koenig, A. M. (2018). Comparing prescriptive and descriptive gender stereotypes about children, adults, and the elderly. *Frontiers in Psychology*, *9*(June), 1086. https://doi.org/10.3389/FPSYG.2018.01086

Lindvall-Östling, M., Deutschmann, M. & Steinvall, A. (2020). An exploratory study on linguistic gender stereotypes and their effects on

Downloaded from https://academic.oup.com/jcmc/article/29/1/zmad049/7596747 by Universiteit van Amsterdam user on 24 March 2026

perception. *Open Linguistics*, *6*(1), 567–583. https://doi.org/10.1515/opli-2020-0033

Liu, B., Wei, L., Wu, M., & Luo, T. (2023). Speech production under uncertainty: How do job applicants experience and communicate with an AI interviewer? *Journal of Computer-Mediated Communication*, *28*(4), 1–12. https://doi.org/10.1093/jcmc/zmad028

Mackenzie, A. (2005). The performativity of code. *Theory, Culture and Society*, *22*(1), 71–92. https://doi.org/10.1177/0263276405048436

Marino, M. C. (2020). *Critical code studies*. The MIT Press.

Nguyen, D., Doğruöz, A. S., Rosé, C. P., & de Jong, F. (2016). Computational sociolinguistics: A survey. *Computational Linguistics*, *42*(3), 537–593. https://doi.org/10.1162/COLI_a_00258

Obermüller, F., Bloch, L., Greifenstein, L., Heuer, U., & Fraser, G. (2021). Code perfumes: Reporting good code to encourage learners. *Proceedings of WiPSCE'21: Workshop on primary and secondary computing education*. Erlangen, Germany, Online Conference. https://doi.org/10.1145/3481312.3481346

Peters, T. (2004, 19th August). *The Zen of Python*. PEP 20. https://peps.python.org/pep-0020/#the-zen-of-python

Pylint. (2023). *Pylint 3.0.0a6: Documentation*. Logilab. https://pylint.readthedocs.io/en/latest/

Rother, K. (2017). *Pro Python best practices: Debugging, testing and maintenance*. Apress.

Terrell, J., Kofink, A., Middleton, J., Rainear, C., Murphy-Hill, E., Parnin, C., & Stallings, J. (2017). Gender differences and bias in open source: Pull request acceptance of women versus men. *PeerJ Computer Science*, *3*, e111. https://doi.org/10.7717/peerj-cs.111

Trinkenreich, B., Wiese, I., Sarma, A., Gerosa, M., & Steinmacher, I. (2021). Women's participation in open-source software: A survey of the literature. *ACM transactions on software engineering and methodology*, *31*(4), 1. https://doi.org/10.1145/3510460

van Rossum, G., Warsaw, B., & Coghlan, N. (2023). *PEP-8: The style guide for Python code*. PEP-8. https://pep8.org/

Vasilescu, B., Posnett, D., Ray, B., Van Den Brand, M. G. J., Serebrenik, A., Devanbu, P., & Filkov, V. (2015). Gender and tenure diversity in GitHub Teams. *Proceedings of CHI '15: Conference on Human Factors in Computing Systems, (Gender and Technology Track), Seoul Republic of Korea* (pp. 3789–3798). https://doi.org/10.1145/2702123.2702549

Vedres, B., & Vásárhelyi, O. (2019). Gendered behaviour as a disadvantage in open-source software development. *EPJ Data Science*, *8*(1), 25. https://doi.org/10.1140/epjds/s13688-019-0202-z

Vedres, B., & Vásárhelyi, O. (2022). Inclusion unlocks the creative potential of gender diversity in teams. *Scientific Reports*, Nature. *13*(1). https://doi.org/10.2139/ssrn.4085990

Viafore, P. (2021). *Robust Python*. O'Reilly.

Downloaded from https://academic.oup.com/jcmc/article/29/1/zmad049/7596747 by Universiteit van Amsterdam user on 24 March 2026