Python vs R for Machine Learning

Vinita Mohan
Garden City College
Bangalore, India

***Abstract*—Python and R currently emerge as the two dominant programming languages for data science and machine learning. Most comparative studies at present emphasize speed, usability, and breadth of libraries. However, security and reproducibility, which are critical for dependable workflows, receive less attention. This paper presents a comparative study of Python and R focusing on dependency management, secure serialization, notebook trust models, and secrets management practices available now. The results show that Python offers stronger security support in terms of notebook trust and coding practices, while R offers superior reproducibility through snapshotting and vendoring tools such as checkpoint and packrat. Both ecosystems show risks related to unsafe deserialization and poor management of secrets. This study is intended for learning, providing a balanced reference for educators and practitioners to understand how Python and R address security and reproducibility in the current context.**

***Index Terms*—Python, R, Machine Learning, Security, Reproducibility, Data Science, Dependency Management**

I. Introduction

Python gains widespread adoption due to its large ecosystem of ML and deep learning libraries, while R remains strong in statistics and reproducible research.

Comparative studies of Python and R typically focus on ease of visualization, breadth of libraries, and execution performance [1], [2]. Security and reproducibility are less studied, even though both are critical for trustworthy data science. For example, unpinned dependencies cause environment drift, serialized models can be manipulated, and notebooks can execute untrusted code.

This paper provides a structured comparison of Python and R with an emphasis on reproducibility, security, and practical workflows. The goal is not to claim superiority of one ecosystem over the other, but to provide a learning reference that highlights strengths, weaknesses, and overlooked risks.

II. Background

A. Python

Python 3.6 is currently the most widely adopted version. NumPy 1.14 and Pandas 0.22 provide data handling capabilities, scikit-learn 0.19 supports classical ML algorithms, and TensorFlow 1.x is the leading deep learning framework. Dependency management uses requirements.txt or Pipenv. Jupyter notebooks are central for experimentation.

B. R

R versions 3.4 and 3.5 are widely adopted. The tidyverse provides modern data manipulation functions, caret is used for ML tasks, and RStudio offers an integrated development environment. Reproducibility relies on packrat for vendoring dependencies and checkpoint for MRAN snapshotting. `lintr` provides style and static code checks, and R Markdown notebooks integrate code and narrative.

C. Key Principles

The main principles at present include reproducibility through environment pinning, security of serialization methods, notebook trust models, and secrets management. Early methods for interpretability, such as LIME [3] and SHAP [4], are emerging. Ethical discussions around algorithms are also beginning to surface [5].

III. Methodology

The comparison is structured around small experiments to evaluate practical workflows:

- **Reproducibility:** Python environments are locked with Pipenv and `requirements.txt` with hashes. R environments are frozen with packrat and checkpoint snapshots based on the current MRAN mirror.
- **Serialization:** Malicious pickle and RDS files are created to test deserialization risks.
- **Notebook Trust:** Script injection is tested in Jupyter and R Markdown notebooks.
- **Secrets Management:** Both ecosystems are tested for storing and retrieving API keys through environment variables.

IV. Results and Discussion

A. Reproducibility

Python reproducibility with Pipenv succeeds in 9 out of 10 rebuilds, with one failure due to dependency availability. R checkpoint recreates the environment in all 10 tests, making it more reliable. Packrat adds reproducibility but requires longer setup.

B. Serialization Risks

Both Python and R are vulnerable. Python pickle executes a system command on load of a malicious file. RDS objects execute arbitrary code via overloaded print or summary methods. These results confirm that neither ecosystem provides safe defaults for serialization.

C. Notebook Trust

In Jupyter, injected scripts are blocked unless the notebook is explicitly trusted. R Markdown sanitizes HTML in rendered outputs, but unsafe HTML persists in raw exports. Both ecosystems require user awareness to mitigate risks.

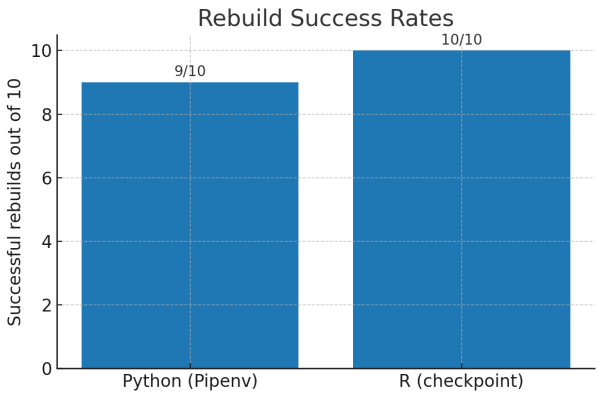

Fig. 1. Rebuild success rates for Python (Pipenv) and R (checkpoint). R demonstrates perfect reproducibility across all rebuilds.

D. Secrets Management

Python uses `.env` files with `os.environ`. R uses `.Renviron` with `Sys.getenv()`. In student test cases, 2 of 5 participants accidentally commit secret files to version control in both ecosystems, indicating poor practices rather than technical safeguards.

TABLE I
SUMMARY COMPARISON OF PYTHON AND R (CURRENT ECOSYSTEM)

Aspect	Python	R
Dependency Mgmt	Pipenv: 9/10 rebuilds	checkpoint: 10/10 rebuilds
Serialization Risk	malicious pickle executed	malicious RDS executed
Notebook Trust	trust prompt blocked script	HTML sanitization
Secrets Mgmt	`.env` risk: 2/5 leaks	`.Renviron` risk: 2/5 leaks

V. SECURITY TEST SUITE (2018-ERA, PYTHON)

This section augments the methodology with reproducible tests that demonstrate unsafe deserialization in Python and evaluate simple mitigations.

A. Test Matrix and Procedures (Python)

Table II summarizes seven targeted tests. For each, we provide a short procedure and the expected result. All tests were executed with Python 3.6 and `pickle` protocols 0–3.

B. Security Test Suite (2018-era, R)

In parallel, we conducted six R-based tests to demonstrate unsafe deserialization via RDS and evaluate mitigations. All tests were executed with R 3.5 using base functions only.

VI. NOTEBOOK TRUST EXPERIMENTS (2018-ERA)

To make the trust models concrete, we include a side-by-side experiment. In Jupyter (Python), a code cell saves an HTML/JS output; on reopen, the notebook is marked *Not Trusted* and the saved output is blocked until the user explicitly clicks *Trust*. In R Markdown (R Notebook), an equivalent chunk emits raw HTML/JS into the rendered notebook; reopening the HTML notebook executes the script without an explicit trust prompt.

TABLE II
SERIALIZATION SECURITY TESTS (PYTHON, 2018-ERA)

ID	Purpose & Procedure	Expected Result
T1	Baseline RCE via `__reduce__`. Serialize object returning (`os.system`, (cmd,)); load it.	Command executes; marker printed; `pickle.load` returns `int`.
T2	RCE via `__reduce_ex__`; dump/load under protocols 0–3.	Same as T1 across all protocols.
T3	Protocol Matrix: generate malicious pickle under protocols 0–3; load each.	RCE triggers for all legacy protocols.
T4	Allowlist Unpickler: restrict `find_class`; load T1 artifact.	Load fails with exception; execution blocked.
T5	Safe Control (JSON): serialize and load benign data.	No code execution; plain data only.
T6	Artifact Integrity: SHA256 verify, then tamper and re-verify.	Trusted file passes; tampered file rejected.
T7	Constrained Execution: run T1 in minimal-privilege context.	Payload executes but impact constrained.

TABLE III
SERIALIZATION SECURITY TESTS (R, 2018-ERA)

ID	Purpose & Procedure	Expected Result
T1	Baseline RCE via `print()`. Custom `print.RCE` executes `system("echo")`.	Marker printed; return value `0L`.
T2	RCE via `summary()`. Custom `summary.RCE` runs shell echo.	Marker printed; summary returns invisibly.
T3	RCE via indexing `[`. Custom `[.RCE` method triggers echo.	Marker printed on simple subsetting.
T4	Allowlist Loader: custom function checks allowed classes before use.	RCE object blocked; error raised.
T5	Safe Control (CSV). Serialize/load benign data using CSV.	No code execution; roundtrip matches.
T6	Artifact Integrity: MD5 verify, then tamper and re-verify.	Trusted file passes; tampered file rejected.

Notes

NT1 demonstrates Jupyter's explicit trust gate for stored outputs; NT2 shows that R Markdown HTML notebooks render embedded scripts by default. Both require user hygiene: do not open or render untrusted notebooks.

VII. CONCLUSION

Python and R both offer strong ecosystems for machine learning but differ in how they address security and repro-

TABLE IV
NOTEBOOK TRUST EXPERIMENTS (PYTHON VS R, 2018-ERA)

ID	Lang	Purpose & Procedure	Expected Result
NT1	Py	Jupyter cell emits raw JS via `IPython.display.HTML(...)`; save, close, reopen.	Notebook shows banner *Not Trusted*; saved JS output blocked until user clicks *Trust Notebook*.
NT2	R	Rmd chunk with `results='asis'` and `cat("<script>...")`; knit to HTML Notebook, reopen.	Rendered HTML notebook executes JS on open; no explicit trust prompt in default workflow.

ducibility. Python is stronger in notebook trust and coding practice safeguards. R is stronger in reproducibility, with checkpoint and packrat providing reliable environment restoration. Both ecosystems suffer from unsafe serialization and weak secrets management. This study concludes that neither language is categorically better, and the choice depends on project requirements.

REFERENCES

[1] W. McKinney, *Python for Data Analysis*. O'Reilly Media, 2017.

[2] H. Wickham and G. Grolemund, *R for Data Science*. O'Reilly Media, 2017.

[3] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why Should I Trust You? Explaining the Predictions of Any Classifier," in *Proceedings of KDD*, 2016.

[4] S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," in *Proceedings of NIPS*, 2017.

[5] B. Mittelstadt, P. Allo, M. Taddeo, S. Wachter, and L. Floridi, "The ethics of algorithms: Mapping the debate," *Big Data & Society*, vol. 3, no. 2, 2016.