Learning Data Visualization in Python Utilizing an Autograding and Feedback System

Nikola Dimitrijević[1,*], Nemanja Zdravković[1] and Vijayakumar Ponnusamy[2]

[1] *Faculty of Information Technology, Belgrade Metropolitan University, Tadeuša Košćuška 63, 11000 Belgrade, Serbia*
[2] *Department of Electronics and Communication Engineering, SRM Institute of Science and Technology, Kattankulathur, Chennai, India*

Abstract

In this paper, we propose an interactive tool for Python data visualization libraries in order to overcome some of the typical problems encountered in other, predominantly online, Jupyter-based courses. We also set up an automated grading system that can automatically assess solutions to multiple test cases for students.

The widespread use of the Python programming language over the past few decades has brought interest in to a peak with an ever-increasing amount of people realizing its potential for incorporation into various educational environments from primary school to the university setting. Python – being one of the most common and popular languages for data analysis, visualization and machine learning, the python ecosystem has a richer set of libraries providing power for data visualization. Over the last few years, many data visualization libraries for Python have been created and this often renders researchers and analysts making a choice for a specific application very confusing not only because there are relatively many libraries but also many competing libraries, could work for different situations.

Additionally, we include a curriculum for learning these libraries, as well as a refresher on Python with all necessary libraries as prerequisites.

Keywords

data visualization, elearning, jupyter, python

1. Introduction

The ability to effectively visualize data is becoming an essential component of data analytics and the fields of data science and applied artificial intelligence (AI). Data visualization allows analysts, researchers, and students to convert complex data into informative graphics. This conversion facilitates easier pattern recognition, trend analysis, and helps make well-informed decisions [1, 2]. As organizations and research bodies increasingly rely on data visualization to communicate complex findings, the demand for these skills is growing significantly. Despite its importance, learning data visualization may be challenging for trainees who lack experience in programming, certain statistical concepts, or the tools used for creating visualizations [3]. Furthermore, confusion and ineffective learning often result from the wide array of available software and libraries and the absence of structured guidance in numerous online resources [4]. However, understanding and learning can be increased by conveying complex information intuitively and clearly.

Python has become a leading language for data visualization, thanks to its simplicity, versatility, and an extensive range of libraries, including Matplotlib, Seaborn, Plotly, and Bokeh. Python packages include libraries that enable data visualization and provide the ability to build specialized software. These libraries possess powerful tools that let users create everything from basic bar charts to intricate, interactive dashboards [5, 6]. It is a general-purpose language that allows smooth integration with machine learning models, data processing, and computations. As a result, it is favored by both academia and industry. Python's nature as an open-source platform reduces barriers to entry significantly when compared to proprietary software like MATLAB, which requires costly licensing fees citeozgur2017matlab.

Proceedings for the 15th International Conference on e-Learning 2024, September 26-27, 2024, Belgrade, Serbia

*Corresponding author.

nikola.dimitrijevic@metropolitan.ac.rs (N. Dimitrijević); nemanja.zdravkovic@metropolitan.ac.rs (N. Zdravković); vijayakp@srmist.edu.in (V. Ponnusamy)
0000-0002-6595-9277 (N. Dimitrijević); 0000-0002-2631-6308 (N. Zdravković); 000-0002-3929-8495 (V. Ponnusamy)

This makes Python an affordable option for students, researchers, and professionals working in educational and professional contexts. Unlike proprietary tools, Python's ecosystem supports innovation and widespread accessibility without requiring extensive financial resources. Moreover, Python's capabilities offer robust data visualization that can be integrated into numerous applications. Python, being adaptable and comprehensive, allows users to innovate without restriction. Its popularity continues to rise, reinforcing its status as a vital tool in modern data visualization.

Python is widely used, but other programming languages also have important roles in data visualization. For example, R is appreciated by statisticians and data scientists because of its rich visualization packages, like ggplot2 and Shiny. These tools permit sophisticated statistical graphics and interactive web applications [8]. R, however, can be complex and less intuitive for beginners compared to Python, especially for those who lack a statistics background. In contrast, Python is popular due to its robust plotting functions and seamless mathematical computations. However, MATLAB has the downside of being costly since it is commercial software requiring licensing fees. This aspect makes it less attractive than other tools. MATLAB's ecosystem is less flexible compared with modern data science frameworks and open-source tools. The primary disadvantage is that MATLAB's high cost limits its accessibility, particularly for individuals without resources. The integration capabilities with modern data science frameworks are also less flexible in MATLAB compared to the environment within Python [9].

Considering the numerous benefits of Python, such as its accessibility, ease of learning, and performance, it has undoubtedly become the go-to language for developing data visualization solutions. For those interested in learning data visualization in Python, many online courses and tutorials rely heavily on Jupyter-based notebooks. These often lack real-time feedback and structured assessments, making it challenging for learners to evaluate their understanding or get immediate corrections when errors are made. We propose an interactive learning tool to address this issue, which incorporates an automated grading and feedback system to assist students in comprehending data visualization, its curriculum structure, and the benefits of the automated learning environment for Python's data visualization. By enhancing the learning process, students are better equipped to increase their knowledge of the subject matter.

The remainder of the paper is organized as follows. Section 2 highlights some related works in data visualization and in autograding systems, which form the basis of our proposed model. Afsterwards, Section 3 and 4 show our approach to designing a course on data visualization coupled with the curriculum, respectively. Section 5 shows the implementation itself, while Section 6 concludes the paper, with some indication of future work.

2. Related work

The integration of automated grading systems and interactive learning environments has been a growing trend in computer science education. With the increasing adoption of programming languages like Python in both academia and industry, various tools and frameworks have been developed to facilitate learning, provide real-time feedback, and automate the assessment of programming exercises. In this section, we review several existing approaches related to automated grading tools, interactive learning platforms, and their impact on programming education.

2.1. Learning Data Visualization in Programming Courses

Data visualization is a crucial skill for students and professionals alike in technical fields such as statistics, engineering, computer science, information technologies, and data science. Effective data visualization can enhance comprehension, and can allow individuals to detect patterns, identify correlations, and ultimately communicate insights more effectively. The integration of data visualization into programming courses has been widely studied, with Python emerging as the preferred language due to its rich ecosystem of libraries and ease of use. However, structured approaches to teaching data visualization remain an ongoing challenge in programming education.

One of the key aspects of integrating data visualization into programming curricula is ensuring that students can engage with visualization techniques interactively while receiving structured feedback. The authors of [10] explore the use of visualization techniques in Python programming courses, highlighting the benefits of using graphical representations to enhance learning outcomes in non-computer science students. Their study emphasizes that data visualization can serve as a bridge for students transitioning into computational thinking, making it a valuable pedagogical tool.

Several researchers have also investigated how program visualization techniques impact student engagement and understanding. By introducing Jype, an interactive program visualization and exercise tool for Python, the authors in [11] designed the tool to support beginners in learning programming logic. Similarly, the integration of data visualization into programming exercises showed students to grasp fundamental concepts more effectively at the K–12 level [12]. Their findings suggest that visual representations of code execution significantly improve learning retention and reduce conceptual misunderstandings.

Beyond general programming, data visualization plays a fundamental role in specialized areas like machine learning (ML) and big data analytics. A recent study was done in [13], where the authors used Python-based visualization techniques to analyze educational datasets, reinforcing the importance of real-world, project-based applications in learning environments. Additionally, [14] discusses the development of interactive tutorials for teaching data visualization using Python's scikit-learn and other libraries, demonstrating how hands-on activities can enhance student engagement.

Indeed, despite the overall progress in integrating interactive visualization tools into programming education, challenges still remain in automating feedback and assessment. Most currently existing platforms, including Jupyter-based courses, provide an interactive coding environment; however, they lack structured grading mechanisms for visualization exercises. For instance, the authors in [15] investigate the use of PythonTutor as a visualization tool for introductory programming, concluding that automated feedback improves student performance by helping them correct errors in real time. Their study however primarily focuses on debugging rather than evaluating visualization quality, thus leaving a gap in assessing aesthetic, readability, and interpretability aspects of student-generated visualizations.

To address these gaps, our proposed approach extends existing autograding methodologies by incorporating structured feedback for data visualization assignments. By automating the evaluation of visualization outputs and ensuring adherence to best practices, we aim to provide a comprehensive, interactive learning experience for students learning data visualization in Python.

2.2. Autograding tools for Programming Education

The general concept of autograder systems has been present in Higher Education Institutions (HEIs) for over half a century. Autograders help teaching staff such as professors and teaching assistants by reducing the work load of having to manually grade all students' programming assignments. The bias of the teaching staff is removed as well, as the autograder always reports objectively. This is especially important for students in the fields related to data visualization, as the vast majority of their assignments would in the form of writing a computer program to run a specific task and present some data. Autograders have evolved with the emergence of new technologies and programming languages [16, 17]. The first generation of autograders was simple, and were mostly tied to lower-level programming languages. An autograder would yield a type of "True" or "False" answers, and the answer was determined by checking a strict set of successive instructions written by the student. The second generation of autograders often employed tools, which came with the operating system, and programming languages such as C, C++ and Java were used to build the autograders. In turn, these systems could check assignments in their respective language. The third and last generation of autograders emerged with the rise of the high-speed Internet and modern web development technologies. In such systems and platforms, a web application is hosted on a server with a user interface (UI) to the student. A student can write their program in a browser, without the need of an interpreter/compiler or an development environment software installed on their computer [18, 19].

3. Proposed Approach and implementation

We first analyzed the current available tools for data visualization in Python, i.e. the libraries, and how to find the best manner to incorporate the tools in a education-friendly environment such as Jupyter.

3.1. Matplotlib

Matplotlib is one of the most widely used data visualization libraries in the Python programming language. It provides a low-level, yet flexible interface for generating both static and animated visualizations, and interactive ones as well. Matplotlib is designed to work with NumPy arrays and allows users to create a wide range of plots, including line plots, scatter plots, bar charts, histograms, and more. Its versatility has made it the foundation of several other Python visualization libraries, including Seaborn and Plotly [20].

Matplotlib is particularly effective for scientific computing and academic research, where reproducibility and fine-grained control over visual elements are crucial. It is commonly used in engineering and data science applications. As demonstrated [21], Matplotlib is highly effective for presenting complex data in Python-based environments. The use of Matplotlib is also good for more complex visuals like 3D data visualization, especially within the Jupyter Notebooks, therefore demonstrating its capability to visualize multidimensional datasets interactively [22].

3.2. Seaborn

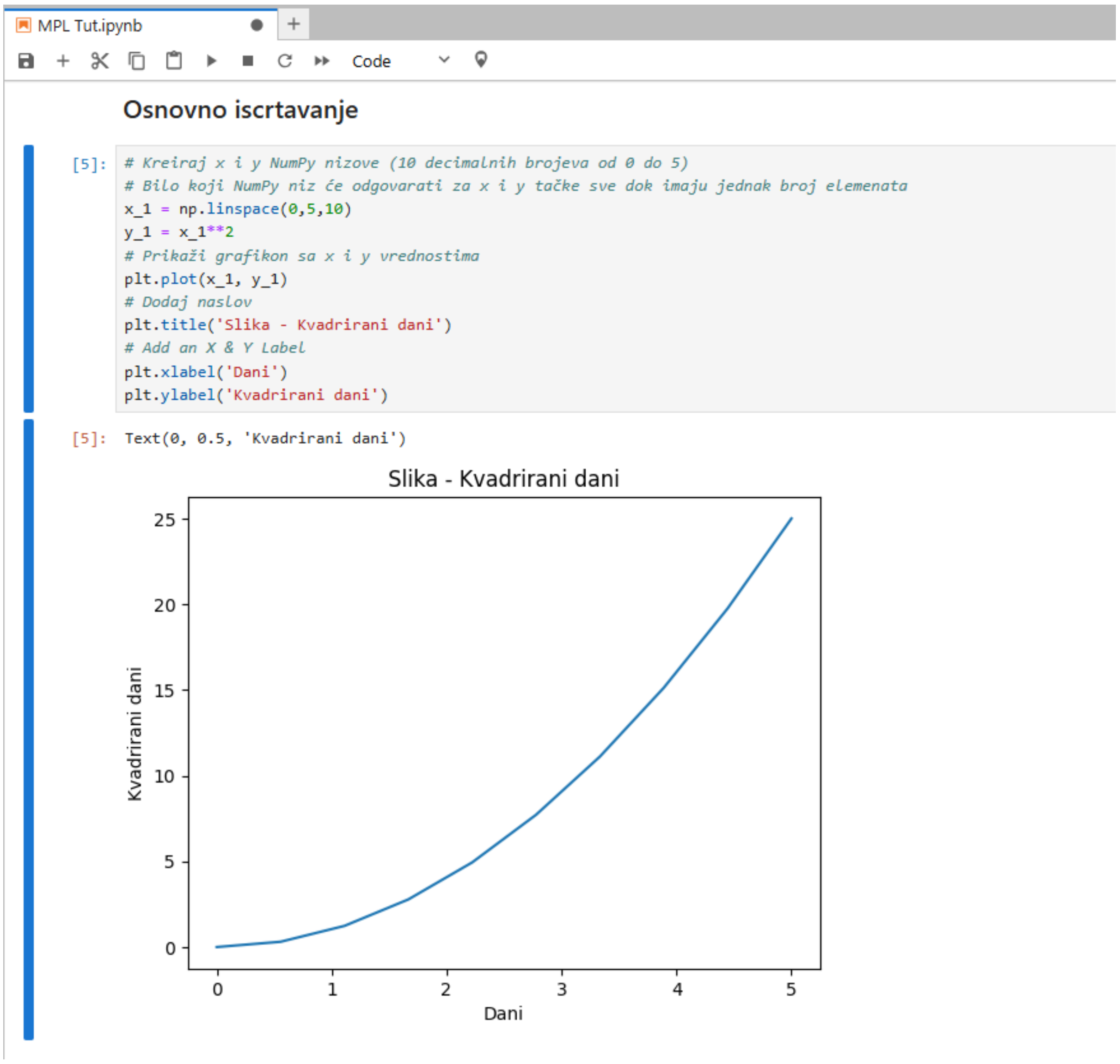
Built on top of Matplotlib, Seaborn is a high-level visualization library. It simplifies the process of creating aesthetically pleasing statistical visualizations by providing built-in themes and more intelligent default settings. Compared to Matplotlib, Seaborn is designed for statistical data visualization first and foremost, making it especially useful in data analysis and ML applications. It integrates seamlessly with Pandas DataFrames, enabling quick visualization of complex datasets with minimal code. Studies such as in [23] emphasize Seaborn's advantages in generating insightful statistical visualizations, particularly for exploratory data analysis (EDA) in research and education. The authors of [24] also discuss how Seaborn enhances data storytelling by providing advanced features such as heatmaps, violin plots, and regression plots, which help in uncovering hidden trends in data.

3.3. NumPy

NumPy is the core numerical computing library in Python, forming the backbone for many data analysis and visualization tasks. It provides efficient array operations, enabling fast computation of large datasets, which is essential for visualizing high-dimensional data. NumPy is extensively used in ML, finance, and scientific research, where handling large matrices and performing numerical operations are critical. NumPy plays a crucial role in data preprocessing and transformation before visualization, acting as a bridge between raw data and graphical representation [25]. Additionally, Sundaram *et al.* highlight NumPy's role in feeding structured numerical data into visualization tools like Matplotlib and Seaborn, making it indispensable for effective data presentation [26].

3.4. Jupyter Notebooks

Jupyter Notebooks provide an interactive, web-based environment for writing and running Python code, making them a preferred tool for teaching and practicing data visualization. Jupyter supports inline plotting, enabling users to visualize data immediately within the notebook interface, which enhances the learning process and facilitates iterative data exploration. Jupyter's integration with Matplotlib, Seaborn, and NumPy allows seamless interaction with visualization tools. Jupyter has a significant role in real-world data visualization applications, as it demonstrates its utility in presenting data trends using Python libraries [27]. Additionally, the authors of [28] emphasize the use of Jupyter in data science education, where interactive coding helps students understand complex visualization concepts.



```
[5]: # Kreiraj x i y NumPy nizove (10 decimalnih brojeva od 0 do 5)
     # Bilo koji NumPy niz će odgovarati za x i y tačke sve dok imaju jednak broj elemenata
     x_1 = np.linspace(0,5,10)
     y_1 = x_1**2
     # Prikaži grafikon sa x i y vrednostima
     plt.plot(x_1, y_1)
     # Dodaj naslov
     plt.title('Slika - Kvadrirani dani')
     # Add an X & Y Label
     plt.xlabel('Dani')
     plt.ylabel('Kvadrirani dani')
```

```
[5]: Text(0, 0.5, 'Kvadrirani dani')
```

Figure 1: Test assignment using Matplotlib.

An example of an assignment for plotting a simple curve using Jupyter with Numpy and Matplotlib is shown in Fig. 1. Plots can be directly displayed within the notebook using the `%matplotlib inline` command. This will generate a high-quality, customizable plot inside the notebook, leveraging Matplotlib's versatility.

Figure 2 illustrates the workflow of an autograding system implemented within Jupyter Notebooks. The system is designed to evaluate students' code correctness and visualization outputs against predefined tests and provide structured feedback while storing results in an external database. The system follows a similar approach to our previous works, described in [19, 29, 30]. Below is a detailed breakdown of each component in the flowchart

1. **Task Cells**
   These are predefined problem statements provided to students inside the Jupyter Notebook. Each task cell describes a coding problem, typically including:
   - The dataset to be used;
   - The type of visualization required (e.g., bar plot, scatter plot);
   - Expected output format or specific requirements.

   This ensures standardization across all student submissions.

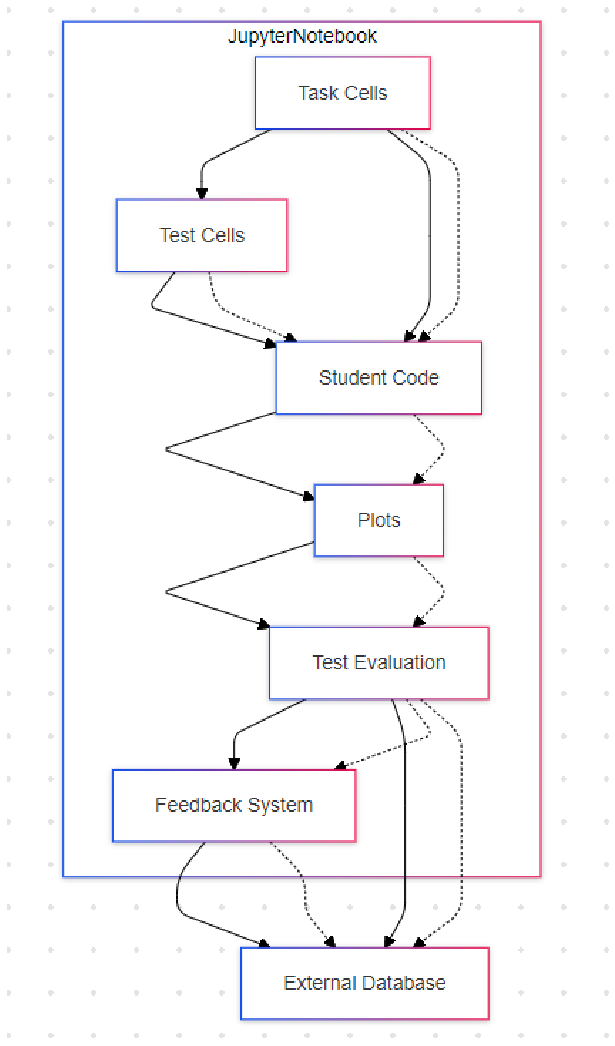


Figure 2: Workflow of the Jupyer autograding system.

2. **Test Cells**
   The test cells contain predefined test cases that automatically validate students' code. These tests check for:
   - Correct function outputs (e.g., whether a function returns expected numerical values);
   - Proper visualization elements (e.g., correct labels, title, color scheme);
   - Code execution correctness (e.g., avoiding runtime errors).

   In the workflow, the system compares student responses with expected outputs to ensure correctness. If a student's code deviates, the test fails, prompting feedback generation.
3. **Student Code**
   The student writes and executes code in response to the given tasks. Their code interacts with the predefined test cells, which either:
   - Pass the execution, confirming correctness;
   - Fail execution, requiring revisions.

   This allows the automatic evaluation system to check whether the solution meets the expected structure and logic.
4. **Plots Generation**
   If the student's solution involves data visualization, their code should generate plots. Evaluation Criteria:

    - Correct plot type (e.g., histogram vs. scatter plot);
    - Proper labeling (titles, axis labels);
    - Accuracy of plotted data points;
    - Formatting and aesthetics.

   The system captures the generated plots and compares them with reference outputs.
5. **Test Evaluation**
   This is the core of the grading system, where all student code outputs (including plots) are analyzed. The grading mechanism:
    - Runs predefined test cases on the student's code;
    - Compares expected vs. actual outputs (numeric values, plots, etc.). Checks for code efficiency, correctness, and adherence to problem constraints.

   If the student's solution meets all requirements, it passes; otherwise, it fails, triggering corrective feedback.
6. **Feedback System**
   After test evaluation, the system generates automatic feedback for students. The feedback includes:
    - Pass/Fail status for each test case; Error messages for failed tests, helping students debug their code; Hints or suggestions (if enabled) for improvement.

   This real-time feedback helps students iterate on their solutions before final submission.
7. **External Database**
   The final test results are stored in an external database for further use. The database stores student identifiers, test scores and evaluation metrics, execution logs (to track errors and improvements). This allows instructors to analyze student performance over time, track common mistakes and learning gaps, and to provide personalized learning suggestions.

4. Curriculum and Learning Path

Project-based learning (PBL) and interactive educational approaches have been widely recognized as effective strategies for teaching programming. The authors of [31] investigate the impact of project-based learning in programming courses and report positive effects on student motivation and problem-solving skills. Their study suggests that engaging students in hands-on projects leads to deeper learning and retention of programming concepts. Our approach builds on this idea by incorporating structured project-based assignments within an interactive Python visualization curriculum, ensuring that students not only learn visualization techniques but also apply them in practical scenarios.

The project-based course structure ensures that students learn by doing, with each module building towards real-world applications. Instead of just theoretical learning, every module will have a mini-project, and the course will culminate in a final capstone project where students apply everything they've learned. After completing the four phases, students can take the final capstone project where students select a real-world dataset (from Kaggle, government open data, or business case studies) and create a complete visualization project, presenting findings using Jupyter Notebooks. The summary of the modules is given in Table 1.

Before starting the course, students should complete a Python refresher if needed, covering:

- Python Basics (data structures, loops, functions);
- Jupyter Notebooks for coding and documentation;
- Basic Pandas & NumPy (handling data and numerical operations);
- Fundamentals of Data Cleaning.

Students who need a crash course in Python can complete an introductory self-paced module. The course is divided into four phases, each having two separate modules.

Table 1
Project-Based Learning Path for Data Visualization with Python

Phase	**Module**	**Topics and Mini-Projects**
	Phase 1: Fundamentals & First Project	
Introduction to Data Visualization	Why Data Visualization?	Importance of visualization, types of visualizations. **Mini-Project:** Extract insights from a messy dataset.
Setting Up the Environment	Python, Jupyter, Matplotlib, Pandas	Working with Jupyter, setting up visualization libraries. **Mini-Project:** Analyzing sales trends.
	Phase 2: Statistical Visualizations & Storytelling	
Seaborn for Statistical Visualization	Distributions, KDE, Boxplots	Introduction to Seaborn, visualizing distributions. **Mini-Project:** Exploring population demographics.
Categorical Data	Bar plots, count plots	Visualizing categorical data and comparisons. **Mini-Project:** Analyzing customer segmentation.
	Phase 3: Advanced & Interactive Visualizations	
Interactive Visualization with Plotly	Interactive line, scatter, bar charts	Plotly for interactive data exploration. **Mini-Project:** Social media trends dashboard.
Geospatial Data Visualization	Folium, Plotly Mapbox, Choropleth maps	Creating interactive geospatial visualizations. **Mini-Project:** COVID-19 spread analysis.
	Phase 4: Dashboards & Final Project	
Time-Series Analysis	Time-series plots, moving averages	Analyzing trends with time-series data. **Mini-Project:** Stock market insights dashboard.
Dashboard Development	Dash, Streamlit, combining multiple plots	Creating dashboards with interactivity. **Mini-Project:** Business intelligence dashboard.
Capstone Project	Final Data Visualization Report	Full project using a real-world dataset. **Capstone:** Create a full visualization project (sports analytics, climate change, e-commerce trends).

5. Conclusion and future work

This paper presents a novel approach to teaching data visualization in Python through an interactive learning platform that integrates automated grading and real-time feedback within Jupyter Notebooks. Unlike traditional programming courses that rely solely on static assignments, our system dynamically evaluates both code correctness and visualization quality, ensuring that students not only learn how to generate plots but also understand the principles of effective data communication. By leveraging structured problem-solving tasks, test-driven evaluations, and an automated feedback loop, this system enhances the learning experience, reduces instructor workload, and promotes self-guided learning. The expected outcome is a more engaging and scalable educational framework where students progressively build proficiency in Python visualization libraries, such as Matplotlib, Seaborn, and Plotly, while receiving instant formative assessment.

Future research can expand upon this framework by integrating AI-driven adaptive learning to personalize feedback based on individual student performance. Additionally, incorporating natural language processing (NLP) techniques could allow the system to interpret student queries and provide context-aware explanations for errors. Another direction involves real-world dataset integration, enabling students to work with live data streams for real-time visualization projects. Further advancements could also explore cross-platform compatibility, allowing the system to be implemented in cloud-based environments and hybrid learning settings, making data visualization education even more accessible and effective.

Acknowledgment

This paper was supported in part by the Blockchain Technology Laboratory at Belgrade Metropolitan University, Belgrade, Serbia and in part by the Ministry of Education, Science and Technological Development, Republic of Serbia ref. no. 451-03-47/2023-01/200029.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] L. M. Hudiburgh, D. Garbinsky, Data visualization: Bringing data to life in an introductory statistics course, Journal of Statistics Education 28 (2020) 262–279.

[2] D. Nolan, J. Perrett, Teaching and learning data visualization: Ideas and assignments, The American Statistician 70 (2016) 260–269.

[3] S. Nestorov, N. Jukić, S. Rossi, Design and implementation of a data visualization course with a real-world project component in an undergraduate information systems curriculum, Journal of Information Systems Education 30 (2019) 202.

[4] B. Bach, M. Keck, F. Rajabiyazdi, T. Losev, I. Meirelles, J. Dykes, R. S. Laramee, M. AlKadi, C. Stoiber, S. Huron, et al., Challenges and opportunities in data visualization education: A call to action, IEEE Transactions on visualization and computer graphics (2023).

[5] D. Embarak, Karkal, Data Analysis and Visualization Using Python, Springer, 2018. URL: https://link.springer.com/content/pdf/10.1007/978-1-4842-4109-7.pdf.

[6] J. Rogel-Salazar, Statistics and Data Visualisation with Python, Taylor and Francis, 2023. URL: https://www.taylorfrancis.com/books/mono/10.1201/9781003160359/statistics-data-visualisation-python-jesus-rogel-salazar.

[7] C. Ozgur, T. Colliau, G. Rogers, Z. Hughes, Matlab vs. python vs. r, Journal of Technology and Education (2017). URL: https://www.airitilibrary.com/Article/Detail/16838602-201707-201711160005-201711160005-355-371.

[8] S. Fahad, A. Yahya, Big data visualization: allotting by r and python with gui tools, in: IEEE International Conference on Big Data, 2018. URL: https://ieeexplore.ieee.org/abstract/document/8538413/.

[9] A. Navlani, A. Fandango, I. Idris, Python Data Analysis: Perform data collection, data processing, wrangling, visualization, and model building using Python, Packt Publishing, 2021. URL: https://books.google.com/books?hl=en&lr=&id=DN4SEAAAQBAJ&oi=fnd&pg=PP1&dq=importance+of+learning+data+visualization+and+role+of+python+compared+to+r+and+matlab&ots=P4xcaD_a-g&sig=OB9IRDJev6OWkFdjrxqie2nREY4.

[10] X. Kui, W. Liu, J. Xia, H. Du, Research on the improvement of python language programming course teaching methods based on visualization, in: 2017 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE), 2017. URL: https://ieeexplore.ieee.org/abstract/document/8085571/.

[11] J. Helminen, L. Malmi, Jype-a program visualization and programming exercise tool for python, in: Proceedings of the 10th Koli Calling International Conference on Computing Education Research, 2010. URL: https://dl.acm.org/doi/abs/10.1145/1879211.1879234. doi:10.1145/1879211.1879234.

[12] M. Mladenović, v. Žanko, The impact of using program visualization techniques on learning basic programming concepts at the k–12 level, Computer Applications in Engineering Education (2021). URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cae.22315. doi:10.1002/cae.22315.

[13] S. Dol, P. Jawandhiya, Data visualization for the dataset collected from education sector using python, in: 2024 IEEE International Conference on Data Science and Applications (DSA), 2024. URL: https://ieeexplore.ieee.org/abstract/document/10593329/.

[14] L. Lo, Y. Ming, H. Qu, Learning vis tools: Teaching data visualization tutorials, IEEE Transactions on Visualization and Computer Graphics (2019). URL: https://ieeexplore.ieee.org/abstract/document/8933751/.

[15] O. Karnalim, M. Ayub, The effectiveness of a program visualization tool on introductory programming: A case study with pythontutor, Journal of Computer Science Education (2017). URL: https://journal.binus.ac.id/index.php/commit/article/view/3704. doi:10.21512/commit.v11i2.3704.

[16] C. Douce, D. Livingstone, J. Orwell, Automatic test-based assessment of programming: A review, Journal on Educational Resources in Computing (JERIC) 5 (2005) 4–es.

[17] J. C. Caiza, J. M. Del Alamo, Programming assignments automatic grading: review of tools and implementations, INTED2013 Proceedings (2013) 5691–5700.

[18] N. Dimitrijević, N. Zdravković, D. Cvijanović, An overview of developed automatic grading teaching tools for learning different programming languages, in: IAI Academic Conference Proceedings, 2021, pp. 20–24.

[19] N. Zdravković, N. Dimitrijević, D. Cvijanović, A system for interactive learning of the python programming language with autograding support, in: E-Learning 2021 : proceedings / The Twelfth Internacional Conference on E-Learning, Belgrade, 23-24 September 2021., 2021, pp. 131–136.

[20] A. Lavanya, L. Gaurav, S. Sindhuja, H. Seam, Assessing the performance of python data visualization libraries: a review, ResearchGate (2023). URL: https://www.researchgate.net/publication/369533034_Assessing_the_Performance_of_Python_Data_Visualization_Libraries_A_Review.

[21] A. Yim, C. Chung, A. Yu, Matplotlib for Python Developers: Effective techniques for data visualization with Python, Packt Publishing, 2018. URL: https://books.google.com/books?id=G99YDwAAQBAJ.

[22] M. Kabir, Python for data analytics: A systematic literature review of tools, techniques, and applications, ResearchGate (2024). URL: https://www.researchgate.net/publication/385805476_Python_For_Data_Analytics_A_Systematic_Literature_Review_Of_Tools_Techniques_And_Applications.

[23] P. Gupta, A. Bagchi, Data Visualization with Python, Springer, 2024. URL: https://link.springer.com/chapter/10.1007/978-3-031-43725-0_7.

[24] A. Achanta, R. Boina, Advanced techniques in python for effective data visualization, ResearchGate (2024). URL: https://www.researchgate.net/publication/377331587_Advanced_Techniques_in_Python_for_Effective_Data_Visualization.

[25] A. Belorkar, S. Guntuku, S. Hora, A. Kumar, Interactive Data Visualization with Python: Present your data as an effective and compelling story, Packt Publishing, 2020. URL: https://books.google.com/books?id=_1PdDwAAQBAJ.

[26] J. Sundaram, K. Gowri, S. Devaraju, An exploration of python libraries in machine learning models for data science, in: ResearchGate, 2023. URL: https://www.researchgate.net/publication/373919503_An_Exploration_of_Python_Libraries_in_Machine_Learning_Models_for_Data_Science.

[27] K. Pandey, R. Panchal, A study of real-world data visualization of covid-19 dataset using python, International Journal of Modern Health (2020). URL: https://www.ijmh.org/wp-content/uploads/papers/v4i8/H0834044820.pdf.

[28] D. Embarak, Karkal, Data Analysis and Visualization Using Python, Springer, 2018. URL: https://www.academia.edu/download/103702444/Data_Analysis_and_Visualization_Using_Py.pdf.

[29] N. Dimitrijević, V. Milićević, N. Zdravković, D. Cvijanović, Learning the kotlin programming language using an autograding system, in: E-Learning 2021 : proceedings / The Twelfth Internacional Conference on E-Learning, Belgrade, 23-24

September 2021., 2021, pp. 137–141. URL: https://www.metropolitan.ac.rs/files/2021/10/Proceedings-12th-International-Conference-on-e-Learning-2021.pdf.

[30] N. Dimitrijevic, N. Zdravkovic, V. Milicevic, An automated grading framework for the mobile development programming language kotlin, International Journal for Quality Research 17 (2023) 313–324. URL: http://ijqr.net/paper.php?id=1071. doi:10.24874/ijqr17.02-01.

[31] E. Kisić, M. R. Milić, N. Zdravković, M. A. Conde, The effects of implementing project-based learning in the programming course, in: eLearning 2023 : The Fourteenth International Conference on eLearning 2023 / Proceedings 14th International Conference on eLearning (eLearning 2023), Belgrade, Serbia, September 28-29, 2023., 2024, pp. 55–66.