DOI: 10.15514/ISPRAS-2025-37(4)-26

Integrating an Ontology-Driven Approach to Data Visualization and AI-Based Visualization with Plotly

A.D. Dzheiranian, ORCID*:* 0009-0000-8916-2855 <*ADDzheyranyan@edu.hse.ru*>
L.N. Lyadova, ORCID: 0000-0001-5643-747X <*LNLyadova@gmail.com*>
HSE University,
38, Studencheskaia St., Perm, 614070, Russia

Abstract. This study introduces an AI-driven assistant prototype that automates the generation of data visualization scripts from natural language queries, eliminating the need for users to have programming skills. The article examines research aimed at developing tools for effective data visualization, compares data visualization systems based on the use of artificial intelligence, and shows the limitations of the existing tools. The proposed approach to data visualization is based on integrating knowledge-driven DSM platform (language toolkits) and generative AI tools. The proposed methodology categorizes tasks of data visualization into two distinct types: standard and non-standard. Standard tasks are solved with a code-generation approach based on prompts within a visual environment. Non-standard tasks are handled by extending existing libraries with user-defined packages. The language-oriented approach with DSM tools effectively unifies both categories: for standard tasks, users work with pre-existing DSLs and adjust parameters as necessary, whereas for non-standard tasks, users develop new DSLs with language toolkits automating visual DSL creation and code generation. The core of the language toolkits is multifaceted ontology. By integrating a large language model (LLM) with a knowledge-driven framework and a multifaceted ontology, the system enables dynamic, context-aware visualization workflows that ensure semantic traceability and reproducibility. The ontology not only stores descriptions of data visualization tasks but also facilitates the reuse of generated scripts, thereby enhancing the system's adaptability and fostering collaborative analytical work among user communities. The dataset, containing entries and variables encompassing different domains, is used to demonstrate the functionality of the prototype. The article provides examples of developing several visualization options, demonstrating the application of the proposed approach. Case studies demonstrate the prototype's efficacy in creating histograms, scatter plots, and other visualization methods, while reducing technical barriers for users. Future work will extend the assistant's functionality by incorporating user-defined visualization packages and additional LLM training to address non-standard tasks and complex visualization scenarios.

Keywords: data visualization; artificial intelligence; domain specific modeling; language toolkits; ontology; Python; Dash; Plotly.

For citation: Dzheiranian A.D., Lyadova L.N. Integrating an Ontology-Driven Approach to Data Visualization and AI-Based Visualization with Plotly. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 4, part 2, 2025, pp. 191–206. DOI: 10.15514/ISPRAS-2025-37(4)-26.

Интеграция подхода к визуализации данных на основе онтологии и визуализации на основе ИИ с использованием Plotly

А.Д. Джейранян, ORCID: 0009-0000-8916-2855 <*ADDzheyranyan@edu.hse.ru*>
Л.Н. Лядова, ORCID: 0000-0001-5643-747X <*LNLyadova@gmail.com*>

Национальный исследовательский университет «Высшая школа экономики», Россия, 614070, г. Пермь, ул. Студенческая, д. 38.

Аннотация. В данном исследовании представлен прототип ассистента, управляемого искусственным интеллектом, который автоматизирует генерацию скриптов визуализации данных на основе запросов на естественном языке, устраняя необходимость пользователей владеть навыками программирования. В статье рассматриваются исследования, направленные на разработку средств для эффективной визуализации данных, проводится сравнение систем визуализации данных, основанных на использовании искусственного интеллекта, показаны ограничения существующих средств. Предлагаемый подход к визуализации данных основан на интеграции DSM-платформы (языкового инструментария), управляемого знаниями, и инструментов генеративного искусственного интеллекта. Предлагаемая методология разделяет задачи визуализации данных на два типа: стандартные и нестандартные. Стандартные задачи решаются с помощью генерации кода на основе подсказок в визуальной среде. Нестандартные задачи обрабатываются путем расширения существующих библиотек с помощью определяемых пользователем пакетов. Языково-ориентированный подход с использованием средств DSM эффективно объединяет обе категории: для стандартных задач пользователи работают с существующими DSL и настраивают параметры по мере необходимости, тогда как для нестандартных задач пользователи разрабатывают новые DSL с помощью языкового инструментария, автоматизирующего создание визуальных DSL и генерацию кода. Ядром языкового инструментария является многоаспектная онтология. Благодаря интеграции большой языковой модели (LLM) с фреймворком, управляемым знаниями, и многоаспектной онтологией, система обеспечивает динамические, контекстно-ориентированные рабочие процессы визуализации, которые гарантируют семантическую прослеживаемость и воспроизводимость. Онтология не только хранит описания задач визуализации данных, но и облегчает повторное использование сгенерированных скриптов, повышая тем самым адаптивность системы и способствуя совместной аналитической работе сообществ пользователей. Набор данных, содержащий записи и переменные, охватывающие различные предметные области, используется для демонстрации функциональности прототипа. В статье приведены примеры разработки нескольких вариантов визуализаций, демонстрирующие применение предлагаемого подхода. Приведённые примеры демонстрируют эффективность прототипа в создании гистограмм, точечных диаграмм и других методов визуализации и снижении технических барьеров для пользователей. В перспективе функциональность ассистента будет расширена за счёт поддержки пользовательских пакетов визуализации и дополнительного обучения LLM для решения нестандартных задач и сложных сценариев визуализации.

Ключевые слова: визуализация данных; искусственный интеллект; предметно-ориентированное моделирование; языковой инструментарий; онтология; Python; Dash; Plotly.

Для цитирования: Джейранян А.Д., Лядова Л.Н. Интеграция подхода к визуализации данных на основе онтологии и визуализации на основе ИИ с использованием Plotly. Труды ИСП РАН, том 37, вып. 4, 2025 г., стр. 191–206 (на английском языке). DOI: 10.15514/ISPRAS-2025-37(4)-26.

1. Introduction

The field of data visualization has gained significant academic attention, as evidenced by a 77% increase in related publications in the Scopus database, rising from 3 232 in 2015 to 5 724 in 2024.

In data-driven decision-making, visualization tools play a crucial role in interpreting complex datasets, identifying patterns, and supporting insightful decision processes. These tools are used by a wide range of professionals, including data analysts, scientists, business executives, managers, educators, students, and other stakeholders.

Despite the growing reliance on data visualization, many users encounter substantial challenges in generating effective and accurate visual representations. A major concern is the proliferation of low-quality or misleading visualizations that either fail to convey meaningful insights or even distort the underlying data 0. This issue is widely discussed in communities such as "Data is Ugly" (www.reddit.com/r/dataisugly/), where enthusiasts and experts alike showcase and critique poor visualization practices. To address these challenges, the integration of user-defined visualization specifications is essential 0. These specifications enable users to customize visualizations according to the specific requirements of their tasks and domains, ensuring a more effective and informative presentation of data.

Current solutions – including spreadsheets, business intelligence (BI) platforms, programming languages with libraries for data visualization, domain specific languages (DSLs) for customizing visualization models, and approaches leveraging LLMs – offer varied levels of customization and implementation of specifications. However, in most cases, customization is limited to basic parameter adjustments, which may not always be sufficient. Some scenarios require more advanced modifications, such as the ability to create custom chart types. For example, designing a basketball specific scatter plot, where the plotting area mimics the layout of a basketball court rather than a basic rectangle, etc. 0.

Many researchers also highlight the inherent complexity of creating high-quality visualizations. As noted in 0, traditional visualization methods often require high programming expertise for customizing models, which poses significant challenges for users. Similarly, 0 emphasizes that designing effective visualizations remains a time-consuming and complex task, even for experienced professionals.

Thus, there is a growing need for a novel approach that enables the customization of diagrams to align with domain specific requirements and task objectives – without the necessity for advanced programming skills.

In previous studies, a promising data visualization methodology that categorizes visualization tasks into standard and non-standard ones was proposed. The platform architecture adheres to knowledge-driven principles, with a multifaceted ontology serving as the central repository for expert knowledge [5-7]. One method for addressing data visualization tasks within this methodology involves the generation of scripts in a programming language, thereby simplifying the workflow for non-technical users when developing and customizing diagrams.

Building upon this foundation, the present research aims to design and implement a prototype AI-assistant for the automated generation of visualization scripts, integrated with knowledge-driven architecture. The tasks of the study include:

1) an analysis of existing tools for automated visualization generation;
2) the architectural design of the AI-assistant prototype;
3) the prototype implementation, including the development of the subontology;
4) the demonstration of the prototype's functionality using a representative dataset.

2. Related Works

Programming languages offer extensive capabilities for creating effective and customizable visual representations. The most widely used languages in this field are Python, R, and JavaScript, each providing a rich ecosystem of visualization libraries 0. Maximum data visualization customization is provided for users, but coding skills are required. Specialized libraries reduce this barrier, though a basic understanding of programming remains essential. Recent advances in artificial intelligence have begun bridging this gap by automating data visualization workflows [9-11], particularly through natural language interfaces and script generation.

Modern AI-driven systems leverage large language models like GPT-3 and ChatGPT to interpret user queries, generate visualization scripts, and refine outputs iteratively. For instance, Chat2VIS

demonstrates how prompt engineering can produce context-aware visualizations while maintaining data security 0. Similarly, the platform described in 0 integrates NLP techniques and generative AI with Pandas to automate Python script generation for CSV-based datasets. ChatVis 0 is an iterative assistant built atop GPT-4 that synthesizes Python scripts for scientific visualization. Using an error-feedback loop, ChatVis captures runtime exceptions from each script execution and resubmits them to the model for correction, repeating until the code runs successfully. Data Formulator 2 0 combines a drag-and-drop UI with natural-language instructions to define new data fields and desired charts. It compiles UI selections, dialogue context, and NL directives into an LLM prompt, then generates Python code for both data transformation and visualization in an interactive, iterative session.

However, current AI-driven visualization systems exhibit critical limitations (Table 1).

Table 1. Limitations of current AI-driven visualization systems

Limitation	Description
Model "hallucinations"	LLMs may produce syntactically invalid code, or misinterpret parameters, undermining trust and necessitating expert oversight of generated scripts [12, 13]
Poor handling of ambiguity	When faced with vague or multi-interpretation queries, systems often generate overly generic or incomplete visualizations, failing to capture the full nuance of user intent in complex analytical scenarios 0
Restricted chart types	The use of standard libraries (for example, Matplotlib, Pandas) prevents the creation of non-standard visual forms, which forces users to refine low-level code manually
Limited creative design support	Customization options rarely go beyond chart type, color palette, and basic annotations; detailed control over layout, styling, and individual visual elements remain largely unsupported
Lack of domain specific expertise	Open models trained on broad web data struggle with specialized terminology and industry standards, leading to misinterpretation of subject-area concepts 0

While current LLMs cannot fully replace human expertise and still require robust verification and fine-tuning frameworks, they hold significant potential for accelerating code generation in data-analysis and visualization workflows.

This research develops and integrates a natural language script generation method into a broader knowledge-driven data visualization platform. The AI-assistant allows users to describe their charting requirements in natural language, automatically generates and runs the corresponding Python scripts, and uses a multifaceted ontology to audit user queries and facilitate the reuse of generated code across similar scenarios. The platform's library extension subsystem is designed to further train the model on custom visualization packages, empowering it to tackle domain specific, non-standard plotting tasks.

3. Data Visualization Technique Based on Knowledge and Generative AI

3.1 Proposed Approach to Data Visualization

The proposed methodology categorizes tasks of data visualization into two distinct types: standard and non-standard. For *standard tasks*, a code-generation approach based on prompts within a visual environment is employed, which facilitates the automatic creation and execution of scripts. In contrast, *non-standard tasks* are handled by extending existing libraries with user-defined packages. The *language-oriented approach* effectively unifies both categories: for standard tasks, users work with pre-existing DSLs and adjust parameters as necessary, whereas for non-standard tasks, users develop new DSLs. This development process is streamlined through the automated mapping of the domain ontology onto the metamodel of a base language, as described in [5-7].

The system architecture design follows a *knowledge-driven approach*, where the core of the platform is a *multifaceted ontology*. This ontology stores essential knowledge across the platform and is organized into six primary groups:

1. *Data sources ontology* includes information on the structure, types of data, etc.
2. *Domain ontology* contains the subjective knowledge of the expert community: concepts of the subject area, their relationships, and limitations (including rules).
3. *DSM-knowledge ontology* encompasses models, visualization languages, subject domains, rules for the transformation and generation of DSLs, etc.
4. *Scripts ontology* captures information regarding prompts, generation outcomes, and relevant metadata.
5. *Packages ontology* contains detailed descriptions of user-defined visualization packages.
6. *User task repository* is an ontology of user tasks and applied methods. Using this ontology facilitates search and reusable deployment of established models.

The system's overall architecture is shown in Fig. 1.

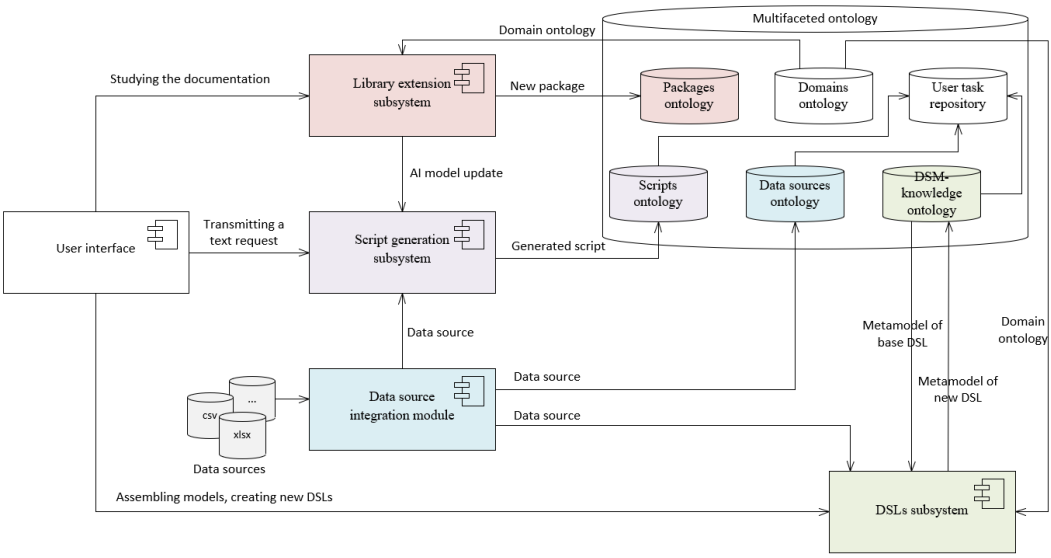

Fig. 1. Generalized structure of the data visualization tools prototype

In addition to the ontology, the system consists of the following components:

1. *User interface* is implemented as a single-entry point: the UI unifies three isolated, specialized interfaces (visualization library management, script generation, and DSM-platform interaction) behind a Facade pattern that provides seamless navigation between sections without exposing their internal logic.
2. *Library extension subsystem* is responsible for creating new packages atop existing visualization libraries (e.g., Matplotlib, Seaborn, D3, etc.) and for training the language model to understand and leverage these extended libraries.
3. *Script generation subsystem* automatically generates and executes visualization scripts based on user input.
4. *DSLs subsystem* constitutes the DSM-platform (language toolkit), manages both the utilization of preexisting DSLs and the creation of new ones through an automated mapping of the domain ontology onto the metamodel of a base language.

5. *Data source integration module* provides connections to CSV and XLSX data stores. It uses the *Data sources ontology* to automatically save metadata about any dataset uploaded to the system.

3.2 AI-assistant Architectural Layers and Component Responsibilities

The AI-assistant is presented as a *Script generation subsystem*, that integrates semantic technologies with LLMs to automate visualization code generation, while maintaining an ontology-grounded history of user interactions. The system adheres to layered architecture (Fig. 2), decoupling *presentation*, *business logic*, and *data management* concerns to ensure extensibility and maintainability.

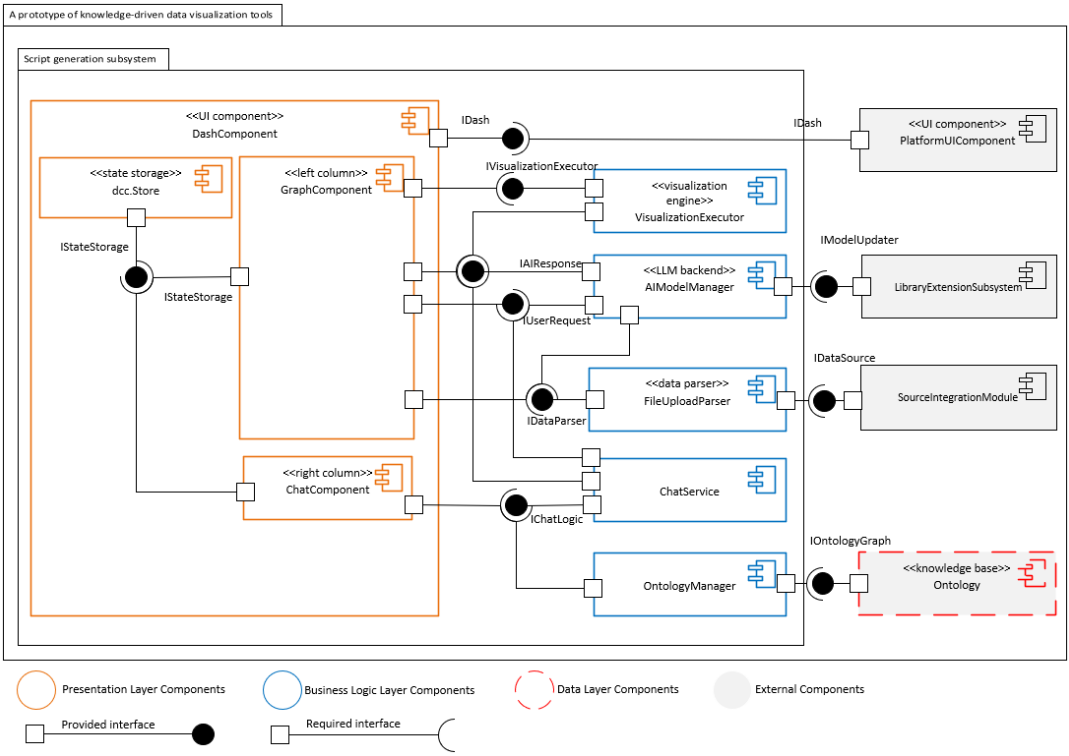

Fig. 2. Script generation subsystem (AI-assistant) architecture

The *presentation layer* follows a reactive programming model, with Dash callbacks mediating between user actions and backend services. The *DashComponent* serves as the root container, implementing the user interface via the Dash framework. It orchestrates three subcomponents:

1. *ChatComponent* manages real-time chat interactions through the *IChatLogic* interface, rendering message history and propagating user inputs.
2. *GraphComponent* handles visualization outputs by coordinating dependencies through three critical interfaces: *IDataParser* – for structured data ingestion; *IAIResponse* – for displaying the generated LLM response; *IVisualizationExecutor* – for dynamic code execution. It also implements the *IUserRequest* interface to expose user-entered queries.
3. *dcc.Store* acts as an in-browser, client-side state container. It provides the *IStateStorage* interface for any UI component to read or write transient JSON data (e.g., current query, graph parameters, or intermediate results). This component eliminates unnecessary server roundtrips by keeping frequently accessed state in the browser.

Five components of the *business logic layer* implement domain specific processing:

1. *FileUploadParser (IDataParser)* implements CSV/Excel parsing via Pandas, extracting columnar data.
2. *ChatService (IChatLogic)* manages chat session state, enforcing conversation context persistence.
3. *AIModelManager (IAIResponse)* utilizes LangChain pipelines with the ChatGroq API (Llama3-70B) to generate Plotly visualization code, conditioned by uploaded data (*IDataParser*) and user request (*IUserRequest*).
4. *VisualizationExecutor (IVisualizationExecutor)* safely executes generated Python code in sandboxed environments.
5. *OntologyManager* maintains the RDFlib knowledge graph, implementing ontology operations (prompt/script entity creation, model versioning, etc.) through *IOntologyGraph*.

The *Ontology* component of the *data layer*, exposed through the *IOntologyGraph* interface, acts as the system's persistent knowledge base.

The *Script generation subsyste*m interacts with *external components* (third-party prototype components):

1. The *PlatformUIComponent* is the core user interface of the platform. It integrates the DashComponent to display visualization and chat interactions via linked interface *IDash*.
2. The *LibraryExtensionSubsystem* is responsible for developing and publishing new visualization packages. It implements the *IModelUpdater* interface, through which the *AIModelManager* component initiates an update or selection of the LLM model version. When preparing a query to a language model, the *AIModelManager* switches the context to the latest trained model via *IModelUpdater* if necessary.
3. When a new file is uploaded to the system, the *SourceIntegrationModule* passes it to the *FileUploadParser* component via the *IDataSource* interface for further processing.

3.3 User-System Interaction Workflow

The *data visualization assistant* operates through a multistage process (Fig. 3), which is initiated once a user uploads a data file. This workflow can be summarized as follows:

1. *File upload and data parsing*. After the user uploads a data file (CSV or Excel), the system validates the file to confirm compatibility with supported data types. Then it converts the contents into a structured representation suitable for subsequent processing and displays an interactive preview, allowing users to inspect and verify the data before proceeding.
2. *User request processing*. When a visualization request is submitted, the application constructs a context-aware prompt to provide the system with sufficient details for generating suitable visualization. This information is passed on to the generative model, which produces an executable script tailored to the user's stated objective.
3. *Code execution and visualization*. The generated script undergoes a basic sanitization process to ensure compliance with the application's requirements. Subsequently, it is executed in a contained environment. The visualization output is then rendered within the application interface, enabling users to explore the data interactively. Errors or inconsistencies within the script are handled gracefully, with feedback provided to the user as needed.
4. *Semantic metadata persistence*. All pertinent data from each session (including user prompts, the resulting scripts, and associated metadata) are stored in a semantic repository.
5. *Chat history and interface updates*. The human–AI interactions are appended to a session-specific chat history, formatted with distinct styling for human–AI messages. The interface dynamically refreshes to display both the visualization and code snippet, ensuring transparency in the AI's output.

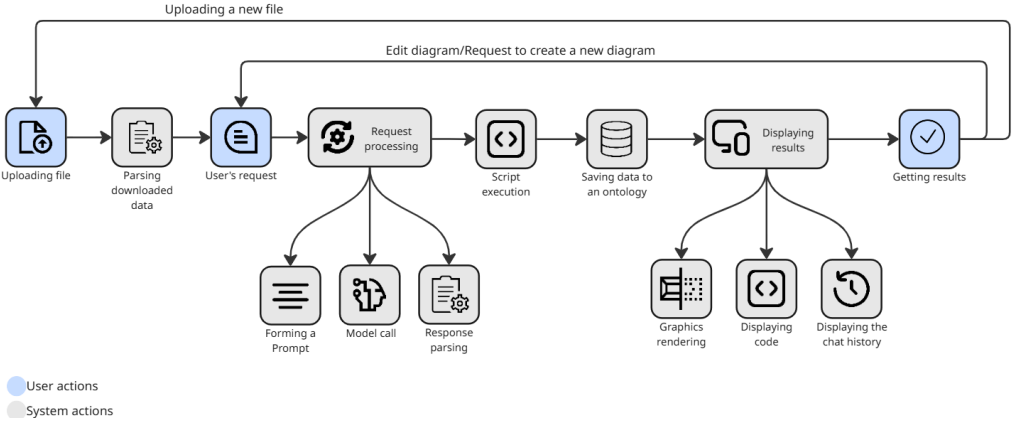

Fig. 3. User-system interaction workflow

This workflow emphasizes automation, reproducibility, and semantic traceability by integrating interactive visualization with knowledge graph technologies. The ontology facilitates long-term analytics on AI-generated artifacts, thereby adhering to FAIR (Findable, Accessible, Interoperable, and Reusable) data principles 0.

4. Prototype Development

4.1 Selection of Development Tools

Programming language Python with *Plotly* visualization library was selected for its versatility and extensive support across data analytics and visualization domains. Plotly, in particular, was chosen due to its capabilities for interactive visualization and its comprehensive suite of supported chart types.

The *ontological framework* was developed using *Protégé ontology editor*. This framework enables the persistence of necessary data within an RDF-based ontology managed via the *Python library RDFlib.*

The user interface is implemented using *Dash* – a Python framework designed for building interactive web applications.

AI integration for natural language processing and prompt management is implemented with the *Groq API* in conjunction with the *llama3-70b-8192 model*. This configuration provides an 8192-token context window, supporting extensive natural language input without limitations on maximum completion tokens or file size.

The system leverages the *LangChain* framework for the *pipeline orchestration* to orchestrate the end-to-end process of generating Plotly-compatible Python code. LangChain facilitates prompt formulation and task chaining. It ensures that the model adheres to predefined constraints (such as omitting data-loading steps) and focuses solely on visualization logic (e.g., axis mapping and chart type selection).

4.2 Ontology Development

The structure of the ontology comprises three core classes: *Prompt*, *GeneratedScript*, and *Metadata*, interconnected via object properties to model provenance relationships (Table 2). Data properties are defined to capture temporal, contextual, and technical attributes. The relationships are structured as follows: *"Prompt → (producedScript) → GeneratedScript → (hasMetadata) → Metadata".* The ontology was serialized in RDF/XML to ensure interoperability with semantic web tools.

Table 2. Ontology entities

Class	Properties	
	Data properties	Object properties
Generated_scripts	hasCodeContent (string) hasLanguage (string) hasTimestamp (dateTime)	hasMetadata
Metadata	hasFileName (string) hasModelVersion (string)	–
Prompts	hasContent (string) hasSessionID (string) hasTimestamp (dateTime)	producedScript

4.3 Interface Design

The interface features a dual-panel layout (Fig. 4). Left Panel (*GraphComponent*) combines data upload functionality, visualization controls, and code display. Key components are:

- a drag-and-drop file uploader supporting CSV/Excel formats;
- an interactive grid (Dash AG Grid) for real-time data preview;
- a text area for natural language queries and a button to trigger AI processing;
- dynamic rendering of Plotly graphs and generated code snippets.

Right Panel (*ChatComponent*) displays a session-specific chat history with distinct styling for user inputs (blue background) and AI responses (grey background).

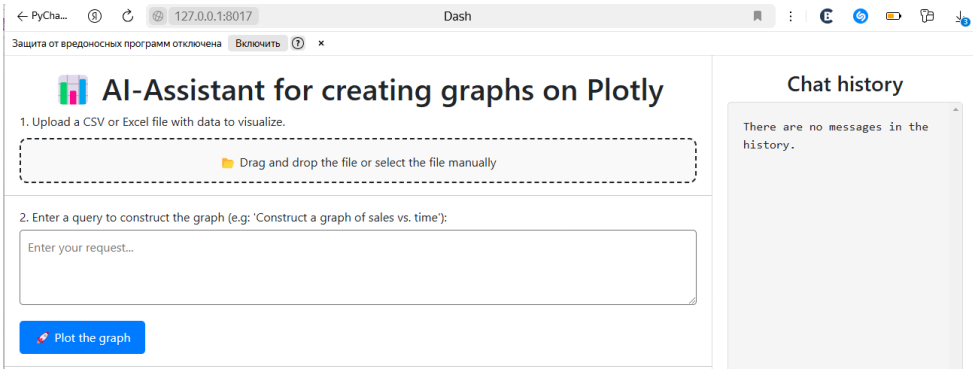

Fig. 4. AI-assistant interface

4.4 Prompt Formation

When a visualization request is submitted, the application constructs a context-aware prompt using LangChain's *ChatPromptTemplate*. The prompt integrates the following elements: the original filename; the first five rows of the dataset as a CSV string; historical chat messages (if available) to maintain conversational continuity; a user request.

After the prompt is generated, it is processed by the Groq API using the llama model.

4.5 Code Extraction and Visualization Execution

The *VisualizationExecutor* component orchestrates the transformation of AI-generated code into executable visualization workflows.

Regular expression patterns for the code transformation:

```
r'```(?:[Pp]ython)?(.*?)```'          (1)
r'df\s*=\s*pd\.read_\w+\(.*?\)'       (2)
r'(?m)^\s*fig\.show\(\)\s*$'          (3)
```

Upon receiving the model's raw output, a regular expression pattern (1) identifies and extracts Python code blocks. The sanitization phase removes non-essential operations, including: redundant data loading – predefined DataFrame declarations (pd.read_csv() / pd.read_excel()) are stripped (2), as the dataset is already cached client-side; rendering commands – calls to fig.show() are omitted (3) to prevent runtime conflicts within Dash's callback architecture.

The purified code is executed dynamically via Python's function *exec()* within a sandboxed environment.

4.6 Chat History Management

The *ChatService* component ensures conversational continuity by persisting user-AI interactions in a JSON-serializable format. Key features include the following functions:

- message typing: messages are classified as *HumanMessage* (user queries) or *AIMessage* (model outputs), each with timestamped metadata;
- state preservation: the *dcc.Store* component caches chat history client-side, enabling seamless navigation during active sessions;
- dynamic rendering: messages are displayed in chronological order with auto-scrolling to the latest entry.

4.7 Ontology-to-Prototype Integration

The integration of semantic knowledge representation with application logic is facilitated by a dedicated *OntologyManager* component. This component orchestrates the following processes:

1. Ontology Initialization:
   - the component loads an existing RDF/XML file or initializes a new RDF graph;
   - custom namespace (*"MY")* is bound to ensure Uniform Resource Identifier (URI) consistency across entities, adhering to semantic web standards.
2. Individuals Generation:
   - the *add_prompt(content, session_id)* method generates unique prompt individuals, assigning Universally Unique Identifiers (UUIDs), user provided content, and optional session identifiers to enable cross-request traceability;
   - the *add_generated_script(code_content, language = "Python")* function encapsulates generated code snippets, programming language specifications (default: Python), and timestamps to record the exact time of code synthesis;
   - the *add_metadata(file_name, model_version)* method associates scripts with technical metadata, including the source filename and AI model version, ensuring reproducibility.
3. Semantic Relationship Mapping:
   - RDF triples are dynamically constructed using object properties *producedScript* and *hasMetadata*;
   - the methods *link_promt_to_script (prompt_uri, script_uri)* and *link_script_to_metadata (script_uri, metadata_uri)* formalize these connections within the ontology, establishing a provenance chain from user requests to technical execution parameters.

5. A Case of Data Visualization Generation

5.1 Dataset Description

To demonstrate the functionality of the prototype, the *Sleep_health_and_lifestyle_dataset.csv* was used. This dataset contains 400 entries and 13 variables, encompassing demographic, behavioral, and physiological factors related to sleep health, such as age, gender, occupation, sleep duration and quality, physical activity, stress, BMI category, blood pressure, heart rate, daily steps, and the presence of sleep disorders.

5.2 Experiments with Generating Data Visualizations

After uploading the dataset, the system displays the data in a tabular format, allowing users to filter and sort columns (Fig. 5). At this initial stage, the message history is empty.

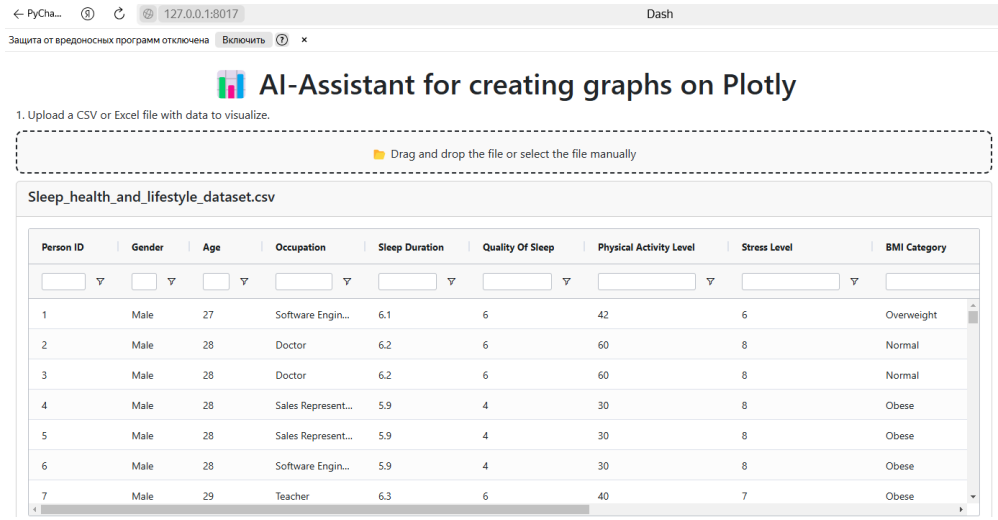

Fig. 5. Display of loaded data in the system

5.2.1 Step 1 “No Chart Type, Only Concept”

Task: *“Build a visualization of the top 10 professions with the highest stress level”.*

Response and commentary: the system autonomously selected an appropriate visualization type – a bar chart – to represent the data (Fig 6). The interactive graph displays tooltips upon hovering over a bar. Examination of the code (Fig. 7) reveals that the process involves grouping by occupation, computing the average stress level, and subsequently sorting the results to identify the top ten professions.

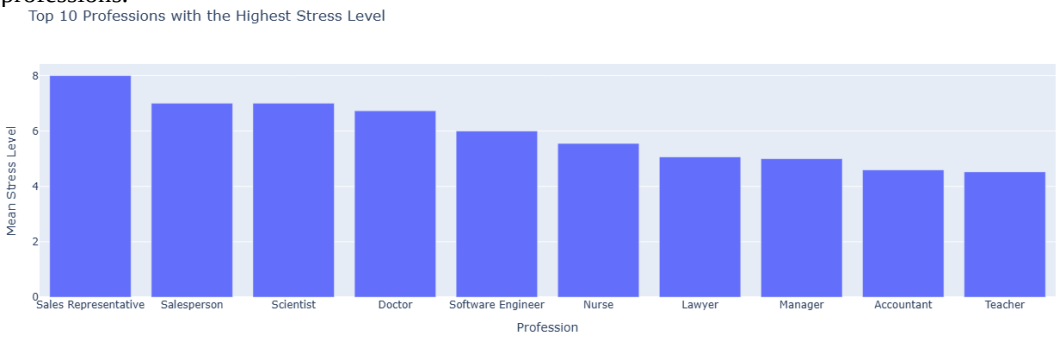

Fig. 6. Experiment №1 – Generated visualization

Here's the code to create a bar chart using Plotly to visualize the top 10 professions with the highest stress level:

```python
import pandas as pd
import plotly.express as px

# Load the data
df = pd.read_csv('Sleep_health_and_lifestyle_dataset.csv')

# Group by Occupation and calculate the mean Stress Level
stress_by_profession = df.groupby('Occupation')['Stress Level'].mean().reset_index()

# Sort the data by mean Stress Level in descending order
stress_by_profession = stress_by_profession.sort_values('Stress Level', ascending=False)

# Select the top 10 professions with the highest stress level
top_10_stressful_professions = stress_by_profession.head(10)

# Create a bar chart using Plotly
fig = px.bar(top_10_stressful_professions, x='Occupation', y='Stress Level',
             title='Top 10 Professions with the Highest Stress Level',
             labels={'Occupation': 'Profession', 'Stress Level': 'Mean Stress Level'})

fig.show()
```

This code will generate a bar chart showing the top 10 professions with the highest mean stress level. The x-axis will display the profession, and the y-axis will display the mean stress level.

Fig. 7. Experiment № 1 – Generated script

5.2.2 Step 2 “Customization of a Previously Created Chart”

Task: “*Change the color of bars to yellow, set the title to “Professions with the highest stress level”, and display only the top 5 professions*”.

Response and commentary: the model successfully addressed all requirements, dynamically updating the visualization (Fig. 8). The chat history is updated to document the changes and interactions (Fig 9).

5.2.3 Step 3 “Complex Task with No Specified Chart Type”

Task: *“Show me a visualization from which I can draw many conclusions”.*

Response and commentary: given the open-ended nature of the task, the model produced a scatter plot (Fig. 10). This visualization illustrates the relationship between Sleep Duration, Physical Activity Level, and Stress Level for each individual. The system further provides interpretative questions (e.g., “Do individuals with higher Physical Activity Level tend to have lower Stress Levels or higher Daily Steps?”, “Are there any clusters or patterns in the data that could indicate underlying relationships between variables?”), thereby facilitating comprehensive analytical inquiry.

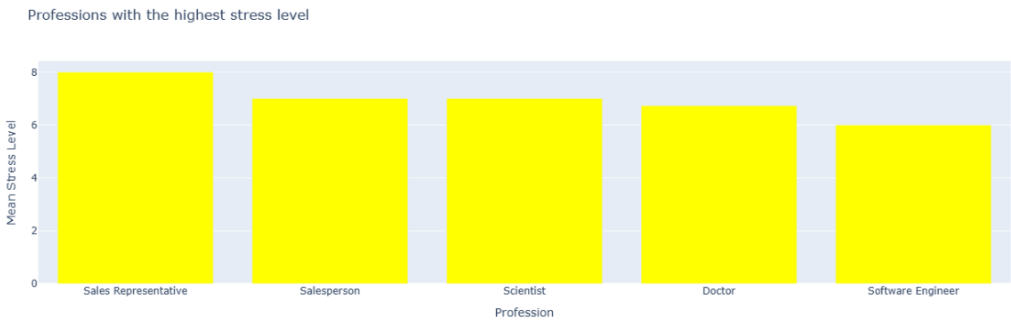

Fig. 8. Experiment № 2 – Generated visualization

stress level. The x-axis will display the profession, and the y-axis will display the mean stress level.

User: Change color of bars to yellow. Set title as "Professions with the highest stress level". Demonstrate only top-5 professions.

AI: Here's the updated code: ```python import pandas as pd import plotly.express as px # Load the data df = pd.read_csv('Sleep_health_and_lifestyle_dataset.csv') # Group by Occupation and calculate the mean Stress Level stress_by_profession = df.groupby('Occupation')['Stress Level'].mean().reset_index() # Sort the data by mean Stress Level in descending order stress_by_profession = stress_by_profession.sort_values('Stress Level', ascending=False) # Select the top 5 professions with the highest stress level top_5_stressful_professions = stress_by_profession.head(5) # Create a bar chart using Plotly fig = px.bar(top_5_stressful_professions, x='Occupation', y='Stress Level', title='Professions with the highest stress level', labels={'Occupation': 'Profession', 'Stress Level': 'Mean Stress Level'}, color_discrete_sequence=['yellow']) fig.show() ``` This code will generate a bar chart showing the top 5 professions with the highest mean stress level. The x-axis will display the profession, the y-axis will display the mean stress level, and the bars will be yellow.

Fig. 9. Experiment № 2 – Chat history

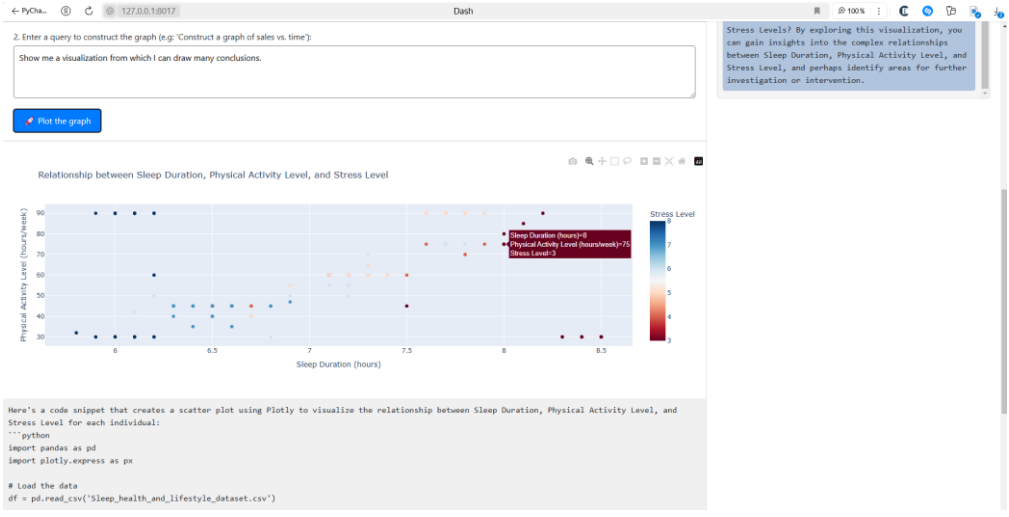

Fig. 10. Experiment № 3 – Generated visualization

5.2.4 Step 4 “Predefined Chart Type, No Specific Concept”

Task: *“Construct a pie chart”.*

Response and commentary: in response, the model generated a pie chart to display the distribution of BMI categories within the dataset (Fig. 11).

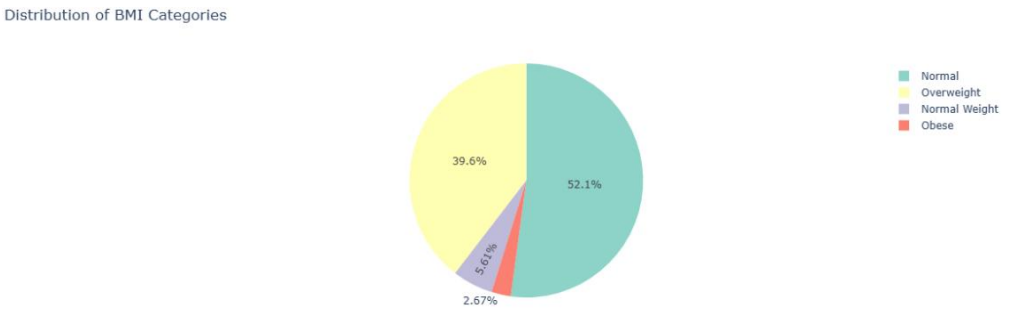

Fig. 11. Experiment № 4 – Generated visualization

5.3 Results of the Experiments: Summary

Collectively, these examples demonstrate a fraction of the visualization capabilities achievable with the AI-assistant. Moreover, an analysis of the ontology reveals that it has been dynamically updated throughout these interactions. The sequence of relationships “prompt – generated script – metadata” is preserved within the ontology individuals, confirming the robust integration of the knowledge-driven framework (Fig. 12).

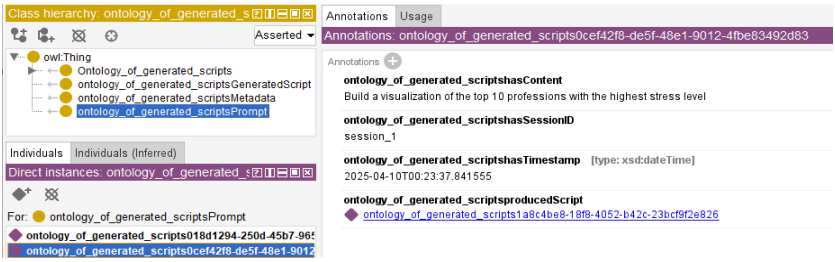

Fig. 12. An example of Prompt class individual

6. Evaluation of the AI-assistant

The AI-assistant automates the creation and execution of Plotly scripts by transforming user’s natural-language requests into ready-to-run code. During testing, it demonstrated strong performance across a variety of tasks: from drawing basic bar charts to interpreting more abstract specifications. It also simplifies choosing the most appropriate visualization for a given dataset or analysis goal. By displaying the generated code, the assistant serves as a teaching aid for both Python and Plotly.

The prototype’s architectural novelty lies in integration with two core components:

1. A *multifaceted ontology* that logs every user request and generated script, enabling scalable knowledge accumulation, reuse of code templates across similar scenarios, and enhanced collaborative analytics.
2. A *library extension subsystem* that develops custom plugins atop existing libraries, trains the model to invoke these domain-driven functions natively, and in future iterations will leverage an *AIModelManager* to automatically select or update to the latest fine-tuned LLM version.

However, generative models remain prone to "hallucinations", inventing nonexistent methods, producing syntactically incorrect code, or misinterpreting parameters, which can undermine script's reliability. To mitigate these risks, the interface displays a persistent footer reminder to verify the assistant's outputs and shows a clear *"Error when generating the graph"* notification if execution fails.

Currently, users are responsible for validating the generated code via these error notifications, because the system does not automatically check the accuracy of the model output against the user's intent. To address this issue, future work includes training the model on real data and implementing performance criteria for effective visualizations. Derived from cognitive-visualization research [17, 18], these metrics will be formalized and embedded in the toolkit to enable automated compliance checks of visualizations against best practices.

7. Conclusion

In conclusion, this study has designed and developed a data visualization assistant as a part of a prototype of knowledge-driven data visualization tools. The innovative aspect of this approach lies in the integration of an AI-assistant with an ontology-based framework, which facilitates scalable knowledge accumulation. This integration significantly enhances collaborative analytics by enabling the sharing of standardized scripts and domain specific insights among user communities, thereby reducing redundancy and fostering robust reproducibility in data-driven workflows.

Future work will first expand the Plotly ecosystem with new custom packages and then train the model on user-defined modules to better support atypical visualizations. Criteria for effective visualizations [17, 18] will also be formalized and embedded into the platform to automatically analyze the compliance of constructed diagrams with best practices and help users avoid common mistakes.

References

Sawicki J., Burdukiewicz M. VisQualdex: a Comprehensive Guide to Good Data Visualization. Scientific Visualization, 2023, vol. 15, no. 1, pp. 127–149. DOI: 10.26583/sv.15.1.11.

Qin X., Luo Y., Tang N., Li G. Making Data Visualization More Efficient and Effective: A Survey. The VLDB Journal, 2020, vol. 29, no. 1, pp. 93–117. DOI: 10.1007/s00778-019 00588-3.

Goldsberry K. Courtvision: New Visual and Spatial Analytics for the NBA. In Proc. of the 2012 MIT Sloan Sports Analytics Conference, 2012, pp. 12–15.

Singh A. Democratizing Data Visualization and Insights Extraction with Pandas, Generative AI, and CSV Data. International Journal of Scientific Research in Engineering and Management, 2024, vol. 8, no. 5, pp. 1–5. DOI: 10.55041/ijsrem33437.

Dzheiranian A. D., Ermakov I. D., Proskuryakov K. A., Lyadova L. N. Designing Data Visualization System Based on Language-Oriented Approach. Proceedings of the Institute for System Programming of the RAS, 2024, vol. 36, no. 2, pp. 127–140. DOI: 10.15514/ISPRAS-2024-36(2)-10.

Dzheiranian A. D., Ermakov I. D., Proskuryakov K. A., Lyadova L. N. Development of Data Visualization Tools Based on Domain Specific Modeling. In Proc. of the 34th International Conference on Computer Graphics and Machine Vision (GraphiCon 2024), 2024, pp. 300–314. DOI: 10.25206/978-5-8149-3873-2-2024 300-314.

Dzheiranian A. D., Ermakov I. D., Proskuryakov K. A., Lyadova L. N. An Approach to Developing Data Visualization Tools Based on Domain Specific Modeling. Scientific Visualization, 2024, vol. 16, no. 4, pp. 82–101. DOI: 10.26583/sv.16.4.08.

Shakeel H. M., Iram S., Al-Aqrabi H., Alsboui T., Hill R. A Comprehensive State-of-the-Art Survey on Data Visualization Tools: Research Developments, Challenges and Future Domain Specific Visualization Framework. IEEE Access, 2022, vol. 10, pp. 96581–96601. DOI: 10.1109/ACCESS.2022.3205115.

Shi D., Xu X., Sun F., Shi Y., Cao N. Calliope: Automatic Visual Data Story Generation from a Spreadsheet. IEEE Transactions on Visualization and Computer Graphics, 2021, vol. 27, no. 2, pp. 453–463. DOI: 10.1109/TVCG.2020.3030403.

Wang X., Wu Z., Huang W., Wei Y., Huang Z., Xu M., Chen W. VIS+AI: Integrating Visualization with Artificial Intelligence for Efficient Data Analysis. Frontiers of Computer Science, 2023, vol. 17, no. 6. DOI: 10.1007/s11704-023-2691-y.

Wu A., Wang Y., Shu X., Moritz D., Cui W., Zhang H., Zhang D., Qu H. AI4VIS: Survey on Artificial Intelligence Approaches for Data Visualization. IEEE Transactions on Visualization and Computer Graphics, 2021, vol. 28, no. 12, pp. 5049–5070. DOI: 10.1109/tvcg.2021.3099002.

Maddigan P., Susnjak T. Chat2VIS: Generating Data Visualisations via Natural Language using ChatGPT, Codex and GPT-3 Large Language Models. IEEE Access, 2023, vol. 11, pp. 45181–45193. DOI: 10.1109/access.2023.3274199.

Mallick T., Yildiz O., Lenz D., Peterka T. ChatVis: Automating Scientific Visualization with a Large Language Model. In Proc. of the SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2024, pp. 49–55. DOI: 10.1109/SCW63240.2024.00014.

Wang C., Lee B., Drucker S., Marshall D., Gao J. Data Formulator 2: Iteratively Creating Rich Visualizations with AI. arXiv preprint, 2024. arXiv: 2408.16119.

Hong J., Seto C., Fan A., Maciejewski R. Do LLMs Have Visualization Literacy? An Evaluation on Modified Visualizations to Test Generalization in Data Interpretation. IEEE Transactions on Visualization and Computer Graphics, 2025. DOI: 10.1109/TVCG.2025.3536358.

Alves T. FAIR Data: What It Is and How We Can Support Its Principles. Science Editor, 2024. DOI: 10.36591/se-4703-04.

Midway S. R. Principles of Effective Data Visualization. Patterns, 2020, vol. 1, no. 9. DOI: 10.1016/j.patter.2020.100141.

Midway S. R., Brum J. R., Robertson M. Show and Tell: Approaches for Effective Figures. Limnology and Oceanography Letters, 2023, vol. 8, no. 2, pp. 213–219. DOI: 10.1002/lol2.10288.

Информация об авторах / Information about authors

Анна Даниеловна ДЖЕЙРАНЯН – студент бакалавриата Национального исследовательского университета «Высшая школа экономики» (НИУ ВШЭ –Пермь), образовательная программа «Программная инженерия». Сфера научных интересов: анализ и визуализация данных, генеративные модели, предметно-ориентированное моделирование.

Anna Danielovna DZHEIRANIAN – undergraduate student at the National Research University – Higher School of Economics (HSE University, Perm Branch), educational program “Software Engineering”. Research interests: data analysis and visualization, generative models, domain specific modeling.

Людмила Николаевна ЛЯДОВА – кандидат физико-математических наук, доцент, доцент кафедры информационных технологий в бизнесе Национального исследовательского университета «Высшая школа экономики» (НИУ ВШЭ–Пермь). Сфера научных интересов: языки моделирования, предметно-ориентированное моделирование, языковые инструментарии, CASE-средства, системы имитационного моделирования.

Lyudmila Nikolaevna LYADOVA – Cand. Sci. (Phys.-Math.) in Computer Science, Associate Professor of the Department of Information Technologies in Business of the National Research University – Higher School of Economics (HSE University, Perm Branch). Research interests: modeling languages, domain specific modeling, language toolkits, CASE tools, simulation systems.