

FinSight AI: An Integrated Platform for Stock Market and Mutual Fund Recommendation

Karishma Urkude¹, Prathmesh Bhoyar², Shreya Mahajan³, Vedant Zod⁴, Ishani Karemore⁵
^{1,2,3,4,5}*Department of Artificial Intelligence and Engineering,
G H Rasoni University, Amravati, Maharashtra, India
doi.org/10.64643/IJIRT209380*

Abstract—FinSight AI is a Streamlit application that brings historical stock analysis and personalized mutual fund recommendation into one local tool. This paper describes what the repository really contains and where its limits lie. A stock dashboard and a fund recommender share a home page and a Python calculation engine, and everything runs from two static CSV files: 117,884 daily stock records for 336 names in 20 countries (January 2025 to May 2026), and 771 mutual fund records with 35 columns. A questionnaire produces a 0–100 risk score, which sets an Equity, Hybrid and Debt mix. Funds are then ranked with a six-factor weighted score, and corpus growth is projected at fixed rates of 8%, 10% and 12%. The analytics are descriptive and rule-based. There is no live feed, forecasting model, trading, database or login, and the source of both datasets is undocumented. We list the validation and governance work still needed before the outputs could inform real decisions.

Index Terms—Financial Analytics, Stock Dashboard, Mutual Fund Recommendation, Risk Profiling, Goal-Based Allocation, Decision Support, Rule-Based Systems

I. INTRODUCTION

A. Motivation

People who invest usually juggle several questions at once. How has a stock behaved, and how risky is it? How does one market compare with another? How much loss can this particular investor stomach, and what will a monthly contribution grow into? The answers tend to sit in separate places: charts here, fund factsheets there, a calculator somewhere else. Putting them side by side helps only if the user can see where each number came from and where the data ends.

FinSight AI is our attempt at that. It is a browser-based Streamlit application [1] that combines stock exploration, risk profiling, goal-based allocation, fund ranking and scenario projection. It runs on local files

and is meant for exploration and teaching. It makes no attempt to replace a broker or to give regulated advice.

B. Scope and Objectives

Portfolio theory, fund performance measures and risk-tolerance questionnaires are all well studied [2]– [5], and robo-advisory has been surveyed [6]. We propose no new financial model. The contribution here is practical. We show how two local datasets, one questionnaire and one rule-based engine produce a dashboard, an allocation, a fund ranking and a projection, and we are exact about what that does and does not demonstrate. Specifically, we (1) document the architecture, (2) define the stock indicators and recommendation rules, (3) profile the bundled data as recorded, and (4) state the limits on interpreting any output.

II. RELATED WORK

A. Theory and Risk Profiling

Mean–variance selection made the trade-off between expected return and variance explicit and explained why diversification pays [2]. Advisers then ask a second question, how much risk this person will accept, and usually answer it with a questionnaire. Validated versions exist [3]. Our score follows the same pattern but uses hand-picked weights, so we do not present it as a validated psychometric instrument.

B. Fund Measures and Robo-Advice

The Sharpe ratio relates excess return to total volatility [4]; the Sortino ratio swaps in downside deviation [5]. Both feed our fund score, and both are read from the dataset rather than recomputed from prices. Robo-advisers automate profiling, allocation and monitoring under regulatory duties [6]. The prototype covers profiling, allocation and ranking only. It cannot

monitor, rebalance, link accounts or execute orders. Its novelty lies in the assembly, not in any single algorithm, and we ran no comparison with commercial tools, no backtest and no user study.

III. SYSTEM ARCHITECTURE AND DATA

A. Architecture

The application has three layers (Fig. 1). Three Streamlit pages form the interface: `dashboard_app.py` handles onboarding, `stock_app.py` the stock dashboard and `mf_app.py` the fund recommender. Beneath them, stock metrics are computed in the page code while `mf_engine.py` holds the financial logic. At the bottom sit two CSV files in the project root. There is no database, API service or authentication layer. A user action travels from a widget to session state, into an in-memory pandas DataFrame [7], through a calculation function, and back out as a Plotly [9] chart or panel. The logic is deterministic: same inputs, same outputs.

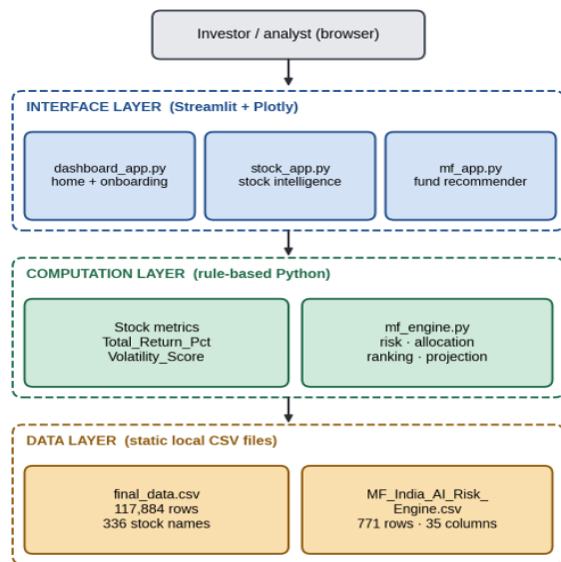


Fig. 1. Three-layer architecture: Streamlit pages, rule-based computation and static CSV data.

B. Datasets

`final_data.csv` holds 117,884 daily records for 336 stock names in 20 countries, dated 2025-01-01 to 2026-05-31, or about 351 rows per name. `MF_India_AI_Risk_Engine.csv` holds 771 fund schemes in 35 columns. The two files share no keys and are never joined. Nothing in the repository says where they came from, how they were collected or

what license applies, so we treat them as bundled project data and not as official or representative market statistics. Table I lists the fields the code uses (Fig. 2 shows the two files side by side).

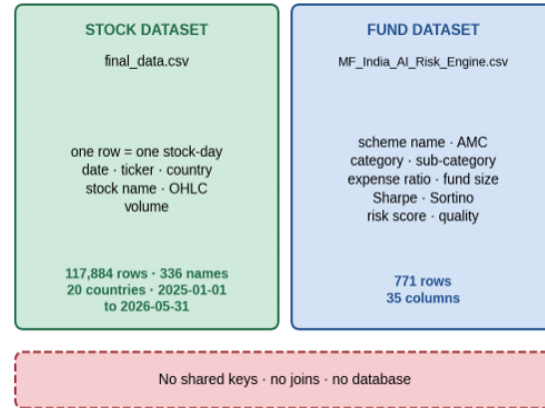


Fig. 2. The two bundled datasets: independent files with no joins or shared keys.

Table I
Dataset Fields Used by the Code

Dataset	Fields	Role
Stock	date, ticker, country, stock name	Filtering, search, grouping
Stock	open, high, low, close, volume	Price analysis, volume totals
Stock (derived)	Total_Return_Pct, Volatility_Score	Return and risk comparison
Fund	scheme name, AMC, category, sub-category	Identity, asset bucket
Fund	expense ratio, fund size, return metrics	Cost and return terms
Fund	Sharpe, Sortino, risk score, quality indicators	Ranking terms

C. Processing

Processing happens in memory (Fig. 3). The code converts financial fields to numbers, assigns each fund to an Equity, Hybrid or Debt bucket by matching category strings, filters by country, dates or eligibility, aggregates, and derives scores. Bucket assignment is only as reliable as the labels. A differently spelled category could land in the wrong bucket, since no canonical mapping is described, and no validation of missing values, duplicates or outliers is documented. Updating the data means replacing a file; nothing refreshes on a schedule.

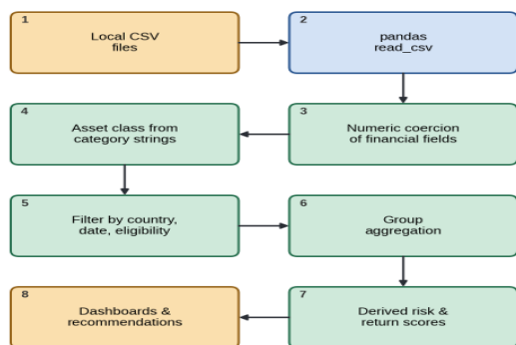


Fig. 3. Data path from local CSV files to dashboards and recommendations.

D. Execution Scope

The repository ships no deployment manifest, container configuration or hosted-service definition. Fig. 4 shows the resulting picture: a browser talking to one local Streamlit process that reads two files. Backend, database, authentication and cloud hosting are all absent, so we claim no live deployment, uptime or multi-user operation.

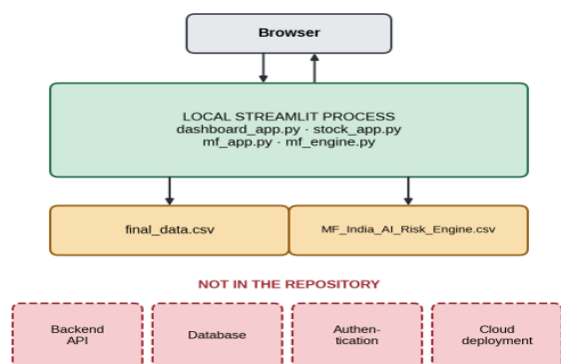


Fig. 4. Local execution model; absent services are shown dashed.

IV. STOCK ANALYTICS MODULE

The dashboard in stock_app.py has a sidebar for country, stock search, stock selection and date range. The filtered data drives KPI cards (stock count, country count, data volume, leading performer), stock-level charts, country comparisons, a data-quality summary and a ranking view (Fig. 5). Indicators include daily closing behavior, total return percentage, a volatility score, moving-average comparisons, price extremes, and country averages with aggregate volume. Total_Return_Pct and Volatility_Score are computed from the daily rows and then aggregated, so securities and markets can be compared on return and risk. Prices stay in the dataset's local units; no currency conversion is described.

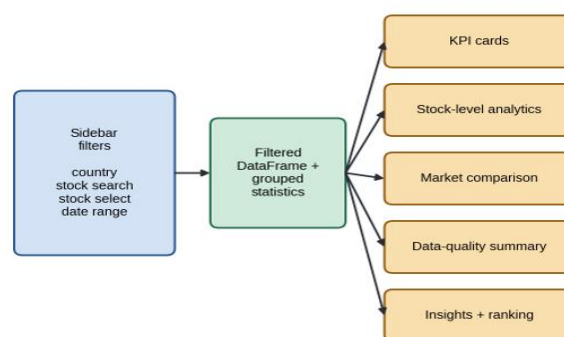


Fig. 5. Stock module: sidebar filters drive five dashboard outputs.

All of this is descriptive. There is no forecast, trading signal or strategy backtest, and a strong return in the selected window says nothing certain about the next one. Table II maps each feature of the system to its inputs, processing and outputs.

Table II
System Features and Data Flow

Feature	Input	Processing	Output
Home and onboarding	User selection	Routes to a module	Stock or fund workflow
Stock KPIs	Filtered rows	Counts, leading performer	KPI cards
Stock analytics	Stock and dates	Return, volatility, extremes, moving averages	Charts, tables
Market comparison	Country choice	Country averages, aggregate volume	Comparison panels
Data-quality summary	Loaded dataset	Dataset characteristics	Summary section
Risk profiling	Questionnaire answers	Weighted score, rescaled to 0–100	Score and label
Allocation	Goal, horizon, risk	goal_strategy(), adjust_for_risk()	Equity, Hybrid, Debt mix
Fund ranking	Fund data, bucket	Weighted six-factor score	Top fund per bucket
Projection	SIP or lumpsum, horizon	Future value at three rates	Corpus and gains

V. MUTUAL FUND RECOMMENDATION ENGINE

A. Workflow

The recommender (mf_app.py and mf_engine.py) walks the user through a multi-step form (Fig. 6): personal details, goal, investment type, target corpus, a risk questionnaire and a time horizon. Session state carries answers forward, and the last step displays the portfolio. The user drives every step.

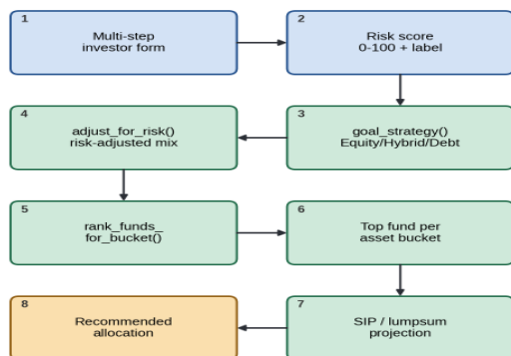


Fig. 6. Recommendation pipeline from questionnaire to allocation.

B. Risk Score and Allocation

calculate_investor_risk() combines six answers with fixed weights (loss reaction, decision priority, experience, income stability, comfort with fluctuations and primary priority) into a 0–100 score, then maps it to one of five labels from Very Conservative to Very Aggressive (Fig. 7). Someone chose those weights by hand; nothing was learned from data. goal_strategy() then sets Equity, Hybrid and Debt percentages from the goal and horizon, and adjust_for_risk() shifts the mix, toward safer holdings for conservative profiles and toward equity for long-horizon aggressive ones.

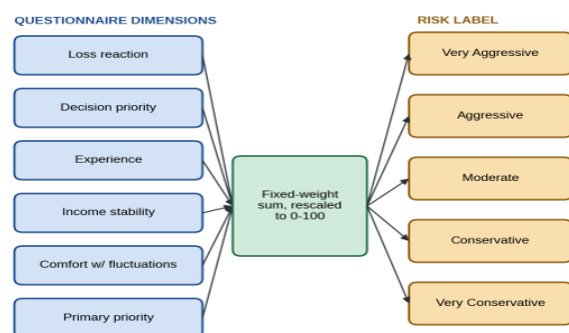


Fig. 7. Six questionnaire dimensions become a 0–100 score and a five-level label.

C. Ranking and Projection

rank_funds_for_bucket() scores each candidate on six weighted factors: risk compatibility, quality score, return, Sharpe contribution, Sortino contribution and expense ratio (Fig. 8). The top fund per bucket forms the suggested portfolio. Every score traces to a formula, but the weights are heuristic, and we did no sensitivity analysis or calibration.

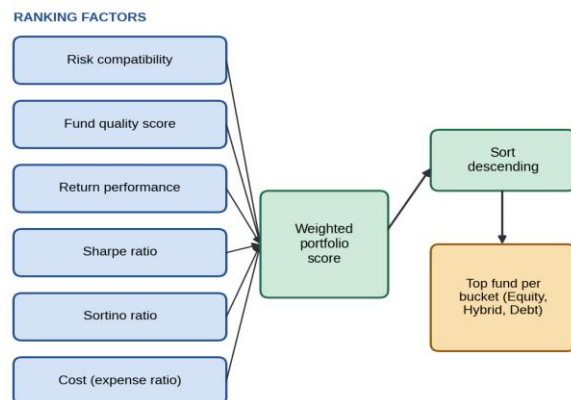


Fig. 8. Six factors feed a weighted score; the top fund per bucket is chosen.

Future corpus is estimated from monthly SIP or lumpsum amounts at 8.0%, 10.0% and 12.0% a year (Fig. 9), with projected gains shown against principal. These are planning scenarios, not forecasts. They do not come from the fund data, and inflation, taxes, fees and return uncertainty are not modeled. Table III summarizes the engine functions.

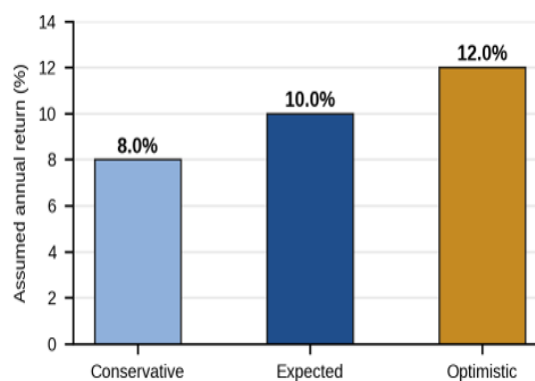


Fig. 9. Fixed annual return scenarios used in the projections.

Table III
Engine Functions

Function	What It Does	Output
calculate_investor_risk()	Weights six answers into a score and label	Risk score, label
goal_strategy()	Sets a base mix from goal and horizon	Equity, Hybrid, Debt targets
adjust_for_risk()	Shifts the mix by risk level	Adjusted allocation
rank_funds_for_bucket()	Scores and sorts funds on six factors	Ranked funds per bucket
Projection functions	SIP and lumpsum value at three rates	Corpus and gains

VI. IMPLEMENTATION AND INTERFACE

The repository is small and keeps interface and logic apart: three-page files, `mf_engine.py` and two datasets. Dependencies are Streamlit, pandas, NumPy, Plotly and Python's math module. There is no API layer, database, container setup or authentication code. Because the engine sits apart from the pages, its rules can be read and changed without touching the interface. The interface uses a dark, high-contrast theme with KPI cards, comparison tables and risk labels. That makes the formulas approachable, though polish can imply more certainty than static data and heuristic weights warrant. We ran no accessibility audit or cross-device test. Controls apply to one module at a time, because the modules share a framework but no filter state. Fig. 10 shows the whole path from data to a human decision.

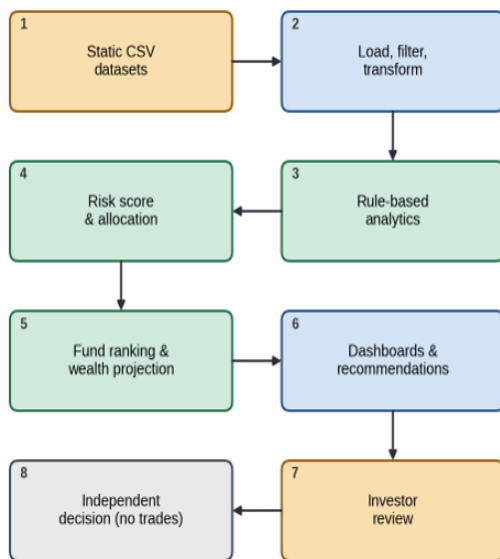


Fig. 10. End to end: static data, rule-based analytics, recommendations, then the investor decides.

VII. EVALUATION AND DISCUSSION

A. What We Checked

We limited ourselves to checks anyone can repeat from the repository: a file-level audit of the app and engine modules, a profile of the two datasets, and a review of how each function maps inputs to outputs. There is no automated test suite, benchmark, backtest or user study. The audit confirmed that every file described in Section VI exists, that the workflows behave as described, and that the datasets match the figures above. The risk, allocation, ranking and projection functions form a deterministic, readable pipeline.

B. What Remains Open

These checks show what the software is. They do not show numerical correctness against an outside reference, whether a recommendation suits a real investor, predictive accuracy, speed or usability (Table IV). A fuller evaluation would start with data lineage, check the engine against independent calculations of returns, volatility, SIP value and weighted scores, test ranking sensitivity to the weights, compare risk labels with an established instrument, and run a usability study. Any forecasting feature would need a stated target, a temporal holdout, baselines and uncertainty reporting.

Table IV
Evidence and Open Questions

Check	Evidence	Not Established
Repository audit	Files and modules inspected	Runtime correctness
Dataset profile	Recorded row and column counts	Provenance, completeness
Logic review	Engine functions read	Weight calibration
Predictive accuracy	No model exists	Any forecasting claim

Backtest, user study, benchmark	None performed	Performance, usability, speed
---------------------------------	----------------	-------------------------------

C. Discussion

The main result is integration: one codebase links market exploration to a goal-based recommender, with logic kept apart from the pages. Rule-based design aids explainability but fixes the weights and scenario returns as unvalidated assumptions. The fund file has only 771 schemes and both datasets end in May 2026, so a “top fund per bucket” is the output of one scoring rule on one snapshot. It is not a prediction, and it is not advice to buy.

VIII. ETHICS AND LIMITATIONS

A. Ethical Considerations

Investment output can sway real decisions. Here the risk label rests on an unvalidated questionnaire and the projections on fixed rates, so both should be framed as educational aids, with a clear note that past performance guarantees nothing and that this is not regulated advice. The app stores no accounts and writes nothing to a database, but the questionnaire collects personal and financial details held in session state. A hosted version would need consent, retention and access controls the repository lacks, along with authentication, input validation and a security review. Explicit rules are inspectable yet can still encode bias, for instance by reading certain age or income answers as low risk tolerance, so the weights should be documented and tested.

B. Limitations

The data are static, with unknown provenance, license and update path, so completeness and representativeness cannot be judged. The stock data end in May 2026, and no as-of date is given for the funds. No predictive model or evaluation pipeline exists, ranking and risk scores use hand-set weights, projections use three fixed rates, and the questionnaire has not been validated. The platform places no trades, monitors no portfolios and links to no accounts (Fig. 11). Table V pairs each limitation with its impact, how the prototype currently copes, and a possible fix.

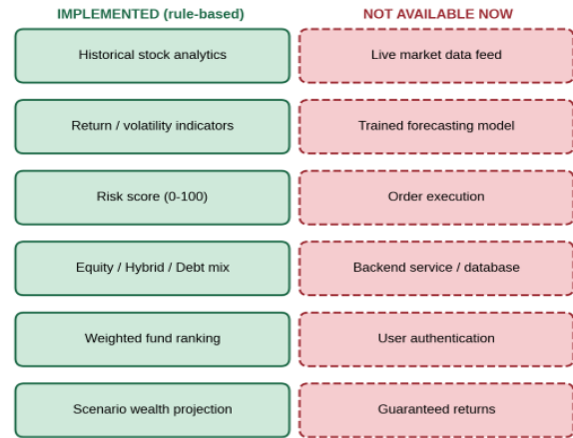


Fig. 11. Implemented rule-based features against capabilities not available in this version.

IX. FUTURE WORK AND CONCLUSION

A. Future Work

We would sequence the work in three phases (Fig. 12), with data governance first.

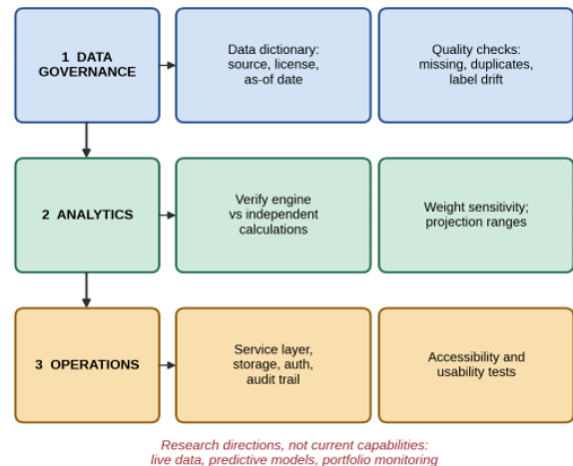


Fig. 12. Proposed roadmap: data governance, then analytics, then operations.

1) Data Governance: Write a data dictionary giving the source, license, as-of date, update process and field definitions for both datasets, and add checks for missing values, duplicates and category-label drift before any file is replaced.

2) Analytics: Verify the engine against independent calculations, test the weights for sensitivity and calibrate them with domain input. Show projection

ranges instead of single rates, and include inflation, fees and taxes. The stock module could gain benchmark comparison, drawdown and risk-adjusted return over clearly defined windows.

3) Operations: Add a service layer, durable storage, authentication and an audit trail, then test accessibility and usability with real users. Live data, predictive modeling and portfolio monitoring remain research directions, each needing a defined target, leakage-resistant validation, baselines, fairness and privacy review, and human oversight.

B. Conclusion

FinSight AI shows that risk profiling, goal-based allocation, weighted ranking and scenario projection can share one explainable workflow on top of two static datasets. It is not a trading platform, a deployed service or a forecasting system. Before anyone relies on its output, the data provenance needs documenting, the engine needs independent verification, and the weights and questionnaire need validation. Until then it is an exploratory and teaching tool.

REFERENCES

- [1] Streamlit Inc., “Streamlit documentation.” [Online]. Available: <https://docs.streamlit.io>. Accessed: Oct. 5, 2026.
- [2] Markowitz, H., “Portfolio selection,” *The Journal of Finance*, vol. 7, no. 1, pp. 77–91, 1952.
- [3] Grable, J. E., and R. H. Lytton, “Financial risk tolerance revisited: The development of a risk assessment instrument,” *Financial Services Review*, vol. 8, no. 3, pp. 163–181, 1999.
- [4] Sharpe, W. F., “Mutual fund performance,” *The Journal of Business*, vol. 39, no. 1, pt. 2, pp. 119–138, 1966.
- [5] Sortino, F. A., and L. N. Price, “Performance measurement in a downside risk framework,” *The Journal of Investing*, vol. 3, no. 3, pp. 59–64, 1994.
- [6] Jung, D., E. Glaser, W. Köpplin, A. Kraus, and D. Schmidt, “Robo-advisory: Digitalization and automation of financial advisory,” *Business & Information Systems Engineering*, vol. 60, no. 1, pp. 81–86, 2018.
- [7] McKinney, W., “Data structures for statistical computing in Python,” in *Proc. 9th Python in Science Conference (SciPy 2010)*, 2010, pp. 56–61.
- [8] Harris, C. R., et al., “Array programming with NumPy,” *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020.
- [9] Plotly Technologies Inc., “Collaborative data science,” Montreal, QC, Canada, 2015. [Online]. Available: <https://plot.ly>.