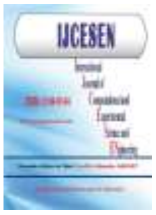
Copyright ©IJCESEN

International Journal of Computational and Experimental Science and ENgineering (IJCESEN)

Vol. 10-No.3 (2024) pp. 461-469
http://www.ijcesen.com



ISSN: 2149-9144

Research Article

Comparative Analysis of Programming Languages Utilized in Artificial Intelligence Applications: Features, Performance, and Suitability

Güzin TÜRKMEN[1]*, Arda SEZEN[2], Gökhan ŞENGÜL[3]

[1]Atılım University, Engineering Faculty, Computer Engineering Department, 06830, Ankara-Turkiye
* **Corresponding Author Email:** guzin.turkmen@atilim.edu.tr - **ORCID:** 0000-0003-0884-4876

[2]Atılım University, Engineering Faculty, Computer Engineering Department, 06830, Ankara-Turkiye
Email: arda.sezen@atilim.edu.tr - **ORCID:** 0000-0002-7615-3623

[3]Atılım University, Engineering Faculty, Computer Engineering Department, 06830, Ankara-Turkiye
Email: gokhan.sengul@atilim.edu.tr - **ORCID:** 0000-0003-2273-4411

Article Info:

DOI: 10.22399/ijcesen.342
Received : 11 June 2024
Accepted : 03 September 2024

Keywords

AI
Programming Languages
Machine Learning Applications

Abstract:

This study presents a detailed comparative analysis of the foremost programming languages employed in Artificial Intelligence (AI) applications: Python, R, Java, and Julia. These languages are analysed for their performance, features, ease of use, scalability, library support, and their applicability to various AI tasks such as machine learning, data analysis, and scientific computing. Each language is evaluated based on syntax and readability, execution speed, library ecosystem, and integration with external tools. The analysis incorporates a use case of code writing for a linear regression task. The aim of this research is to guide AI practitioners, researchers, and developers in choosing the most appropriate programming language for their specific needs, optimizing both the development process and the performance of AI applications. The findings also highlight the ongoing evolution and community support for these languages, influencing long-term sustainability and adaptability in the rapidly advancing field of AI. This comparative assessment contributes to a deeper understanding of how programming languages can enhance or constrain the development and implementation of AI technologies.

1. Introduction

The importance of selecting the most appropriate programming languages for artificial intelligence (AI) applications has become increasingly evident. As AI technologies advance rapidly, choosing the correct programming language is crucial for the successful development and implementation of AI code-based solutions. Among the various coding languages used in AI projects such as Python, R, Java, and Julia each possesses unique characteristics and capabilities in terms of AI application development. Understanding these languages and assessing their suitability for different AI tasks is essential for developers and researchers seeking to refine their development approaches. This study seeks to provide insights into selecting the most appropriate programming language, considering factors such as ease of use, performance, scalability, and library support through a case study. In this study, we conduct a thorough comparative evaluation of the selected programming languages utilized in AI applications. By examining features, performance metrics, and real-world suitability, this research intends to assist AI practitioners, researchers, and developers in making informed decisions regarding language selection for their AI projects.

1.1. Overview of Programming Languages in AI

In the domain of AI, different programming languages have significant influence over the outcomes of various projects. Python stands out for its simplicity and flexibility, becoming a preferred choice with its extensive frameworks and user-friendly features [1]. Python's popularity in AI development comes from its readability, ease of learning, and vast ecosystem of libraries tailored for machine learning, deep learning, and other AI tasks [2, 3]. R mostly comes forward for tasks involving data analysis and visualization due to its specialization in statistical computing [4]. It offers powerful statistical functions and visualization tools, making it particularly suitable for tasks such as exploratory data analysis, statistical modelling, and data visualization in AI projects [5]. For large-scale AI projects, Java is highly mentioned in the literature for its cross-platform operability and focus on object-oriented programming capabilities [6, 7]. While not as commonly used for AI as Python or R, Java's robustness, scalability, and mature ecosystem make it a suitable choice for developing enterprise-level AI applications, especially in scenarios requiring integration with existing systems or high-performance computing [8, 9]. In the realm of scientific computing requiring high-efficiency computation, Julia emerges as a viable candidate for AI innovation [10]. Julia is designed for high-performance numerical and scientific computing, offering speed comparable to low-level languages like C and Fortran while maintaining high-level syntax similar to Python [11, 12]. Its ability to combine high-performance computing with ease of use makes it attractive for AI tasks which are demanding computational efficiency, such as numerical simulations, optimization, and scientific modelling [13-15]. Each programming language brings distinct advantages and challenges to AI development, requiring careful consideration aligned with project requirements to determine the optimal choice.

2. Methodology

Our methodology for comparative analysis in this study involves a systematic approach to evaluate and compare widely used programming languages commonly employed in AI development, including Python, R, Java, and Julia. Firstly, we selected these languages based on their popularity, relevance, and prevalence in the AI development landscape, aiming to provide a comprehensive comparison that reflects the broader preferences and practices within the AI community [1]. Secondly, we identified key evaluation metrics encompassing aspects such as syntax and readability, performance benchmarks, library support, and overall suitability for AI tasks. These metrics were chosen to capture the multifaceted nature of programming languages' suitability for AI tasks and to ensure a comprehensive evaluation. Thirdly, we performed a linear regression task focused on implementing a model to fit a straight line to a set of data points, minimizing the distance between these points and the line. The dataset employed for this use case is designed to provide a comprehensive performance evaluation across different programming languages and systems. It comprises 1.000.000 observations and 100 independent variables, making it sufficiently large to effectively differentiate execution times. Feature values varies from 0 to 1000, with a noise level of 10% introduced to mirror real-world conditions. Furthermore, the dataset integrates both linear and non-linear patterns through the inclusion of sine and logarithmic functions, thereby enhancing its complexity and realism. The dataset's storage space is approximately 1,85 GB when saved as a CSV file, underscoring its substantial size and intricacy.The creation of this dataset involved several critical steps to ensure its robustness and relevance for benchmarking. Initially, 100 features were generated, followed by the derivation of random coefficients corresponding to these features. The dependent variable was then calculated using these coefficients, with added noise to emulate real-world variability. To ensure consistency in comparison, features were scaled during the generation process. The dataset was generated using Python. This methodological approach ensures that the dataset is not only fit for purpose in benchmarking linear regression models but also can contribute to practical data analysis scenarios. For the development of this use case with different programming languages the VS Code v1.89.0 IDE, is preferred along with the listed extension packs: Java v21, Python v3.11.4, R v4.4.1 and Julia v1.10.3. Use cases are developed with the mentioned programming languages using an Apple silicon machine with 64 GB of memory.

3. Comparative Analysis of Python, R, Java, and Julia

3.1. Comparison of Python, R, Java, and Julia across different aspects relevant to AI development

3.1.1. Syntax and Readability

Python is renowned for its clear and concise syntax, which makes it highly readable and suitable for developers of all levels [16, 17]. Its structure resembles pseudo-code which is a high-level description of an algorithm that uses conventions for human understanding. Also, it enhances comprehension and ease of coding by its straightforward syntax that is close to natural language and its build-in functions that perform common operations and simple control structures. R, on the other hand, emphasizes readability, particularly in statistical analysis, with a focus on data exploration and visualization. While its syntax may be less intuitive for beginners compared to Python, it offers specialized features for statistical computing. Java's syntax tends to be more verbose, which can make it less readable, especially for beginners. However, it has strict syntax rules that contributes to code clarity and maintainability in large-scale projects. Julia offers a clean and expressive syntax like Python, enhancing readability and ease of use. Its minimalist design and consistent syntax contribute to code clarity, making it suitable for both beginners and experienced developers. The code sections measuring the start time of the linear regression model, which is tried as a use case for different languages, are given in Figures 1, 2, 3, 4, respectively.

3.1.2. Availability of AI Libraries

Python supports a vast ecosystem of AI libraries and frameworks, making it a popular choice for AI development. Libraries such as; TensorFlow, PyTorch, and scikit-learn provide comprehensive support for machine learning, deep learning, and data analysis tasks. R offers a wide range of packages tailored for statistical analysis, machine learning, and data visualization. Popular packages include Caret, ggplot2, and dplyr, making R a strong contender for statistical computing tasks. While Java may not have an extensive collection of AI-specific libraries as Python or R, it offers robust support for general-purpose programming and integration with AI frameworks like TensorFlow and Weka. Julia's ecosystem is growing rapidly, with an increasing number of AI libraries and packages being developed. Libraries like Flux.jl, MLJ.jl, and DataFrames.jl provide support for machine learning, deep learning, and data manipulation tasks.

```
# Measure start time
start_time = time.time()
start_local_time = time.localtime(start_time)
print(f"Start Time: {time.strftime('%Y-%m-%d
%H:%M:%S', start_local_time)}")
```

Figure 1. *Python code block for measuring start time*

```
// Measure start time
long startTime = System.nanoTime();
LocalDateTime startLocalTime = LocalDateTime.now();
DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("yyyy-MM-dd
HH:mm:ss");
System.out.println("Start Time: " +
startLocalTime.format(formatter));
```

Figure 2. *Java code block for measuring start time*

```
# Measure start time
start_time = time_ns()
start_local_time = now()
println("Start Time: ", start_local_time)
```

Figure 3. *Julia code block for measuring start time*

```
# Measure start time
start_time <- Sys.time()
start_local_time <- format(start_time, "%Y-%m-%d
%H:%M:%S")
cat("Start Time:", start_local_time, "\n")
```

Figure 4. *R code block for measuring start time*

3.1.3. Support for Data Manipulation and Visualization

Python excels at data manipulation and visualization tasks benefited from the libraries like Pandas, NumPy, and Matplotlib. These libraries offer powerful tools for handling and visualizing data efficiently. R is highly regarded for its data manipulation and visualization capabilities, with packages like dplyr, tidyr, and ggplot2 providing powerful tools for data manipulation and visualization tasks. On the other hand Java offers support for general-purpose programming and integration with visualization libraries like JFreeChart and JavaFX. Julia offers strong support for data manipulation and visualization, with packages like DataFrames.jl, Query.jl, and Gadfly.jl providing efficient tools for data manipulation and visualization tasks.

3.1.4. Integration with External Tools and Platforms

Python offers seamless integration with external tools and platforms, based on its extensive ecosystem of libraries and frameworks. It

Table 1. *Comparison of Python, R, Java, and Julia across different aspects relevant to AI development*

Aspect	Python	R	Java	Julia
Syntax and Readability	Clear and concise syntax, highly readable, resembling pseudo-code.	Readable syntax with a focus on statistical analysis.	Verbose syntax, strict rules contribute to code clarity.	Clean, expressive syntax, suitable for beginners and experienced developers.
Availability of AI Libraries	Vast ecosystem of AI libraries and frameworks like TensorFlow, PyTorch, scikit-learn.	Wide range of packages (Caret, ggplot2, and dplyr) tailored for statistical analysis, machine learning, and data visualization.	Robust support for general-purpose programming and integration with AI frameworks (TensorFlow and Weka).	Growing ecosystem with libraries like Flux.jl, MLJ.jl, DataFrames.jl.
Support for Data Manipulation and Visualization	Extensive libraries like Pandas, NumPy, Matplotlib for efficient data manipulation, large matrix operations and visualization.	Powerful tools like dplyr, tidyr, ggplot2 for data manipulation and visualization tasks.	Limited specialized libraries; support for general-purpose programming (JFreeChart and JavaFX).	Strong support with packages like DataFrames.jl, Query.jl, Gadfly.jl.
Integration with External Tools and Platforms	Seamless integration with databases, cloud services, and other software components.	Integrates well with tools and platforms in statistical computing and data analysis.	Strong integration capabilities for enterprise-level applications.	Growing integration capabilities with support for interfacing with external tools and platforms.

integrates well with databases, cloud services, and other software components, making it suitable for a wide range of applications and environments. R integrates well with external tools and platforms, particularly in the field of statistical computing and data analysis. It offers interfaces to databases, web services, and APIs, allowing for efficient data integration and analysis workflows. Java's strong integration capabilities make it suitable for enterprise-level applications and integration with external tools and platforms. It offers robust support for database connectivity, web services, and enterprise frameworks like Spring. Julia's integration capabilities are growing rapidly, with support for interfacing with external tools and platforms through libraries and packages. It offers integration with databases, web services, and other software components, enabling efficient data exchange and interoperability in AI applications. Table 1 represents multiple aspect comparison of these programming languages.

3.2. Performance Evaluation

3.2.1. Execution Speed and Efficiency

Python is a simple programming language and it has an interpreted nature in terms of code execution. Optimizations like NumPy enhance its execution speed. R, also interpreted, offers specialized optimizations for statistical tasks. Java, a compiled language, provides faster execution speeds, especially for CPU-intensive tasks. Julia, designed for high-performance computing, often outperforms Python and R.

3.2.2. Memory Usage and Resource Consumption

Python and R may exhibit higher memory usage due to dynamic typing and garbage collection. Java's memory management system is efficient, leading to lower memory overhead. Julia's memory management is optimized for high-performance computing, enhancing memory efficiency [18].

3.2.3. Scalability for Large Datasets and Complex Algorithms

Python and R may face limitations in scalability due to their interpreted nature, but frameworks like Dask and PySpark offer solutions. Java excels in scalability. Julia is highly scalable, with native support for parallel processing and efficient memory management.

3.2.4. Parallel Processing Capabilities

Python's parallel processing capabilities are hindered by the Global Interpreter Lock (GIL), but

Table 2. *Comparison of Python, R, Java, and Julia according to performance evaluation*

Performance Factor	Python	R	Java	Julia
Memory Usage and Resource Consumption	Higher memory usage due to dynamic typing and garbage collection.	Memory management can be a concern for large datasets and matrix computations.	Efficient memory management with lower overhead; robust garbage collection.	Optimized for memory efficiency and high-performance computing.
Scalability	Limited scalability due to GIL and interpreted nature; distributed computing frameworks are available.	Limited scalability for memory-intensive operations; parallel processing packages available.	Excellent scalability for large-scale applications; support for distributed computing frameworks.	Well-regarded scalability; native support for parallel processing and distributed computing.
Parallel Processing Capabilities	Limited by GIL	Improving parallel processing capabilities with related packages	Robust support for multi-threading and concurrency; parallel stream processing available.	Native support for parallelism with multi-threading, distributed computing, and GPU acceleration.

libraries like Dask and Ray offer solutions. R's parallel processing capabilities are improving, but it may face challenges due to interpretation. Java excels in parallel processing, supported by robust multi-threading features. Julia stands out with native support for parallelism, making it well-suited for parallel processing in AI applications.

In summary, each language has its strengths and limitations in performance evaluation for AI applications. Python and R offer ease of use. Java excels in execution speed and scalability. Julia stands out for high-performance computing. The selection of the most suitable language depends on the specific requirements and constraints of the AI project, considering factors such as memory usage, scalability, and parallel processing capabilities (see Table 2).

3.3. Comparison of Python, R, Java, and Julia According to Suitability Assessment for AI tasks

3.3.1 Compatibility with AI Tasks

Python is highly compatible with a wide range of AI tasks, including machine learning, natural language processing (NLP), computer vision, and robotics. Its extensive ecosystem of libraries and frameworks such as TensorFlow, PyTorch, NLTK, and OpenCV make it a popular choice for AI development across various domains. While R is primarily known for its strength in statistical computing and data analysis, it also has packages and libraries for machine learning and NLP tasks. However, its support for computer vision and robotics may be limited compared to Python. Java offers robust support for AI tasks, particularly in enterprise-level applications and large-scale systems. While it may not have as extensive a collection of specialized libraries as Python or R. Julia is gaining traction in the AI community, especially for high-performance computing tasks. It offers support for machine learning, NLP, and scientific computing, with growing libraries and packages for these tasks. However, its adoption in areas like computer vision and robotics may be relatively lower compared to Python and Java.

3.3.2. Flexibility for Prototyping, Experimentation, and Deployment

Python is known for its flexibility, making it ideal for rapid prototyping, experimentation, and deployment of AI models. Its simple syntax, extensive libraries, and frameworks facilitate quick iteration and development cycles. R offers flexibility for prototyping and experimentation. However, its deployment capabilities may be limited compared to Python, particularly in production environments. Java provides strong support for building robust and scalable AI applications, making it suitable for deployment in production environments. While it may have a steeper learning curve compared to Python and R, its performance and reliability are advantageous for enterprise-level deployments. Julia offers a balance between flexibility and performance, making it suitable for both prototyping and deployment of AI models. Its high-performance computing capabilities enable efficient execution of complex algorithms, while its syntax and ecosystem support rapid development and experimentation.

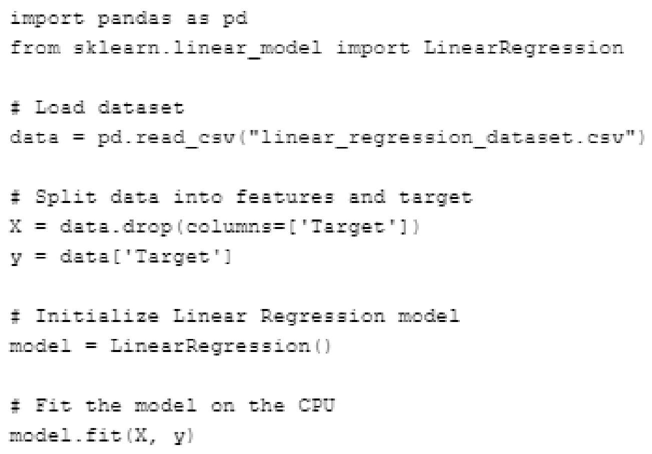
***Table 3**. Comparison of Python, R, Java, and Julia according to suitability assessment for AI tasks*

Aspect	Python	R	Java	Julia
Compatibility with AI Tasks	Widely compatible; extensive ecosystem for machine learning, NLP, computer vision, and robotics.	Strong in statistical computing and data analysis; limited support for computer vision and robotics.	Strong support for enterprise-level AI applications; may lack specialized libraries.	Growing support for high-performance computing, machine learning, and scientific computing.
Flexibility for Prototyping & Deployment	Highly flexible for rapid prototyping and deployment; extensive library support	Flexible for prototyping and experimentation; may have limitations in deployment compared to Python.	Strong support for scalable, reliable deployments; steeper learning curve for beginners.	Balances flexibility and performance; suitable for both prototyping and deployment.

Table 3 provides a concise overview of the strengths and weaknesses of each programming language in terms of suitability for AI tasks, helping to inform decisions based on specific project requirements and objectives.

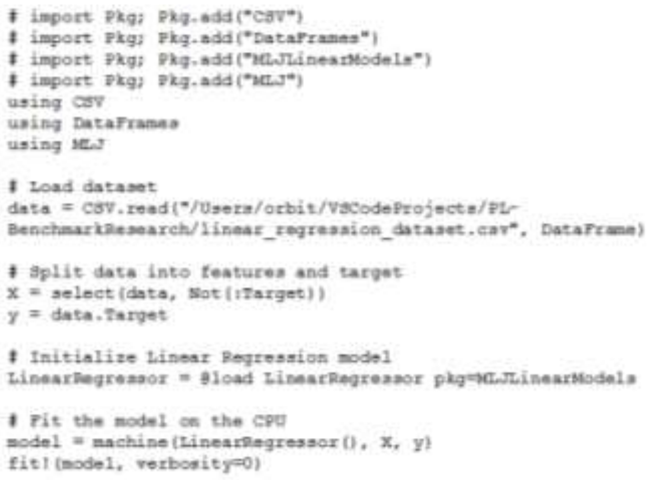
4. Case Study

Linear regression is a fundamental technique in artificial intelligence and machine learning used to predict outcomes based on the linear relationship between variables. It is widely utilized in various domains such as finance, healthcare, and marketing to model and analyse the relationships between multiple variables. This case study will focus on implementing a linear regression model to fit a straight line to a set of data points, minimizing the distance between these points and the line. By comparing implementations in Python, Julia, Java, and R, the syntactic differences, library support, and overall ease of use across these languages in the context of AI applications are evaluated. For this case study, Python, Julia, Java and R benchmark codes are partially given in the figure 5-8. The provided code snippets for implementing a linear regression model in Python, Julia, Java, and R are generally well-structured and demonstrate the basic usage of linear regression fitting function in each language. Model of these snippets demonstrates the basic approach to linear regression in their respective languages, showcasing the differences in syntax and library support for AI applications. Python and R offer the most straightforward syntax for data science tasks, while Julia provides high performance with a syntax similar to Python's. Java, being more verbose, is less common for quick AI prototyping but is invaluable for applications where performance and scalability are critical. Table 4 represents the comparison of the programming languages in a linear regression case study in terms of CPU execution speed in seconds. In Table 5, Comparison of the programming languages in terms of Line of Code (LOC) is represented In comparing programming languages based on lines of code and execution times, Python emerges as a balanced choice with its concise 16 lines of code and the fastest average execution time of



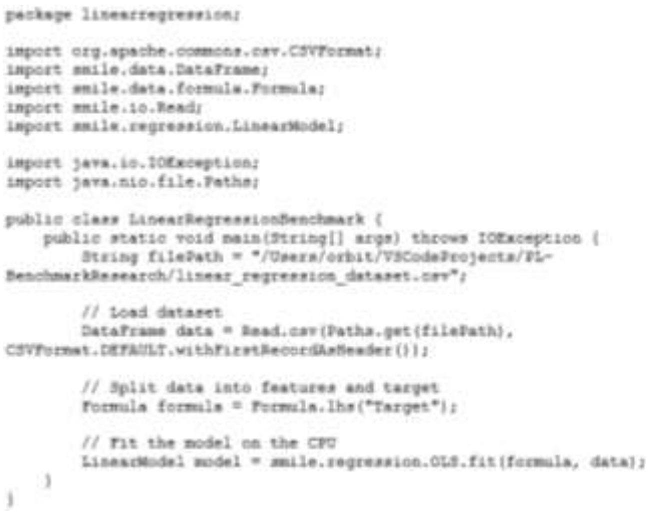
```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Load dataset
data = pd.read_csv("linear_regression_dataset.csv")

# Split data into features and target
X = data.drop(columns=['Target'])
y = data['Target']

# Initialize Linear Regression model
model = LinearRegression()

# Fit the model on the CPU
model.fit(X, y)
```

Figure 5. *Python source code for the case study*



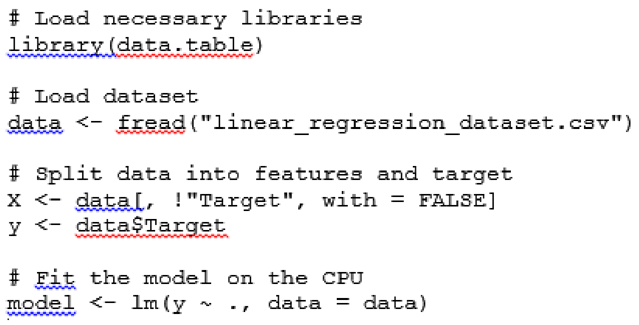
```
# import Pkg; Pkg.add("CSV")
# import Pkg; Pkg.add("DataFrames")
# import Pkg; Pkg.add("MLJLinearModels")
# import Pkg; Pkg.add("MLJ")
using CSV
using DataFrames
using MLJ

# Load dataset
data = CSV.read("/Users/orbit/VSCodeProjects/PL-
BenchmarkResearch/linear_regression_dataset.csv", DataFrame)

# Split data into features and target
X = select(data, Not(:Target))
y = data.Target

# Initialize Linear Regression model
LinearRegressor = @load LinearRegressor pkg=MLJLinearModels

# Fit the model on the CPU
model = machine(LinearRegressor(), X, y)
fit!(model, verbosity=0)
```

Figure 6. *Julia source code for the case study*



```
package linearregression;

import org.apache.commons.csv.CSVFormat;
import smile.data.DataFrame;
import smile.data.formula.Formula;
import smile.io.Read;
import smile.regression.LinearModel;

import java.io.IOException;
import java.nio.file.Paths;

public class LinearRegressionBenchmark {
    public static void main(String[] args) throws IOException {
        String filePath = "/Users/orbit/VSCodeProjects/PL-
BenchmarkResearch/linear_regression_dataset.csv";

        // Load dataset
        DataFrame data = Read.csv(Paths.get(filePath),
CSVFormat.DEFAULT.withFirstRecordAsHeader());

        // Split data into features and target
        Formula formula = Formula.lhs("Target");

        // Fit the model on the CPU
        LinearModel model = smile.regression.OLS.fit(formula, data);
    }
}
```

Figure 7. *Java source code for the case study*



```
# Load necessary libraries
library(data.table)

# Load dataset
data <- fread("linear_regression_dataset.csv")

# Split data into features and target
X <- data[, !"Target", with = FALSE]
y <- data$Target

# Fit the model on the CPU
model <- lm(y ~ ., data = data)
```

Figure 8. *R source code for the case study*

.***Table 4.*** *Comparison of the programming languages in terms CPU Execution duration in seconds*

Benchmark metrics	**Python (v3.11.4)**	**JAVA (v21)**	**Julia (v1.10.3)**	**R (v4.4.1)**
LOC (without benchmark code)	7	14	9	5
LOC (with benchmark code)	16	25	18	14

1,859516 seconds. Java, though requiring the most lines of code with 25 lines, provides relatively fast execution at 4,607241 seconds, suitable for performance-sensitive applications despite its verbosity. Julia, with 18 lines of code, is slower at 5,203379 seconds, possibly due to initial compilation overhead, and is best suited for tasks that leverage its strengths in numerical computing. Although R has the least number of code lines, it has the slowest speed, with an average execution time of 9,706835 in seconds. In the Table 6, an analysis of the Unadjusted Function Points (UFP) for the case study mentioned above with benchmarking code to measure the performance implemented in four examined programming language is given. The UFP is calculated based on the following components: External Inputs (EI), External Outputs (EO), External Inquiries (EQ), Internal Logical Files (ILF), and External Interface Files (EIF). Each component is assigned an average weight mentioned in the literature and the total weight for each language is computed.

- External Inputs (EI): Each implementation includes loading a CSV file, which is considered an external input with a total weight of 4.
- External Outputs (EO): Each implementation includes printing the start time, end time, and execution time, which are considered external outputs with a total weight of 15 (3 outputs, each weighted 5).
- External Inquiries (EQ): None of the implementations include external inquiries, resulting in a total weight of 0.
- Internal Logical Files (ILF): Each implementation involves three logical files (data frames or equivalent), resulting in a total weight of 30 (3 files, each weighted 10).

- External Interface Files (EIF): None of the implementations use external interface files, resulting in a total weight of 0.

The total UFP for each language implementation (Python, Java, Julia, R) is 49, shows us a consistent level of complexity and functionality across all four languages for this specific task.
Unadjusted Function Points are often preferred over adjusted function points because they focuse solely on the functional requirements as perceived by the end user, making it a straightforward metric for comparing different implementations. By avoiding the subjective adjustments, this metric ensures that the measurement remains unbiased and replicable.

Table 5. *Comparison of the programming languages in terms of Line of Code (LOC)*

	MAC M1 MAX CPU EXECUTION (seconds)					
	1st Run	**2nd Run**	**3rd Run**	**4th Run**	**5th Run**	**Avg.**
Python	1,898331	1,873851	1,850640	1,828339	1,846417	1,859516
Java	5,840410	4,899545	4,376204	3,840037	4,080009	4,607241
Julia	5,354401	5,147843	5,136030	5,210301	5,168317	5,203379
R	9,813953	9,742710	9,691265	9,690386	9,595861	9,706835

Table 6. *UFP Calculation Table for the Linear Regression and Benchmark Code*

UFP Calculation Table for the Linear Regression + Benchmark Code	**Weight**				**Input or Output**				**Total**				**Description**			
	Python	**Java**	**Julia**	**R**	**Python**	**Java**	**Julia**	**R**	**Python**	**Java**	**Julia**	**R**	**Python**	**Java**	**Julia**	**R**
External Inputs (EI)	4	4	4	4	1	1	1	1	4	4	4	4	Loading the csv file.	Loading the csv file.	Loading the csv file.	Loading the csv file.
External Outputs (EO)	5	5	5	5	3	3	3	3	15	15	15	15	Printing start time, end time, and execution time.	Printing start time, end time, and execution time.	Printing start time, end time, and execution time.	Printing start time, end time, and execution time.
External Inquiries (EQ)	4	4	4	4	0	0	0	0	0	0	0	0				
Internal Logical Files (ILF)	10	10	10	10	3	3	3	3	30	30	30	30	DataFrames ('data', 'X', 'y')	DataFrames ('data', 'formula', 'model')	DataFrames ('data', 'X', 'y')	DataFrames ('data', 'X', 'y')
External Interface Files (EIF)	7	7	7	7	0	0	0	0	0	0	0	0				
Total UFP									49	49	49	49				

5. Discussion and Conclusion

In this study we compared the Python, Java, R and Julia programming languages for the AI applications. First of all we discussed the applicability of the languages to AI projects, in terms of the decided features presented in the Table 1. In addition to that we developed a use case based on linear regression, and we provided the efficiency and execution speed parameters for each programming languages benefited from both computer science and software engineering areas. Case study results show that if fastest execution time is needed, Python is the best alternative. By its nature, Python is compatible with scripting, therefore for this use case the results present a conflicting result with current literature. However it is not surprising for the codes based on scripting, Python outperforms the other programming languages. In addition to that for applications where execution speed is crucial, Java is suitable for high performance computing, memory intensive operations and low level system operations. Julia's numerical capabilities outweighs its execution time. It should be noted that our case study 100% compatible to highlight the Julia's numerical capabilities. Our results are based on four major tasks in this case study; import necessary libraries, initialize sample data, fit a linear regression model, and finally print the execution time results. Our comparative analysis reveals insights into the strengths and limitations of programming languages in AI development. Python emerges as a versatile and widely adopted language with extensive library support for AI tasks. Especially when we consider cloud development platforms such as; Google Colab provides a comprehensive pre-loaded libraries for data science and AI specific tasks. Julia demonstrates promising performance advantages for specific use cases. But most of the online platforms the language has not supported pre-loaded libraries such as GLM. It is a major problem in terms of usability. As a result, Julia is much more dependent to local environments compared with Python. The choice of programming language should be informed by project requirements, development constraints, and long-term objectives. Future research directions may explore emerging languages and techniques to further enhance AI development practices. In future work, we plan to expand this use case by conducting experiments on both CUDA supported GPU and Apple Silicon processor with neural engine.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.

References

[1] A. Nagpal and G. Gabrani, (2019). Python for Data Analytics, Scientific and Technical Applications. *Amity International Conference on Artificial Intelligence (AICAI),* Dubai, United Arab Emirates, pp. 140-145, doi: 10.1109/AICAI.2019.8701341

[2] Rossum, G.V. (2007). Python Programming Language. *USENIX Annual Technical Conference*.

[3] S. Raschka and V. Mirjalili, (2019). Python Machine Learning. Sebastopol, CA: O'Reilly Media,

[4] R Development Core Team, (2008). R: A Language and Environment for Statistical Computing. Vienna, Austria: R Foundation for Statistical Computing.

[5] H. Wickham et al., (2016). ggplot2: Elegant Graphics for Data Analysis. New York, NY: Springer-Verlag.

[6] Arnold, K., Gosling, J., & Holmes, D. (2005). The Java programming language. Addison Wesley Professional. https://www.acs.ase.ro/Media/Default/documents/java/ ClaudiuVinte/books/ArnoldGoslingHolmes06.pdf

[7] W. Savitch, (2014). Java: An Introduction to Problem Solving and Programming," Upper Saddle River, NJ: Pearson

[8] Raff, E. (2017). JSAT: Java statistical analysis tool, a library for machine learning. *Journal of Machine Learning Research*, 18, 1-5

[9] C. S. Horstmann, (2008). Java Concepts: Compatible with Java 5, 6, and 7," Hoboken, NJ: Wiley

[10] K. Gao, G. Mei, F. Piccialli, S. Cuomo, J. Tu, Z. (2020). Huo Julia language in machine learning: Algorithms, applications, and open issues *Comput Sci Rev*, 37;100254

[11] Cabutto TA, Heeney SP, Ault SV, Mao G, Wang J, (2018). An overview of the Julia programming language. Proceedings of the 2018 *International Conference on Computing and Big Data;* 87-91. doi.org/10.1145/3277104.3277119.

[12] J. Bezanson et al., (2017). Why We Created Julia. *Proc. of the IEEE,* 104(11);18-22

[13] Lang PF, Shin S, Zavala VM. (2020). SBML2Julia: interfacing SBML with efficient nonlinear Julia modeling and solution tools for parameter optimization. *arXiv preprint arXiv*:201102597.

[14] V. B. Shah et al., (2017). The Julia Programming Language. *Sebastopol, CA:* O'Reilly Media.

[15] S. Danisch et al., (2019). Julia for Data Science. *Sebastopol, CA:* O'Reilly Media.

[16] Farooq MS, Khan SA, Ahmad F, Islam S, Abid A. (2024). An Evaluation Framework and Comparative Analysis of the Widely Used First Programming Languages. *PLoS ONE* 9(2): e88941. https://doi.org/10.1371/journal.pone.0088941

[17] Dave, S. (2023). Python Syntax: *The Art of Readability*. https://dev.to/souvikdcoder/python-syntax-the-art-of-readability-10b9 Retrieved: 08.06.2024

[18] CodeLikeAGirl (2023). https://www.codewithc.com/pythons-dynamic-typing-memory-costs/ Retrieved: 08.06.2024