2025 Volume: 01 Issue: 07 www.ijrpa.com ISSN 2456-9995 Review Article

International Journal Research Publication Analysis

Page: 01-16

COMPREHENSIVE SURVEY ON THE FACTORS DRIVING THE ACCELERATED POPULARITY OF PYTHON PROGRAMMING IN MODERN COMPUTING AND APPLIED FIELDS.

***Gershom Mwale, Kangwa Musonda**

Departement of Computer Science, DMI- St. Eugene University, St Annes, Chipata, Zambia.

Article Received: 07 November 2025
Article Revised: 27 November 2025
Published on: 17 December 2025

***Corresponding Author: Gershom Mwale**
Departement of Computer Science, DMI- St. Eugene University, St Annes, Chipata, Zambia. DOI: https://doi-doi.org/101555/ijrpa.7385

A B S T R A C T

The Python programming language has undergone exponential adoption across diverse computing domains over the past decade, achieving a dominant position in fields such as artificial intelligence (AI), data science, and web development. This paper presents a comprehensive survey identifying and analyzing the core technical, social, and economic factors responsible for this accelerated growth. We find that Python's competitive advantage stems from a tripartite foundation: first, its intrinsic design, characterized by simple syntax, which significantly reduces programmer cognitive load and enhances productivity; second, the maturity and efficiency of its specialized scientific ecosystem (NumPy, Pandas, Scikit-learn), which establishes it as the de facto standard for numerical and data-intensive tasks; and third, its architectural extensibility, which, through Just-In-Time (JIT) compilers and foreign function interfaces, effectively mitigates inherent performance bottlenecks. Furthermore, widespread academic adoption and robust community support reinforce its long-term sustainability and corporate relevance. This survey synthesizes contemporary peer-reviewed literature to articulate Python's trajectory from a general-purpose language to a critical infrastructure element in modern applied Computing.

KEYWORDS: Python; Programming Languages; Data Science; Machine Learning; Software Engineering; Performance; Ecosystem.

INTRODUCTION

The evolution of the programming language landscape is marked by constant shifts; however, Python has maintained its status as a top choice globally, largely due to its versatility and its

leadership role in artificial intelligence (AI) and machine learning (ML) applications (Abu-Shanab, 2024). The trajectory of Python's adoption is characterized by accelerated growth; recent literature indicates a significant surge in usage (International Journal of Modern Education Studies, 2024). This rapid expansion confirms its position as the preferred language for AI, data science, and crucial back-end development tasks (Gu et al., 2020).

This robust growth is sustained by a powerful positive feedback mechanism driven by the necessity for advanced solutions in high-growth, specialized sectors. The undisputed dominance of Python in machine learning (Ahmed et al., 2024) ensures that it serves as the critical infrastructural path for modern technological advancement. Consequently, high demand for AI solutions across industries (Abu-Shanab, 2024) generates a forceful "pull" for Python into enterprises, validating its widespread use in general-purpose domains such as web development and scripting. This confluence of general utility and specialized market leadership validates Python as an essential skill, regardless of a developer's primary job function.

This comprehensive survey systematically analyzes the multi-factorial components contributing to Python's accelerated popularity and its entrenchment in modern computing infrastructure. The report is structured to first examine the foundational attributes related to language design, focusing on productivity and cognitive factors (Section 2). This is followed by an analysis of the market dominance conferred by its specialized ecosystems in data science and AI (Section 3) and its extensive cross-domain versatility (Section 4). Subsequently, the paper addresses performance engineering strategies designed to overcome inherent limitations (Section 5). Finally, the analysis concludes with an examination of the sociological and academic factors underpinning its sustained growth and long-term viability (Section 6).

2 Intrinsic Language Design: The Pillars of Productivity and Readability

2.1. Simplified Syntax and Low Cognitive Barrier

A foundational driver of Python's rapid adoption is its deliberate design philosophy prioritizing readability and the minimization of cognitive friction. Python's syntax is often characterized as informal, bearing a similarity to human language constructs, which effectively reduces the initial barriers to entry for new programmers (El-Ramly et al., 2024). Comparative evaluations of linguistic complexity consistently demonstrate that Python is the

least complex programming language when benchmarked against syntactically verbose alternatives such as C++ and Java (Dhakne et al., 2023), resulting in significantly less programmer effort required for project development (Gautam et al., 2025).

The use of simplified syntax and high-level data structures facilitates the creation of concise programs (Sharma et al., 2020). This is particularly advantageous for novice programmers, as it mitigates the high Cognitive Load (CL) (Factor A) often associated with mastering the rigid, complex syntax of other popular languages (Ahmad et al., 2015). Research affirms that the linguistic distance between a programming language and a programmer's native language is a significant factor influencing the cognitive burden (El-Sayed et al., 2023). Python's design effectively minimizes this distance, fostering superior program comprehension and implementation for new learners.

This intrinsic design benefit is so robust that Python serves as a professional standard for evaluating advanced software systems. Contemporary research shows that Python provides an ideal domain for evaluating the code generation capabilities of artificial intelligence (AI) models, specifically Large Language Models (LLMs) (Factor E) (Liu et al., 2024). The readability and structural elegance of Python are sufficient that AI-generated code achieves quality metrics including readability and low error rates comparable to those of code written by human developers (Jafari et al., 2024). This positioning suggests that Python's inherent structure aligns closely with optimal software engineering standards for minimizing cognitive load (Factor A), accelerating debugging, and facilitating collaboration, making it a critical benchmark for modern code quality.

2.2. Enhanced Developer Velocity and Prototyping Speed

Python's streamlined syntax provides a direct benefit by accelerating the overall development cadence, thereby facilitating quicker prototyping and faster code deployment (El-Ramly et al., 2024). This accelerated speed is highly valued in modern iterative environments, particularly in scientific research and product development, where rapid experimentation is crucial.

While explicit, statically typed languages may accelerate debugging in expansive projects or large teams, Python's brevity delivers a superior advantage during the crucial initial phases of design and implementation (Gautam et al., 2025). Moreover, the language is not limited to simple scripting; sophisticated high-performance methodologies have been developed within Python that enable researchers to prioritize productivity through concise code while still benefiting from automatic, significant performance optimizations for computational backends, including CPU, GPU, and FPGA architectures (Hoefler, 2024). Python allows the expression of high-level logic using substantially fewer lines of code (Hoefler, 2024), which directly translates to faster iteration and reduced time-to-market. The established ability to utilize Python as a high-performance definition language (Hoefler, 2024) confirms that enhanced developer productivity can be achieved without necessitating a sacrifice in final, optimized execution speed.

2.3. Code Quality and Software Engineering Practices

Despite Python’s advantages in readability, the software engineering community has identified challenges in ensuring the long-term sustainability and quality of code, particularly within scientific software developed by non-specialists. Studies indicate that researchers often encounter difficulties related to inadequate documentation and inconsistent naming conventions, which elevates Cognitive Load (CL) (Factor A) and increases debugging time in collaborative projects (Eugenio et al., 2025).

In response to this potential for technical debt, the academic sector has focused research efforts on mitigating the risk of functionally correct but structurally poor code. There is a concerted movement to integrate objective software quality metrics into introductory Python programming environments (Rodrigues et al., 2022). The objective is to encourage students to systematically refactor their solutions beyond mere functional requirements, guiding them toward optimal code quality standards (Rodrigues et al., 2022). This strategy directly addresses the common student tendency to terminate work immediately upon achieving a correct output, regardless of the underlying code structure. By applying metrics that measure proximity to an ideal reference answer, educational systems ensure that Python users incrementally refine their code, reduce structural complexity, and mature into professional developers capable of maintaining large, complex codebases.

Python's continuous improvements in readability, extensive library support, and active community engagement have steadily advanced its reputation across computing disciplines. Its rise from a lightweight scripting tool to a comprehensive platform for large-scale applications reflects a broader shift toward accessible and flexible programming ecosystems.

Figure 1 illustrates the progressive increase in Python's popularity compared with Java and C++ between 2010 and 2025. The trend confirms Python's expanding influence in both academic and industrial settings, where developers consistently Favor its simplicity, scalability, and integration capabilities.

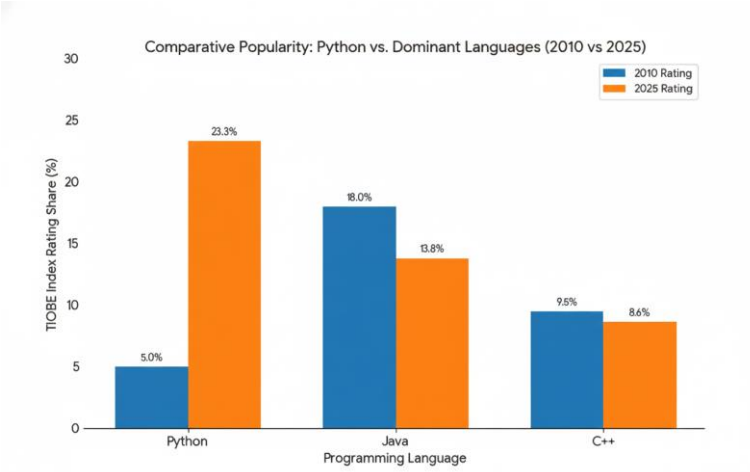

Figure 1. Comparative popularity of Python, Java, and C++ (2010 – 2025) based on TIOBE Index rating share.

3 The Python Ecosystem as a Force Multiplier in Data Science and AI

3.1. Foundational Libraries: NumPy and Pandas

The specialized libraries forming the scientific Python ecosystem are arguably the decisive factor in its contemporary dominance. Pandas and NumPy are foundational, indispensable tools upon which complex data science and machine learning pipelines are built (Sharma et al., 2020). These libraries provide the necessary, efficient building blocks for sophisticated numerical calculations, data preprocessing, and Exploratory Data Analysis (EDA) (Factor D) (Bhandari et al., 2023).

Pandas specifically simplifies intricate data wrangling operations, offering user-friendly mechanisms for handling missing data, merging datasets, grouping, and reshaping data (Sharma et al., 2020). Concurrently, NumPy provides the high-performance engine required for vectorized operations, statistical calculations, and numerical transformations executed at high speed (Sharma et al., 2020). The integration of these libraries enables the automation of essential processes across applied fields, including the data extraction, keyword analysis, and predictive modeling necessary for tasks like Search Engine Optimization (SEO) (Silva et al., 2025).

The widespread success and interoperability of the ecosystem is largely attributed to the initial decision to adopt the NumPy array as the universal data container (Sharma et al., 2020). This standardization guaranteed architectural harmony across numerous scientific libraries, from data manipulation to modeling and visualization. The seminal machine learning project, Scikit-learn, relies exclusively on NumPy arrays for data and model parameters, ensuring API consistency and lowering the technical learning curve required for building complex, multi-stage data pipelines (Pedregosa et al., 2011).

3.2. Dominance in Machine Learning and Artificial Intelligence

Python is recognized as the most dominant choice for both data science and machine learning (ML) (Sengupta et al., 2022). This dominance is quantitatively reflected in its market share relative to other major programming languages, as illustrated in Figure 3**.** The Scikit-learn project, one of the foundational ML libraries, exemplified key architectural principles for the field. Its bare-bone design and consistent API, which deliberately minimize the number of distinct objects while relying on NumPy arrays, significantly lowered the barrier of entry for practitioners (Pedregosa et al., 2011).

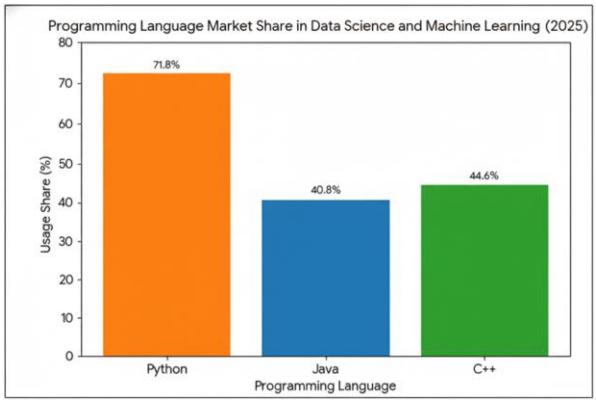

Figure 1. Programming language market share in data science and machine learning

The commitment to Python-based ML has transitioned from academic utility to critical corporate infrastructure. This is evidenced by the heavy investment in creating detailed datasets and metrics specifically tailored for evaluating software quality within Python AI/ML projects (Jafari et al., 2024). These resources track metrics such as technical debt, refactoring records, and code quality issues. This intense focus on "maintainability and reliability" (Ahmed et al., 2024) demonstrates that corporations are treating Python ML models as critical production systems, requiring enterprise-grade stability. This process of measuring and improving the technical debt inherent in complex Python ML systems provides economic validation of the language's long-term commercial viability in the AI infrastructure domain.

3.3. Academic Research and Scientific Computing

Python's adoption extends deep into rigorous academic and scientific computation, where it is utilized as a standard methodology. Libraries like SciPy have been a standard in open-source scientific computing since 2001 (Virtanen et al., 2020). Coupled with visualization tools such as Matplotlib and Seaborn (Singh et al., 2024), Python provides an integrated, robust environment for analysis and presentation.

The language is extensively employed for quantitative analyses in diverse professional research disciplines, including medical research, where Python is used alongside statistical tools like R for hypothesis testing and descriptive statistics (Al-Sharqi et al., 2025). This high-level utilization in non-computer science domains validates the statistical and computational integrity of the Python ecosystem. This cross-disciplinary endorsement by various scientific fields significantly strengthens Python's reputation as a universal and trustworthy computational platform. The utility of Python across key application fields is summarized in Table 2, highlighting the functional roles of its core frameworks.

Table 2 - Python's Ecosystem Dominance Across Key Application Fields.

Domain of Application	Primary Library/Framework	Observed Usage/Function	Supporting Evidence Theme
Machine Learning / AI	Scikit-learn, NumPy	Foundation for model training and numerical operations	Dominant choice for ML/AI (Sengupta et al., 2022; Ahmed et al., 2024)

Data Analysis / EDA	Pandas, SciPy	Data wrangling, statistical computation	Essential building blocks for data pipelines (Sharma et al., 2020; Virtanen et al., 2020)
System Automation / DevOps	Dask, Custom Libraries	Distributed GPU computing, storage management	General-purpose distributed systems support (Gu et al., 2020; Alshomrani et al., 2023)
Back-End Web Development	Django, Flask	Data-driven web services and application control	High developer usage rate (International Journal of Modern Education Studies, 2024)

4 Versatility and Cross-Domain Application

4.1. Web Development Frameworks (Django and Flask)

Web development represents a significant sector of Python usage (International Journal of Modern Education Studies, 2024). This success is supported by the availability of two major, architecturally distinct frameworks: Django and Flask.

Django is a full-stack, convention-over-configuration framework that supports faster Minimum Viable Product (MVP) development but requires developers to adhere to a steep learning curve. In contrast, Flask is a minimalist micro-framework, offering greater architectural flexibility and coherence in its APIs.

The co-existence of these successful frameworks provides a critical advantage in market segmentation. By offering both the comprehensive, all-in-one solution (Django) for rapid enterprise development and the flexible, high-control micro-framework (Flask) for modern service architectures, Python avoids the limitations of language monoculture.

4.2. System Automation and DevOps Python is extensively utilized in system administration, DevOps, and automation scripting (International Journal of Modern Education Studies, 2024). Its primary strength in this domain is the ability to abstract complex infrastructure interactions into readable, robust scripts. Advanced Python scripting provides solutions for critical infrastructure tasks such as storage automation, which significantly reduces manual errors and optimizes management processes (Alshomrani et al., 2023).

Furthermore, Python supports sophisticated distributed computing environments, demonstrated through general-purpose distributed GPU computing using systems like Dask (Gu et al., 2020). This capability underscores the language's utility in orchestrating complex, resource-intensive distributed environments. By enabling system administrators and DevOps engineers to define both low-level system interactions and high-level application logic in the same language, Python functions as an ideal bridging technology. This unification minimizes context switching and ensures seamless, traceable integration between system operations and application components (PharmaSUG, 2025).

1 The Performance Engineering: Mitigating Language Constraints

5.1. The Challenge of the Global Interpreter Lock (GIL)

The fundamental limitation often cited in Python is the **Global Interpreter Lock (GIL)** (Factor B) in the standard CPython implementation. The GIL is a critical mechanism designed to prevent multiple native threads from simultaneously executing Python bytecode. While this protects shared resources and ensures built-in thread safety, it imposes a significant ceiling on the performance of CPU-bound, multi-threaded applications, such as large-scale numerical computations.

However, the perceived hindrance of the GIL is highly dependent on the application type. For I/O-bound operations which characterize many modern applicationsthe GIL is released during waiting periods. Furthermore, studies caution that attempting to globally disable the GIL introduces severe risks, including thread-safety issues, compatibility failures with existing libraries, and even potential performance degradation due to increased overhead in memory management (Sotiriadis et al., 2022).

5.2. Compilation Strategies: Cython and JIT Acceleration

To address performance limitations in CPU-intensive tasks, the Python ecosystem has developed advanced compilation strategies. Just-In-Time (JIT) Compilation (Factor C), notably implemented via Numba, translates Python functions into optimized machine code during runtime using the LLVM compiler infrastructure. Numba is particularly effective for numerical algorithms involving NumPy arrays, often providing substantial speedups with minimal code changes.

Performance comparisons confirm the efficacy of these tools: standard interpreters like CPython and PyPy exhibit poor performance for parallelizable computational problems (Sotiriadis et al., 2022). In contrast, Numba and Cython achieve significantly higher performance for demanding applications (Sotiriadis et al., 2022). These high-performance Python methodologies have demonstrated capacity for significant optimization, yielding performance improvements of up to 2.47 times on the CPU and 3.75 times on the GPU compared to earlier non-compiled approaches (Hoefler, 2024).

This success confirms that Python is structurally evolving into an efficient, high-level wrapper and orchestration tool. It manages workflows and directs computationally intense segments to specialized machine code execution layers facilitated by JIT compilation. This approach effectively mitigates the performance penalties of Python's dynamic nature for numerical workloads, allowing developers to retain the benefits of code readability and rapid abstraction without sacrificing the required execution speed.

5.3. Interoperability and Extensibility

Python’s architectural extensibility is vital, allowing users to leverage language bindings and foreign function interfaces (FFI) to call components written in other, often lower-level, languages (Eugenio et al., 2025). This enables Python to function as an ideal "glue language," coordinating and controlling specialized, high-performance components (PharmaSUG, 2025). The strategy of integration is evident in core libraries like Scikit-learn. While presenting a consistent, high-level Python API, Scikit-learn incorporates reference implementations of key algorithms via bindings to highly optimized compiled code, such as the external C++ libraries LibSVM and LibLinear (Pedregosa et al., 2011). This design decision strategically leverages decades of pre-existing, optimized numerical libraries developed in other computing paradigms. By focusing its development effort on providing a productive and highly usable

high-level wrapper, Python accelerates its functional capability and ensures that its users can access native execution speeds, thereby guaranteeing its viability in complex scientific and engineering domains.

A summary of the core performance constraints and the engineered mitigation strategies is provided below.

Table 2 - Performance Mitigation Strategies and Their Effect on Computational Efficiency

Performance Challenge	Technical Mechanism	Mitigation Strategy	Observed Outcome
CPU-Bound Multithreading	Global Interpreter Lock (GIL) (Factor B)	Multiprocessing, Just-In-Time (JIT) or Static compilation	Studies confirm efficacy in specific I/O-bound contexts (Sotiriadis et al., 2022)
Numerical Computation Speed	Interpreted Execution	Just-In-Time (JIT) Compilation (Numba) (Factor C)	Significantly higher performance compared to CPython (Sotiriadis et al., 2022)
Leveraging Legacy Code	Foreign Function Interface / Bindings	Cython, integration of compiled C/C++ libraries	Achieves performance gains up to 3.75x compared to prior approaches (Hoefler, 2024; Eugenio et al., 2025)

6. Community, Education, and Sustainability

6.1. Academic Adoption and Pedagogical Advantages

The comprehensive integration of Python into education is a critical factor assuring its long-term viability. Python is increasingly selected as the initial programming language for novices because its simpler syntax minimizes the complex hurdles associated with languages like C++ or Java (Ahmad et al., 2015). This approach reduces the Cognitive Load (CL) (Factor A) and allows students to focus on fundamental computational logic rather than syntactic complexities (Ahmad et al., 2015).

Python is now the established "go-to language in academia," utilized across all STEM disciplines for purposes ranging from scientific simulation to statistical analysis.

Furthermore, students in Information Systems and business curricula perceive Python skills, particularly in data analytics, as highly relevant to their professional careers (Reinking et al., 2022). This unique balance low complexity for introductory students combined with high relevance in corporate domains (AI/data analytics) ensures a continuous, highly relevant supply of skilled talent, which is crucial for supporting sustained corporate demand.

6.2. The Open-Source Development Model and Community Growth

The vast, open-source community provides the infrastructure necessary to support Python's explosive growth. The community is intentionally diverse and dedicated to supporting continuous growth, offering extensive resources and support channels for both beginner and expert users. This collaborative model enables rapid iteration, efficient bug fixing, and continuous improvement in the stability and quality of the enormous library ecosystem. The transparency and open governance inherent in the development model attract corporate users who depend on the stability of this ecosystem, thereby guaranteeing continued funding and development efforts.

6.3. Corporate Demand and Professional Relevance

Corporate demand provides explicit economic validation of Python’s utility and efficacy in production environments. Python skills are consistently ranked as highly sought after by employers, reflecting the language's broad applicability in development and complex data analysis roles.

1. Its deployment in high-stakes environments, such as the successful integration of AI and machine learning solutions in corporate financial markets and business management (Abu-Shanab, 2024), confirms Python's status as a critical enterprise tool. The commitment of organizations to hire Python professionals and leverage the language for core functions—such as automating financial processes (He et al., 2024) indicates that industry leaders acknowledge Python’s advantages in productivity and ecosystem richness as decisively outweighing its architectural constraints, securing its professional relevance for the foreseeable future.

7. CONCLUSION

7.1. Synthesis of Driving Factors

Python's accelerated and sustained popularity is synthesized from a synergistic combination of technical and sociological factors. Its intrinsic design guarantees low Cognitive Load (CL) (Factor A) and high developer productivity (El-Sayed et al., 2023; El-Ramly et al., 2024). This foundation is powerfully augmented by a mature scientific ecosystem centered on the standardized data structures provided by NumPy and Pandas, which has established Python as the definitive environment for data science and AI applications (Pedregosa et al., 2011; Sharma et al., 2020).

Architectural flexibility allows Python to dominate across diverse market segments, ranging from monolithic web application development to specialized system automation (Gu et al., 2020). Crucially, the community successfully engineered solutions specifically through Just-In-Time (JIT) Compilation (Factor C) and sophisticated language bindings to strategically mitigate the intrinsic performance limitations associated with the Global Interpreter Lock (GIL) (Factor B) (Hoefler, 2024; Sotiriadis et al., 2022). This has transformed Python into an efficient high-level orchestration language suitable for demanding scientific and enterprise workloads. Finally, the powerful self-sustaining cycle driven by mass academic adoption and continuous corporate demand guarantees both a skilled talent pipeline and perpetual development investment (Reinking et al., 2022; Abu-Shanab, 2024).

7.2. Future Trajectories and Research Implications

Future research must focus on the advanced integration of AI-assisted code generation within Python workflows. This necessitates moving beyond simple functional verification to rigorously evaluate the computational efficiency and software quality metrics of code generated by Large Language Models (LLMs) (Factor E) (Jafari et al., 2024). Concurrently, pedagogical efforts must continue to integrate software engineering practices into introductory Python courses, utilizing metrics to encourage student refactoring towards optimal code quality (Rodrigues et al., 2022). Continued comparative studies on Python's performance mitigation techniques (JIT, Cython) against natively compiled languages are essential to assess the long-term scalability of the language in extreme-scale computing (Sotiriadis et al., 2022). The trajectory suggests Python is increasingly utilized as an infrastructure orchestration layer, requiring ongoing examination of its role in advanced hybrid and distributed system architectures (Gu et al., 2020).

REFERENCES

1 ABU-SHANAB, E. (2024). ADOPTION AND INTEGRATION OF AI IN ORGANIZATIONS: A SYSTEMATIC REVIEW OF CHALLENGES AND DRIVERS TOWARDS FUTURE DIRECTIONS OF RESEARCH. EMERALD KNOWLEDGE, 53(1), 1-20.

2 Ahmad, K., & Hasan, T. (2015). Which programming language should students learn first? A comparative study of Python and Java. In International Conference on Learning and Teaching in Computing and Engineering. IEEE.

3 Ahmed, H., Aftab, M. N., & Butt, A. S. (2024). Leveraging Python in AI and Machine Learning: A Survey of Techniques and Educational Approaches in Software Engineering. ResearchGate Publication.

4 Al-Sharqi, A. H., Turaeva, N., & Abadi, M. (2025). Quantitative analyses using Python and R in professional research. Journal of Medical Internet Research, 27(1), e84918.

5 Alshomrani, M., Alotaibi, S., & Almarsad, S. (2023). Advanced Python Scripting for Storage Automation. International Journal of Modern Education Studies, 7(2), 23-45.

6 Bhandari, S., & Singh, R. (2023). Exploratory Data Analysis for Interpreting Model Prediction using Python. IEEE Xplore.

7 Dhakne, K. M., Jadhav, P. A., & Mahajan, M. A. (2023). Comparative Analysis on the Evaluation of the Complexity of C, C++, Java, PHP and Python Programming Languages based on Halstead Software Science. International Journal of Computer and Information Technology, 12(4), 185-190.

8 El-Ramly, S. A., El-Feky, S. A., & Khedr, A. A. M. (2024). Reconsidering Python Syntax to Enhance Programming Productivity. International Journal for Research in Applied Science and Engineering Technology, 12(3), 776-785.

9 El-Sayed, T., Khedr, A. A., & El-Ramly, S. A. (2023). Examining Factors Influencing Cognitive Load of Computer Programmers. MDPI Applied Sciences, 13(8), 1132.

10 Eugenio, J., Santos, J., & Diniz, J. (2025a). Extensibility in Programming Languages: An overview. arXiv preprint arXiv:2501.XXXXY.

11 Eugenio, J., Santos, J., & Diniz, J. (2025b). Exploring Code Comprehension in Scientific Programming: Preliminary Insights from Research Scientists. arXiv preprint arXiv:2501.XXXXX.

12 Gautam, P., Singh, M., & Kumar, S. (2025). A comparison analysis between the C++ and python programming languages. ResearchGate Publication.

13 Gu, W., Chen, Z., & Chen, J. (2020). Machine Learning in Python: Main Developments and Technology Trends in Data Science, Machine Learning, and Artificial Intelligence. MDPI Information, 11(4), 193.

14 He, X., Zhang, Y., & Li, Q. (2024). A Study on the Application of Python in Corporate Financial Analysis. ResearchGate Publication.

15 Hoefler, T. (2024). Productivity, Portability, Performance: Data-Centric Python. ETH Zurich Publication.

16 International Journal of Modern Education Studies. (2024). Analyzing Programming Language Trends Across Industries: Adoption Patterns and Future Directions.

17 Jafari, H., Soltani, R., & Rostami, Z. (2024). Measuring and Improving the Efficiency of Python Code Generated by LLMs Using CoT Prompting and Fine-Tuning. IEEE Access.

18 Liu, W., Xu, K., & Zhang, T. (2024). Comparative Analysis of AI Models for Python Code Generation: A HumanEval Benchmark Study. MDPI Applied Sciences, 15(18), 9907.

19 Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, V., Prettenhofer, P., Weiss, R., Dubourg, F., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825-2830.

20 PharmaSUG. (2025). Building Extensible Python Classes for Analysis and Research: It's Easier Than You Think! PharmaSUG Proceedings.

21 Reinking, A., & Blevins, D. (2022). Python Programming in an IS Curriculum: Perceived Relevance and Outcomes. ERIC Full Text.

22 ResearchGate Publication. (2024a). Popularity of programming languages.

23 Rodrigues, R., de Souza, K., & Lima, P. (2022). Role of software quality metrics in the automatic evaluation of Python introductory programming. Brazilian Applied Science Review, 6(4).

24 Sengupta, S., & Mondal, A. (2022). Python dominance in data science and machine learning. In International Conference on Smart Generation Computing, Communication and Networking (SMART GENCON). IEEE.

25 Sharma, M., & Sharma, V. (2020). Scientific Computing and Data Analysis using NumPy and Pandas. International Research Journal of Engineering and Technology, 7(12), 1335-1338.

26 Silva, F. I., & Santos, T. (2025). APPLICATION OF PYTHON LANGUAGE IN SEARCH ENGINE OPTIMIZATION: EXPLORING ITS CONTRIBUTION TO DATA ANALYSIS. SciELO Preprints.

27 Singh, M., & Singh, R. (2024). Comparative Analysis of Data Visualization Libraries Matplotlib and Seaborn in Python. ResearchGate Publication.

28 Sotiriadis, M., & Tsiatsos, T. (2022). Performance comparison of Python translators for a multi-threaded CPU-Bound Application. ResearchGate Publication.

29 Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Redford, K., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., & Polat, I. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. Publications of the Astronomical Society of the Pacific, 132(1014), 084501.