https://doi.org/10.7494/geom.2022.16.1.117

Michał Bednarczyk[1]

A Python Library for the Jupyteo IDE Earth Observation Processing Tool Enabling Interoperability with the QGIS System for Use in Data Science

Abstract: This paper describes JupyQgis – a new Python library for Jupyteo IDE enabling interoperability with the QGIS system. Jupyteo is an online integrated development environment for earth observation data processing and is available on a cloud platform. It is targeted at remote sensing experts, scientists and users who can develop the Jupyter notebook by reusing embedded open-source tools, WPS interfaces and existing notebooks. In recent years, there has been an increasing popularity of data science methods that have become the focus of many organizations. Many scientific disciplines are facing a significant transformation due to data-driven solutions. This is especially true of geodesy, environmental sciences, and Earth sciences, where large data sets, such as Earth observation satellite data (EO data) and GIS data are used. The previous experience in using Jupyteo, both among the users of this platform and its creators, indicates the need to supplement its functionality with GIS analytical tools. This study analyzed the most efficient way to combine the functionality of the QGIS system with the functionality of the Jupyteo platform in one tool. It was found that the most suitable solution is to create a custom library providing an API for collaboration between both environments. The resulting library makes the work much easier and simplifies the source code of the created Python scripts. The functionality of the developed solution was illustrated with a test use case.

Keywords: Earth observation data processing, IDE, IPython, Jupyter notebook, web processing service, GIS, data science, machine learning, API

Received: 2 September 2021; accepted: 27 October 2021

© 2022 Author. This is an open access publication, which can be used, distributed and reproduced in any medium according to the Creative Commons CC-BY 4.0 License.

[1] University of Warmia and Mazury in Olsztyn, Faculty of Geoengineering, Institute of Geodesy and Civil Engineering, Department of Geodesy, Olsztyn, Poland, email: michal.bednarczyk@uwm.edu.pl, https://orcid.org/0000-0002-0450-5327

1. Introduction

The functioning of the world is increasingly based on collecting and processing vast amounts of data. Tools and methods of data acquisition are constantly evolving, entering more and more spheres of our existence. Data are collected on everything, at every time and in every place. This causes each area of life to change gradually and today many things are done differently from the past. For example, in the case of scientific research, model-driven approaches have been supplemented with data-driven approaches [1, 2].

In recent years, there has been an increase in the popularity of data science methods in many organizations. Data science is already widely used in business to design successful strategies and policies. The economic sector is facing a significant transformation due to the penetration of data-driven innovation in the business core. A similar transformation is underway within many scientific disciplines [3, 4]. This is especially true of geodesy, environmental sciences and Earth sciences. These are disciplines that use large data sets, such as Earth observation satellite data (EO data) and GIS data. The market for this data is broad and diverse. Companies providing data develop or buy increasingly newer technologies and tools because the data processing techniques and tools used several years ago are no longer sufficient. It is also challenging to do without big data processing and storage techniques in this field [5]. Such a situation was predicted earlier by the scientific community [6]. Today, there has been a significant increase in the number and variety of new data science tools in response to the growing demand for the processing of increasingly larger data sets [7, 8].

Geoinformation derived from Earth observation satellite data is used in many scientific, governmental and planning tasks. These include, among others, geoscience, atmospheric sciences, cartography, resource management, civil security, disaster relief, as well as planning and decision support [9, 10]. Earth observation has irreversibly arrived in the big data era, among others, with the ESA's Sentinel satellites and with the blooming of so-called NewSpace companies, representing the market for private access to space and technologies related to this issue. This not only requires new technological approaches to manage and process large amounts of data but also new analysis methods such as machine learning, artificial intelligence and cluster analysis [11–14].

In 2019, the volume of only the open data produced by Landsat-7 and Landsat-8, MODIS (Terra and Aqua units) and the three first Sentinel missions (Sentinel-1, Sentinel-2 and Sentinel-3) was around 5 PB [15]. These big data sets often exceed the memory, storage and processing capacities of personal computers, imposing severe limits that lead users to take advantage of only a small portion of the available data for scientific research and operational application [16, 17]

The demand for new solutions is constantly increasing. Among the new platforms and tools created to store and process EO data in recent years, for example [18]

are Google Earth Engine (GEE), Sentinel Hub, Open Data Cube (ODC), System for Earth Observation Data Access, Processing and Analysis for Land Monitoring (SEPAL), OpenEO, JEODPP, pipsCloud and Jupyteo IDE.

When writing about data science today, it is hard not to refer to Python. For scientific computing, data science and machine learning it is the most preferred programming language. This is mainly because Python is relatively easy to learn. Its possibilities are very extensive, boosting both performance and productivity by enabling the use of low-level libraries and clean high-level APIs. Python is available on many open-source or free-access platforms, including Jupyter, Anaconda Individual Edition and Google Colab [19–23].

1.1. Jupyteo IDE

The variety of libraries available in Python, like Scikit-learn, Pandas (Python Data Analysis Library), NumPy, TensorFlow, Matplotlib, and PySpark, makes techniques such as machine learning or cluster analysis within reach of anyone who can program and is an expert in the given field and is open to new programming techniques. Although the functionality of existing libraries developed by others is often sufficient, sometimes specialized or custom-made tools are required.

Building new tools and platforms for data science very often consists of adapting and improving existing solutions. In such cases, Python and the solutions that use it are handy because most of them are open-source, making it possible to modify their source code. To build a new tool or platform, it is necessary to formulate the functional requirements of a planned solution. The software components must then be identified among existing products to cover most of the specified requirements. For example, Jupyter would be a good choice as the basis for a data science platform. Functionalities that cannot be achieved by adjusting ready-made elements should be programmed on one's own. In the case of a web platform, the most convenient way is to integrate everything into a single, scalable environment using Docker [24]. This is how the Jupyteo platform was created.

Jupyteo is an online integrated development environment (IDE) for earth observation data processing available on a cloud platform. It was created based on an earlier project: JupyTEP IDE [25]. The current version – Jupyteo – is updated, rebuilt and is more extensive than the original – JupyTEP IDE. The main objective of building the Jupyter notebook IDE for EO data processing (Jupyteo) was to extend the Jupyter software ecosystem [26] and customize the existing components for the needs of EO scientists and other professional and non-professional users strongly related to the EO data community. The general approach was based on the configuration, customization, adaptation and mainly integration of Jupyter, Docker, EO data cloud infrastructure and accessible libraries, EO data tools (application programming interface (API), European Space Agency (ESA) sentinel application platform (SNAP) [27], Orfeo Toolbox (OTB) [28] and geospatial data abstraction

library (GDAL) [29], etc.). Jupyteo also contains a set of extended Docker Stack based on predefined Docker images and designated for different processing environments and different tasks, such as machine learning, advanced scientific data manipulation and SAR or GIS data processing.

Jupyteo is based on a web-based user interface in the form of an extended and modified Jupyter user interface (UI) with a customized layout, EO data processing engine and a set of predefined notebooks, widgets and tools (Fig. 1). The final IDE is targeted to remote sensing experts, scientists and users who can develop the Jupyter notebook by reusing embedded open-source tools, WPS interfaces and existing notebooks. A fully scalable Docker environment is suitable for the demanding and resource-consuming EO data processing community and automatic tasks related to the processing and development of scripts and algorithms. Jupyteo is also equipped with a spatial data viewer based on the Leaflet plugin for web browsers as a presentation layer. It is used to browse EO datasets and display the results of the processing running in Jupyteo on a map.

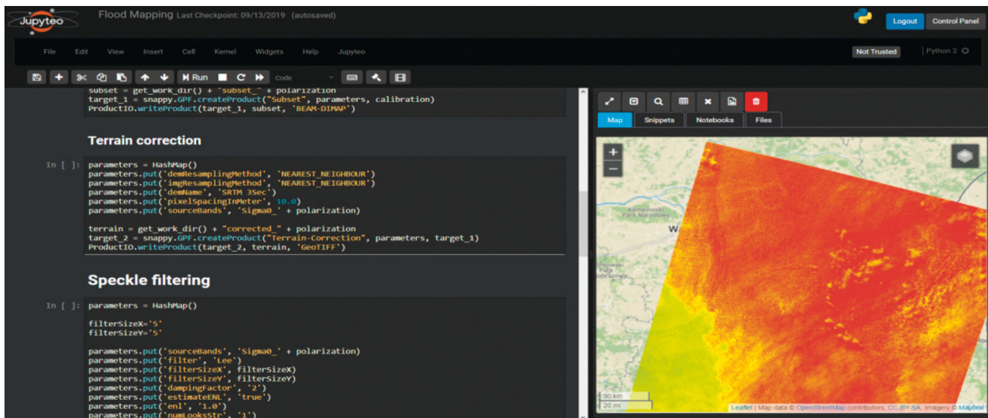


Fig. 1. Screenshot of Jupyteo IDE Earth observation processing tool main screen

The Jupyteo platform is available at https://www.jupyteo.com/. It was created and is maintained by WASAT sp. z o.o. It is made available to external users and used by WASAT in ongoing work for implementing tasks and cooperation within scientific projects. Tasks performed using Jupyteo concern the development of data processing algorithms (data science), mainly in the field of spatial data processing, including Earth observation (EO) data and statistical analyses. The platform is also used to validate all of the new solutions and algorithms. Jupyteo, as a demanding platform, is constantly updated and extended. Jupyteo and all its components run under a Linux system encapsulated in Docker containers. Any further considerations and examples in this paper also apply to software running under Linux.

1.2. Motivation

If one needs to analyze and edit spatial information or compose and export graphical maps using an open-source GIS system, QGIS is a good choice. It supports vector and raster layers in many formats. QGIS is also well-integrated with other open-source GIS packages, including PostGIS, GRASS GIS, and MapServer. Plugins written in Python or C++ extend QGIS's capabilities. Plugins can help for example with geocoding using the Google Geocoding API, perform geoprocessing similar to ArcGIS's standard tools and provide interfaces to PostgreSQL/PostGIS SpatiaLite and MySQL databases.

However, there is occasionally a need to perform statistical analysis, visualize a graph, train and use a machine learning model or access cluster data sources using PySpark and Hadoop. The best way to perform such tasks is to use a programming language, e.g. Python and a dedicated online platform such as Jupyteo with all libraries and APIs preconfigured and integrated onboard.

There are occasionally projects which are needed to use both GIS and a dedicated platform to perform several advanced tasks. In this case, the exchange of data between both environments is necessary. However, it is a non-standard activity where a case-by-case approach is often required, especially when it comes to exchanging data between different environments and formats. Therefore, a question arises that at the same time allows the purpose of this study to be formulated: what is the most convenient way to combine the functionality of the GIS (QGIS) system with the functionality of the Jupyteo platform in one tool? In this article the author wishes to share his thoughts on the problem and discuss how best to solve it.

Therefore, the main goal of this work was to develop a solution that allows for a convenient combination of the functionality of the QGIS system with the functionality of the Jupyteo platform (Jupyter). The way to achieve this goal was to create a library in Python, providing the API for Jupyter scripts (notebooks) that made this connection possible. It should be emphasized here that at the time of commencement of works, there was no solution providing similar functionalities for both the Jupyteo and Jupyter platforms. In the sources, one can find some attempts to solve the problem in question, but none of the presented methods turned out to be sufficient. They are discussed later in this article on current trends among solutions for Jupyter to enable cooperation with QGIS.

Thus, the assumption was to combine the functionality of QGIS with the Jupyteo service. Jupyteo, as a web-based tool, gives online access, works in the cloud, has access to repositories of spatial data and has the possibility of parallel and distributed processing. Moreover, Jupyteo has many tools pre-installed and implemented, such as the already mentioned Scikit-learn, TensorFlow, or Pandas (and others). However, QGIS allows the increased performance of spatial analyses and map editing, while Jupyteo does not have such advanced functionalities. It was necessary to analyze possible scenarios and choose the most optimal and useful approach to combine both environments.

The common element that connects both environments is the Python interface. QGIS has its Python interface – PyQGIS, which gives access to its functionality. In turn, in Jupyteo, Python is the primary programming language. Since Jupyteo is based on Jupyter, the search for a solution began with analyzing existing solutions that allow Jupyter to interact with QGIS.

The sought after solution should work in such a way as to enable the notebook to be integrated with QGIS using as little source code as possible. It should be ready to use with a single library call. The same applies to individual functions such as reading data from layers, saving or analyzing. Each of them should be supported by one or several necessary methods. Thanks to this, the obtained solution will be easy to use in many projects without the need to unnecessarily increase the volume of the source code of the scripts, making them more readable and easier to modify. The author's experience has so far showed that the functionalities needed to work with QGIS in notebooks using the PyQGIS interface are often complex. This can be seen in the code snippet provided in section 2.2 of this paper. This snippet is an example of printing the list of layers, where instead of a single call (e.g. listLayers), the user needs to provide a loop to iterate and display the result. The solution discussed in this paper should be able to somehow "hide" this complexity and simplify the work. Several existing solutions on this subject were analyzed, three of which are described below.

1.3. Current Trends among Solutions for Jupyter to Enable Cooperation with QGIS

The search showed that there are relatively few existing solutions that enable QGIS to cooperate with Jupyter. Three of them were taken into consideration:

1) Simple import of "qgis" library into Python script.
2) Use of the extension for Jupyter – 3Liz nbextension.
3) Connection of a Jupyter / IPython notebook to the Python console in QGIS.

Jupyteo is a server-side web-based system. Each modification of its components requires installation or uninstallation of software components on the platform's backend on which it is running. Generally speaking, Jupyteo instance based on the Linux system runs in the form of a Docker container with all necessary components and configurations. For this, a separate Docker image is preconfigured with all necessary components predefined in a Docker file. A QGIS image is based on an extension of SciPy Jupyter Docker Stack, an entry point for the definition of a QGIS Docker image for Jupyteo. At the stage of starting the Jupyteo QGIS environment, a Docker container with all QGIS configurations boots in the form of a highly usable encapsulated system with all necessary QGIS-related libraries. The last part of the starting process is opening a Jupyter notebook and importing post configuration scripts that enable paths for QGIS resources.

Simple Import of a "qgis" Library into a Python Script

When QGIS is installed in the operating system, the Python interface for QGIS can be connected to a notebook script by importing libraries associated with it. Here is an example use of the QGIS library. Throughout this paper, "In [1]:" and "Out [1]:" statements in the code stand for input-output IPython/Jupyter cells, respectively.

1. Import QGIS:

```
In[1]:
  import qgis
```

2. Import needed libraries e.g.:

```
In[2]:
  from qgis.gui import *
  from qgis.core import *
  from PyQt5.QtCore import *
  from qgis.analysis import QgsNativeAlgorithms
```

The above example illustrates the most basic method. Thanks to this, PyQGIS can be used in a similar way to the built-in Python console in QGIS, apart from functions related directly to the QGIS GUI. However, creating a functional notebook script and performing more advanced operations requires a large amount of source code. This will make it more difficult, for example, to display the content of GIS layers or processing results. Loading data into tools used in data science, such as Pandas or PySpark, will also be problematic because PyQGIS is not compatible with them, as well as it is not compatible with Jupyteo (Jupyter) notebooks. Of course, this does not mean that working with QGIS in this way is impossible, but it can be said that it might be complicated.

Use of the Extension for Jupyter – 3Liz nbextension

Jupyter (and thus Jupyteo) allows adding software components in the form of so-called notebook extensions (nbextensions). It is a plugin-like mechanism. Extensions can be downloaded or created by users. In the case of creation, the extension must be prepared according to the template provided on the Jupyter [30] project pages. It can then be installed in the Jupyter environment. Thanks to this, new and non-standard functionality is added.

Based on this mechanism, another way of Jupyter's cooperation with QGIS is available [31]. This approach is shared on Github under the name qgis-nbextension by 3Liz.com. Once installed, there is no need to import the QGIS library directly into the notebook script. Access to QGIS functionality is available via so-called IPython magic commands, for example:

```
In[1]:
% load_ext qgis_ipython
% qgis --verbose
from qgis.core import Qgis, QgsProject, QgsMapSettings
```

By using this extension, the necessity of connecting to the qgis library is avoided. However, it remains necessary to import individual classes depending on the activities that are to be performed using PyQGIS. Moreover, qgis-nbextension still does not facilitate the collaboration of QGIS with data science and Jupyteo-related tools.

Connection of a Jupyter / IPython Notebook to the Python Console in QGIS

It is also possible to reverse the procedure and connect a Jupyter notebook to QGIS [32]. In this case, it is possible to run notebooks from the Python console in QGIS. However, this method requires Jupyter to be installed on the local machine along with QGIS. In cooperation with a web-based system such as Jupyteo, this method will not be appropriate.

1.4. Clarifcation of the Objectives

After becoming acquainted with the possibilities offered by the existing solutions, their functionalities were compared with the requirements for the solution sought. The results of the comparison are summarized in Table 1. The comparison shows that none of the existing solutions offers all the required functionalities. On this basis, the main goal of this paper could be formulated, which was to create a solution that would enable cooperation between Jypyteo / Jupyter notebook and QGIS, taking into account all the requirements listed in Table 1. It was assumed that this solution would be a Python library providing a properly constructed API.

To achieve the goal, the following objectives were formulated:

- Installation and configuration of QGIS in the Jupyteo platform system. It is not about QGIS software with a graphical interface, but the ability to access PyQGIS from the operating system shell.
- Preparation of a test data set.
- Analysis of data access methods (read / write) and analysis in QGIS projects using the PyQGIS interface.
- Development of a method of transferring map styling from a very extensive QGIS environment to the simplified Leaflet browser, which is used in Jupyteo.
- Implementation of the developed functionalities, such as those mentioned above, inter alia: opening the QGIS project file in a notebook, reading, writing, viewing descriptive and graphical data, cooperation with Pandas tables.
- Implementation of the test case including:
  - data read from QGIS project using Python notebook script
  - processing and analysis in Jupyteo notebook – Python script, especially using tools unavailable in QGIS like Pandas and Scikit-learn
  - re-saving the results in the QGIS project,
  - reading and presentation of the obtained results both in QGIS UI and in Jupyteo.

Table 1. Comparison of the features of the analyzed solutions according to the required functionalities

Functionality description	Simple import	3Liz nbextension	Python console in QGIS	Required
Connection between notebook and QGIS	Yes	Yes	Yes	Yes
Running notebooks that require the installation of additional components on the Jupyter platform	Yes	Yes	No	Yes
Running notebooks remotely online	Yes	Yes	No	Yes
Methods providing data science-related functionality (e.g. direct use of Pandas dataframes (read/write data))	No	No	No	Yes
Methods providing direct web browser viewer layers visualization	No	No	No	Yes
Direct access to Jupyteo EO data repositories	No	No	No	Yes
Other methods simplifying access to QGIS functionality (source code simplification and readability)	No	No	No	Yes

2. Solution Overview, Design Methods and Tools

In response to the problem posed, the author decided to create a Python library called JupyQgis, which could be used from the script level in Jupyteo (Fig. 2). The JupyQgis library provides an API for communication with QGIS via PyQGIS and integrates the functionality of both QGIS and the Jupyteo platform. The library was attached to the Jupyteo platform's API, thanks to which it is available to all users without the need to install it separately on individual virtual machines.

A need to develop the JupyQgis library appeared during research based on EO data using the Jupyteo platform. More than once, projects that our team faced required the use of GIS functionality and performing analyzes in an external environment, which was most often QGIS. Thus, problems arose which allowed to formulate requirements for Jupyteo's cooperation with QGIS. The formulation of the requirements made it possible to identify individual functionalities that should be implemented to achieve the assumed goal. Due to the specificity of the Jupyteo platform, which is based mainly on the use of Python scripts, the most universal form of the solution seemed to be the Python library, integrated with the Jupyteo platform environment, providing the appropriate API.

In general terms, the methodology for developing JupyQgis can be presented as follows:

- Requirement specification – identification of problems to be solved and activities that can be automated.
- Analysis of the existing solutions related to the problem of cooperation between Python notebooks and QGIS.
- Designing the API structure, developing functional assumptions for (programming) methods used in the API.
- Implementation of the developed methods in the form of the JupyQgis library in Python.
- Ongoing code testing and fixes.
- Testing the implemented API functionality.

JupyQgis is constantly being developed and tested along with research in other projects. The main element of the JupyQgis library is the JpQgis class. It contains a set of methods needed for, among other things:

- establishing a connection with the QGIS project,
- viewing information about individual layers,
- reading data from individual layers,
- modification of the table structure of individual layers,
- data editing in layer tables,
- displaying a graphical representation of layers in the map viewer built into Jupyteo.

There are also methods for processing and analyzing EO and GIS data not available in QGIS, such as: merging data from Pandas tables with QGIS tables or performing fundamental statistical analyses such as, e.g., correlation analysis, linear regression or Tukey's test between selected GIS attributes together with an illustration in a graph.

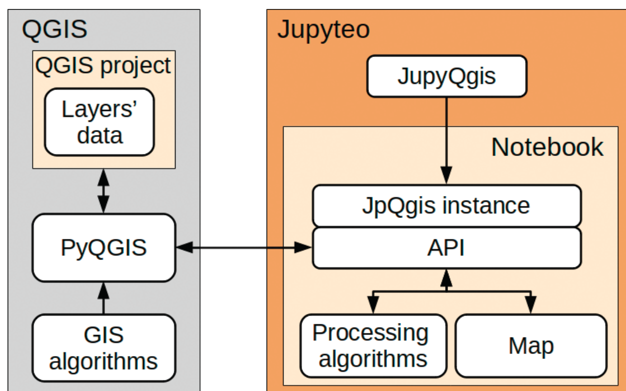


Fig. 2. Cooperation schema between Jupyteo and QGIS with JupyQgis library usage

The JupyQgis library works with any project created in QGIS version 2 or 3. The QGIS project, together with layer files, must be placed on the Jupyteo server by uploading project files, or it should be accessible in any other network location that allows remote data reading and writing to access the data. To illustrate the developed solution, selected functionalities of the JupyQgis library are presented and discussed below.

Working with JupyQgis begins by establishing a connection with the QGIS project by creating a JpQgis instance. The path and filename of the QGIS project must be passed as a string parameter to the constructor method. In this way, one can access all fields and methods of the source class:

```
In[]:
from jupyQgis import *
jpq = jpQgis(<path to qgs of qgz project file>)
```

There are two ways of accessing QGIS data by JupyQgis:
1) direct access to the QGIS project,
2) access through methods of the JpQgis class.

2.1. Direct Access to the QGIS Project

JpQgis allows access to a QgsProject instance through the JpQgis.project field:

```
In[]:
qgs_project_instance = jpq.project
```

This object gives full access to the opened QGIS project and can be accessed by using the PyQGIS methods described in the QGIS documentation [33]. For the purposes of this article, such a mode can be called standard access.

2.2. Access to Descriptive Data through JpQgis Class Methods

This access differs from standard access because the JpQgis class methods enable integration with Jupyteo and provide additional functionality. One of the goals behind the creation of this library, along with the integration with Jupyteo, was to simplify the syntax for the implementation of individual functionalities related to PyQGIS. For example, to list all layer names available in the project, one can use the listLayers() method:

```
In[]:
jpq.listLayers()

Out[]:
['layer1Name','layer2Name','layer3Name']
```

The above code will return a Python list containing names of all layers available in the project. To get the same result with PyQGIS, along with opening the QGIS project, one would have to use the following code:

```
In[]:
#Open QGIS project
prj = QgsProject()
prj.read(<path to QGIS project file>)

#Build array with layer names
layersTmp = []
for layer in self.project.mapLayers().values():
  layersTmp.append(layer.name())
print layersTmp
```

As shown in the example above, thanks to JupyQgis, both operations – opening a project and displaying an array of names – could be reduced into two lines of code. The same will be in the case of any other function implemented in JupyQgis. For example, to display the metadata of the selected layer, the getLayersFieldNames() method can be used:

```
In[]:
jpq.getLayerFieldNames(<layer name>)
```

Access to the data contained in the layer's table using JpQgis methods is realized via Pandas. Thanks to this, data analysis and processing scope have become significantly expanded because the Pandas library has extensive functionality in this area [34]. Moreover, it is very popular and fast, and many other data science libraries are compatible with it, such as Scikit-learn, NumPy, Matplotlib and PySpark, which significantly facilitates the integration and exchange of data between different environments.

For example, let us assume that there is a QGIS layer named 'wojewWGS84'. To get its table data as a Pandas DataFrame data structure, one can use getLayerTableData() method from the JpQgis class:

```
In[]:
import pandas as pd
df = jpq.getLayerTableData('wojewWGS84')
```

To manipulate this table, one can then access it like any other Pandas DataFrame. For example, to select a particular record and attributes, use the code:

```
In[]:
columns=['ID_WOJ','KOD_TERYT','NAZWA']
df[df['ID_WOJ']==3][columns]
```

The results of the above examples of using the JupyQgis library are shown in Figure 3.

Fig. 3. Loading data from QGIS to Jupyteo script with JupyQgis library usage

2.3. Presentation of Graphic Data in Jupyteo Web-map Browser

The presentation of graphic data is one of the essential functions of GIS systems. The Jupyteo platform is designed to process spatial data and therefore it has been equipped with a map viewer. As it is a network solution, the Jupyteo map viewer uses the well-known and popular Leaflet library. However, it is not compatible with QGIS projects and, compared to QGIS, it supports only a few formats. Among these formats – apart from the internal vector format – there are also GeoJson, SVG, JPG, PNG and WMS. When working with QGIS, the user can choose different formats. Implementation of various format support by Jupyteo could turn out to be troublesome and unprofitable. For this reason, a method to automatically convert graphic data to a Leaflet-supported format has been developed. This functionality is currently in the testing phase and works only with vector layers. When a QGIS layer is displayed in the Jupyteo map viewer, it is automatically converted to the GeoJson format and then goes to the map view (Fig. 4).

However, after this conversion, GeoJson data does not contain styling information. Thus, there is a need to acquire additional information about the layer styles during the conversion process. This information is saved differently depending on the styling method used by QGIS for a particular layer. If the layer has a single style for all features, the situation is quite simple. It is only necessary to read the color or line style information and save it along with GeoJson data. However, if a classification has been used for a layer – e.g. due to the individual value of an attribute

or ranges of values – then each feature may have a different appearance style. Individual styling in QGIS is performed using algorithms, each appropriately adapted to a specific method of the layer's chosen style. For this, the QGIS renderer class is used, assigned individually to each layer depending on the styling settings, which may change at any time while working with the program. This is a very flexible mechanism from the point of view of PyQGIS API users. However, styling information is not permanently saved with graphic data because layer styling is done on the fly. For this reason, to obtain styling information for Jupyteo, one had to refer to the individual layer settings via the PyQGIS API and save them in an additional column as a GeoJson layer attribute. This was necessary because there is no QGIS styling mechanism equivalent on the Leaflet library side. Thanks to this, the layer displayed in Jupyteo looks practically the same as in QGIS (Fig. 5).

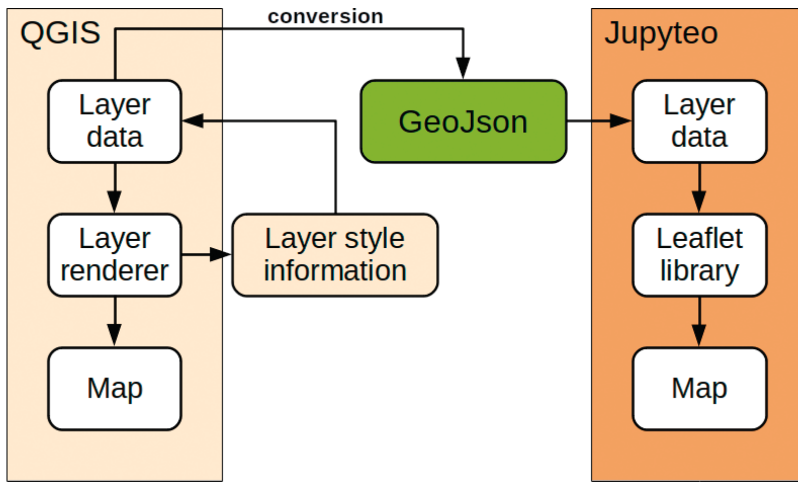


Fig. 4. Data conversion to GeoJson format with layer style information acquisition

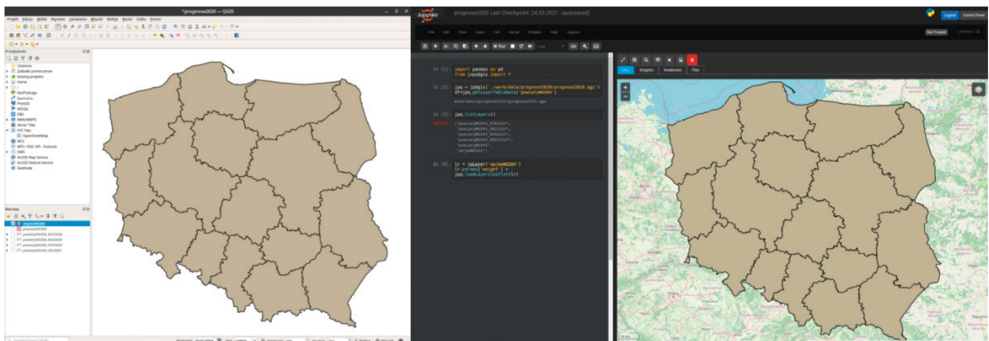


Fig. 5. The same layer data presented in QGIS (left) and Jupyteo (right), converted using the JupyQgis library

3. Application of the JupyQgis Library in Data Processing

The JupyQgis library was created to add GIS functionality to the Jupyteo IDE platform. Thanks to this, it is possible to process the same spatial data sets in both tools in one place. The user has at his disposal such functionalities as:
- downloading data from the QGIS project,
- processing the data on the Jupyteo side,
- calling tools for processing GIS spatial data in Jupyteo,
- saving processing results back to the QGIS project.

The described library is created as a convenience in work on projects that require the processing of both EO data from repositories available in Jupyteo and GIS analytical data and tools. It is also important to consider how the JupyQgis library might be applied in spatial data processing. A sample test case has been developed for this purpose, which requires using GIS vector data and Python data science tools.

3.1. Sample Test Case: Spatial Data Processing

The test case analysis consisted of forecasting the average prices of a square meter of residential real estate in the following statistical year and presenting the results in a thematic map. Since the data at the author's disposal concerned the years 2019 and earlier, forecasts covered the year 2020. The data were obtained from the current databases of the Polish Central Statistical Office [35] and the author's own studies. The conducted analysis is of a statistical and illustrative nature and should not be treated as a method of property valuation, e.g. for market purposes. The main purpose of its conduct was to prepare a test case for the developed JupyQgis library.

Data sets constituting input data for analysis included:
- Own data sources in the form of the QGIS project:
  - administrative map of Poland with a layer of provinces and the data table with codes and names of provinces,
  - administrative map of Poland with a layer of districts and the data table with codes, names of districts and prices of a square meter of real estate in 2019.
- Data obtained from databases of the Central Statistical Office concerning individual districts. Compiled over past years containing:
  - number of residential buildings (2008–2019),
  - average population density (2002–2019),
  - average salaries (2005–2019),
  - average price per square meter of residential premises (2015–2019).

Real estate appraisal is not the subject of this article, but the author decided to provide some details related to this issue due to the test case described. The value

of a property depends on many characteristics. Their effect on the price is different, depending on the national economic and market conditions in which that property is located [36]. When valuing a residential property, the following factors are taken into account: access to roads and communication, distance from the city center, access to power, water and sewage networks, proximity to green and recreational areas, prices of similar properties and many others [37, 38].

Since the publicly available statistical data covering the entire territory of Poland are not that detailed, several features were selected which apply to the whole country and generally result from specific features customarily used in property valuation.

For the purposes of the current paper, it was assumed that the population density and number of residential buildings are associated with the availability of other characteristic features of cities such as a denser road network or an extensive water supply and sewage network. Therefore, they can be treated as a summary generalization of the influence of urban development features. When the population density and the number of buildings are of greater value in a specific area, the area can be treated as more industrialized and vice versa. Analyses presented by financial institutions related to the real estate market also show a correlation between salaries and real estate value in Poland [39, 40]. In recent years, along with the increase in salaries and their level in individual regions, the real estate value has increased. Therefore, this feature is also used in this article as affecting the value of a property.

Time is another quite important factor. It is observed that in a country such as Poland, the value of real estate increases in the following years. There are also regions where the real estate price is higher or lower than in others. This is due, for example, to the level of industrialization (large cities) or the development of tourism (sea, lakes, forests, or mountains). Price differentiation due to the above-mentioned factors is permanent and is related to a specific location. Information on each district's location is expressed by the code value in the KOD_TERYT attribute and was also included in the forecast.

The forecast algorithm consisted of several stages (Fig. 6). Work started with the preparation and compilation of data. Data from both QGIS and the Central Statistical Office were loaded into one script in Jupyteo, where they were standardized and appropriately processed. For the statistical data, it was necessary to select only the columns required in tables, complete province identifiers, standardize column names, change the decimal separator for numerical values and delete records with missing values.

The individual datasets were then combined into one table. For connection, the TERYT administrative unit identifier used in Poland, common to all records, was used. This value appears in described datasets in the field "KOD_TERYT" and applies to individual districts. As a merge result, a dataset was created with the structure shown in Figure 7.

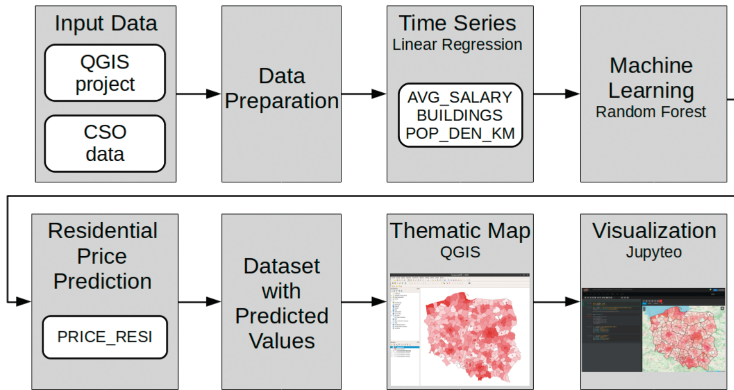


Fig. 6. Sample test case of spatial data processing – algorithm diagram

dfPRCmean
KOD_TERYT
NAME
YEAR
AVG_SALARY
BUILDINGS
POP_DEN_KM
PRICE_RESI

Fig. 7. Input dataset structure

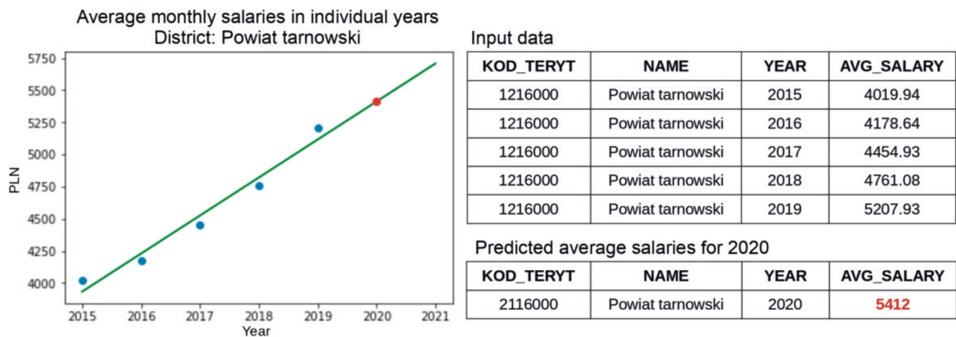
The resulting dataset contains all attributes that have been selected for forecasting the value of a square meter of residential property, broken down by years in individual districts, including:

- AVG_SALARY – average salaries,
- BUILDINGS – number of apartment buildings per district,
- POP_DEN_KM – population density per square kilometer in a district,
- YEAR – the year for which a given record was compiled in the table,
- KOD_TERYT – territorial unit identifier (district), thanks to this attribute, the property location in the country was taken into account in the prediction,
- PRICE_RESI – square meter property price in the district.

The dataset was divided into training and testing parts. In order to train the model, the Scikit-Learn library and a random forest algorithm were used. Model validation showed an MAE error at the level of PLN 366 and accuracy at the level of PLN 0.96. The resulting model allows forecasting the price of a square meter (PRICE_RESI) based on the following attributes: AVG_SALARY, BUILDINGS, POP_DEN_KM, YEAR, KOD_TERYT.

As the data needed for the test case were available only for 2019 and earlier, the developed model will also forecast PRICE_RESI values from this period. In order to obtain the values for 2020, it was decided to supplement individual input attributes with the predicted values for 2020. For this task, the prediction was carried out using linear regression models built for the time series of individual attributes in relation to districts. This operation concerned attributes, which tend to change in time, i.e. AVG_SALARY, BUILDINGS, POP_DEN_KM.

For example, the predicted value of AVG_SALARY for 2020 for the district "Powiat tarnowski" was PLN 5412 (Fig. 8). The time series for its determination was built based on data from 2015–2019. The values of other input attributes were automatically predicted in the same way for each district.



Input data

KOD_TERYT	NAME	YEAR	AVG_SALARY
1216000	Powiat tarnowski	2015	4019.94
1216000	Powiat tarnowski	2016	4178.64
1216000	Powiat tarnowski	2017	4454.93
1216000	Powiat tarnowski	2018	4761.08
1216000	Powiat tarnowski	2019	5207.93

Predicted average salaries for 2020

KOD_TERYT	NAME	YEAR	AVG_SALARY
2116000	Powiat tarnowski	2020	5412

Fig. 8. Illustration of one of the linear regression models, which were automatically built for input attributes prediction for 2020

The data supplemented with values from time series forecasting was then used to predict the PRICE_RESI values for 2020 using the previously described machine learning random forest algorithm. The resulting dataset was transferred to the QGIS project using the JupyQgis library. A thematic map was then prepared to illustrate the distribution of forecasted values in individual districts, which was the ultimate goal of the test case to be achieved.

3.2. Example of Data Processing with JupyQgis

To illustrate the functionality, use and role of the JupyQgis library in data processing, selected essential fragments of the Jupyteo/Python code implementing the described algorithm are presented below.

1. Import required libraries, among others jupyQgis:

```
In[1]:
from jupyQgis import *
import pandas as pd
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
```

2. Load data from QGIS project to Pandas dataframe. It is done with jpQgis object and getLayerTableData method, both from JupyQgis library. As it was discussed earlier in chapter 2.2., accessing data with JupyQgis is simpler than with PyQGIS or pure Python.

```
In[2]:
jpq = jpQgis('./data/admin_qgis84.qgs')
dfQgis=jpq.getLayerTableData('powiatyWGS84')

Out[2]:
['powiatyWGS84', 'wojewWGS84']
```

3. Load remaining datasets. The part of the script that carries out transformation and adaptation of datasets for further processing is omitted here.

```
In[3]:
dfBUD = pd.read_csv('./data/BUILDINGS.csv',dtype=object)
dfPOP = pd.read_csv('./data/PEOPLE_DENS.csv',dtype=object)
dfSAL = pd.read_csv('./data/SALARIES.csv',dtype=object)
dfPRC = pd.read_csv('./data/RESIDENTIAL_PRICE.csv',dtype=object)
```

4. Calculate annual average prices for each district and merge loaded datasets. The resulting dataframe is:

```
In[4]:
dfAll.head()

Out[4]:
```

	KOD_TERYT	KOD_TERYT_NUM	KOD_TERYT_WOJ	NAME	YEAR	AVG_SALARY	BUILDINGS	POP_DEN_KM	PRICE_RESI
5	0200000	200000	0200000	DOLNOŚLĄSKIE	2015	4283.33	365052	146	3959.75
6	0200000	200000	0200000	DOLNOŚLĄSKIE	2016	4455.69	369809	146	4100.00
7	0200000	200000	0200000	DOLNOŚLĄSKIE	2017	4722.44	375103	146	4060.00
8	0200000	200000	0200000	DOLNOŚLĄSKIE	2018	4993.63	381068	145	4271.00
9	0200000	200000	0200000	DOLNOŚLĄSKIE	2019	5388.46	395328	145	4527.00

5. Create and train a machine learning model using the Scikit-Learn Random Forest Regressor. It will be used later.

```
In[5]:
SAMPLE_COLS = [
        'KOD_TERYT','YEAR','AVG_SALARY','BUILDINGS','POP_DEN_KM']
TARGET_COLS = ['PRICE_RESI']
dfeval = dfAll.sample(100)
dftrain = pd.concat([dfAll,dfeval]).
                             drop_duplicates(keep=False)
X_test = dfeval[SAMPLE_COLS]
y_test = dfeval[TARGET_COLS]
X_train = dftrain[SAMPLE_COLS]
y_train = dftrain[TARGET_COLS]
regressor = RandomForestRegressor(
                        n_estimators=20, random_state=0)
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
```

6. Validate the model.

```
In[6]:
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=20)
n_scores = cross_val_score(regressor, X_train, y_train,
               scoring='neg_mean_absolute_error', cv=cv, n_jobs=-1)
n_scores = absolute(n_scores)
print('MAE: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))

Out[6]:
MAE: 366.113 (19.537)
Accuracy: 0.9632694888641313
```

7. Predict the values of individual attributes for each district for 2020. Linear regression was used for forecasting. For this purpose, the makePrediction () function was created, which automatically builds a time series separately for each district based on the values of a given attribute in individual years, then creates a linear regression model and returns the value forecast for the following year. Finally, the forecast results were combined into a single dataset called dfPredictions.

```
In[7]:
dfBudProg = makePrediction(dfQgis,dfBUD,2020,'BUILDINGS',
                     'BUD_2020','KOD_TERYT')
dfPopProg = makePrediction(dfQgis,dfPOP,2020,'POP_DEN_KM',
                     'POP_2020','KOD_TERYT')
dfSalProg = makePrediction(dfQgis,dfSAL,2020,'AVG_SALARY',
                     'SAL_2020','KOD_TERYT_WOJ')
dfPredictions = dfBudProg.merge(dfPopProg, how = 'inner',
                            on='KOD_TERYT')
dfPredictions = dfPredictions.merge(dfSalProg, how='left',
                            on='KOD_TERYT_WOJ')
df.Predictions.head()

Out[7]:
```

	KOD_TERYT	KOD_TERYT_WOJ	BUD_2020	POP_2020	SAL_2020
0	2211000	2200000	20523.666667	152.457516	5233.262476
1	2215000	2200000	36914.803030	173.300654	5233.262476
2	2208000	2200000	9264.696970	95.163399	5233.262476
3	2212000	2200000	18294.287879	43.594771	5233.262476
4	3213000	3200000	10140.681818	54.588235	4841.589810
5	2262000	2200000	18422.287879	1814.764706	5233.262476

8. Next, the dfPredictions dataset was used to forecast prices for a square meter of real estate using the random forest model previously created and mentioned in point 5.

```
In[8]:
dfProgEval = dfPredictions[
                ['KOD_TERYT','SAL_2020','BUD_2020','POP_2020']]
```

```
dfProgEval.insert(1, 'YEAR', 2020)
y_pred = regressor.predict(dfProgEval)
dfPredictions['RESI_2020']=y_pred
dfPredictions.head()
```

Out[8]:

	KOD_TERYT	KOD_TERYT_WOJ	BUD_2020	POP_2020	SAL_2020	RESI_2020
0	2211000	2200000	20523.666667	152.457516	5233.262476	5064.3125
1	2215000	2200000	36914.803030	173.300654	5233.262476	4337.8875
2	2208000	2200000	9264.696970	95.163399	5233.262476	2935.7250
3	2212000	2200000	18294.287879	43.594771	5233.262476	3260.7750
4	3213000	3200000	10140.681818	54.588235	4841.589810	4136.3000
5	2262000	2200000	18422.287879	1814.764706	5233.262476	6135.7375

9. Convert the data types to prepare for loading prediction results to the QGIS project.

In[9]:

```
dfPrognoseResult = dfPredicitions
dfPrognoseResult['SAL_2020']=dfPrognoseResult1['SAL_2020'].
          apply(float).apply(int).apply(str)
dfPrognoseResult['BUD_2020']=dfPrognoseResult1['BUD_2020'].
          apply(float).apply(int).apply(str)
dfPrognoseResult['POP_2020']=dfPrognoseResult1['POP_2020'].
          apply(float).apply(int).apply(str)
dfPrognoseResult['RESI_2020']=dfPrognoseResult1['RESI_2020'].
          apply(str)
```

10. Use the JupyQgis library to create columns and merge Pandas Dataframe into QGIS table with layerMergePandasDf method from JupyQgis library.

In[10]:

```
newFields = [QgsField("SAL_2020",QVariant.Int,"Integer",10),
                QgsField("BUD_2020",QVariant.Int,"Integer",10),
                QgsField("POP_2020",QVariant.Int,"Integer",10),
                QgsField("RESI_2020",QVariant.Double,"Double",12,2)]
jpq.layerMergePandasDf('powiatyWGS84',dfPrognoseResult,
                newFields,'KOD_TERYT')
```

11. Load the thematic map developed in QGIS based on the performed real estate price forecast for 2020 to Jupyteo. First, QGIS project is opened and available layers are listed using mentioned earlier jpQgis and listLayers. Then layers are styled and loaded into map viewer using jpLayer object and loadLayer2leaflet method from JupyQgis. The result is shown in Figure 9.

In[11]:

```
jpq = jpQgis('./work/data/prognose2020/prognose2020_RF.qgz')
df=jpq.getLayerTableData('powiatyWGS84')
jpq.listLayers()
```

```
Out[11]:
['powiatyWGS84_POP2020',
       'powiatyWGS84_SAL2020',
       'powiatyWGS84_BUD2020',
       'powiatyWGS84_RESI2020',
       'powiatyWGS84',
       'wojewWGS84']

In[12]:
lr = jpLayer('powiatyWGS84_RESI_2020')
lr.params['weight'] = 1
lr.params['fill']=['true']
lr.params['fillOpacity']=0.5
jpq.loadLayer2leaflet(lr)
lr = jpLayer('wojewWGS84')
lr.params['weight'] = 2
lr.params['fill']=['false']
jpq.loadLayer2leaflet(lr)
```

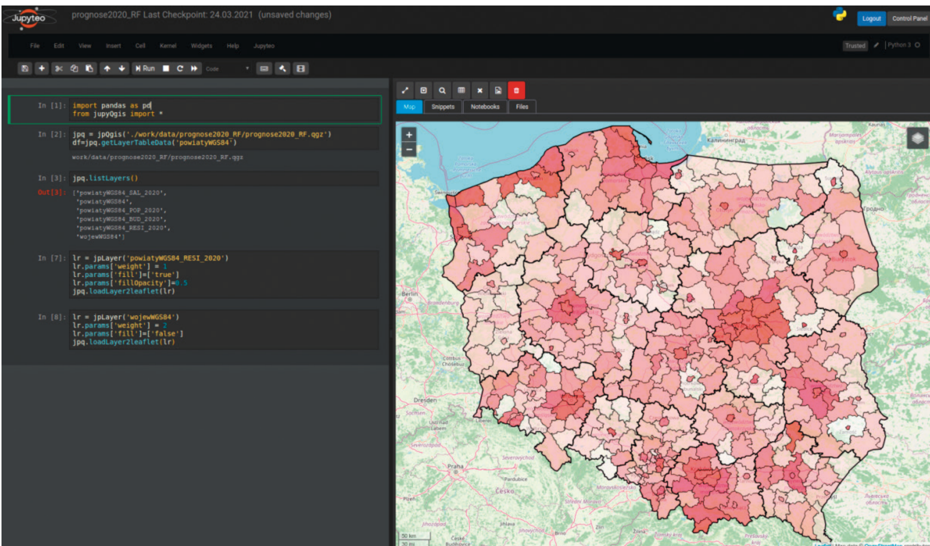


Fig. 9. The thematic map developed in QGIS, based on the performed real estate price forecast for 2020 – loaded in Jupyteo using the JupyQgis library`

4. Discussion

The work carried out in this paper suggests that the most convenient way to combine the functionality of the QGIS system with the functionality of the Jupyteo platform was to create a library providing a straightforward and easy-to-use API. The created API significantly facilitates cooperation between both environments, enabling the QGIS project to be used directly from the Jupyteo script. The interoperability is two-way, which means that the user can easily read data from an existing

QGIS project, process it with tools commonly used in data science and then save the results back to the QGIS project. The use of JupyQgis library shortens and simplifies the source code of the created Python scripts in relation to the original PyQgis. The resulting tool can be easily used and transferred as a software component of any Python or IPython-based platform extending its functionality. Combining the functionality of a GIS system such as QGIS with a network platform such as Jupyteo creates additional possibilities for analyzing and processing GIS data, especially through:

- online cloud data processing,
- using modern and very efficient tools to work with big data, such as Pandas, PySpark and others,
- use of machine learning tools such as Scikit-learn,
- access to data repositories offered by platforms such as Jupyteo (e.g., EO data),
- enabling parallel processing (available in Jupyteo),
- easy data integration from multiple sources,
- enabling new, future solutions and tools that are not yet available by translating a large part of the functionality into processing using a constantly developing language such as Python.

In terms of existing solutions that could be adapted, JupyQgis stands out for its functionality. Using the library involves calling the necessary objects and methods fully integrated with Jupyteo. The existing projects did not meet expectations because they do not fully cooperate with the Jupyteo platform. Using them would involve creating extensive scripts, which, apart from the main functionality, would have to implement cooperation with QGIS.

One of the more difficult problems to solve was presenting spatial data in the form of a map in a Jupyteo map viewer. Leaflet library – component used in Jupyteo uses only one vector format, which is GeoJson. It means that all formats derived from QGIS must be converted to it to be displayed correctly. Additionally, there is a graphic style incompatibility between QGIS and Leaflet. This problem should be given special attention in future work. At the moment, the JupyQgis library created has not yet been thoroughly tested. QGIS is a complex system, which supports many data formats. The design study and test case were based on vector spatial data. The next step will be to develop and test cooperation with raster datasets. At this point, there also might be a problem with converting graphic data formats.

5. Conclusions

The primary purpose of this work was to determine the most convenient way to combine the functionality of the GIS (QGIS) system with the functionality of the Jupyteo platform in one tool.

During the works, it was found out that the existing solutions aimed at QGIS interoperation with Jupyter did not meet most of the assumed requirements. This applies primarily to the possibility of convenient data exchange with data science-related tools (such as Pandas or Scikit-learn), visualization in a web map viewer, or direct, two-way access to the QGIS project (reading and writing data). The elimination of the above-mentioned deficiencies also affects the readability of the created scripts and the simplification of the source code, which translates into a significant work simplification.

To achieve the assumed goal, a practical attempt to solve the problem of interoperability between QGIS and Jupyteo was made. It was found that the most suitable solution would be to create a proprietary library providing API for collaboration between both of the environments mentioned above. The created library meets expectations and enables efficient cooperation. This fact is supported by a practical example of data processing using various tools presented in this article, where data from the QGIS project was imported into a Jupyteo script. The data was easily combined with external data sources and a forecast of the value of real estate data was performed using machine learning algorithms. The forecast results were then transferred from Jupyteo to QGIS, where a thematic map was created. The created map was again displayed in Jupyteo without a problem. It can be said that the described processing chain, combining QGIS and Jupyteo in one process, has been completed. In this way, it was shown that the JupyQgis library fulfils its role and can become another tool used in data science. Thanks to it, it is possible to include data from QGIS for analyses in Jupyteo in real-time, which significantly extends the existing functionality of both environments.

The problem addressed in this paper has only been partially covered in other studies in the literature. Existing studies have offered some solutions but did not provide satisfactory solutions. The current study identified the problems that need to be faced when combining the functionality of an extensive desktop QGIS system with an online platform such as Jupyteo.

The current study considers mainly loading and saving vector data, which certainly narrows the scope of problems that may still arise. In future steps, attention should be paid to the exchange of raster data and cooperation with QGIS analytical tools, paying particular attention to those that save the results in the QGIS project data in real-time.

The emerging JupyQgis library will be published under an open license, which will make the author's contribution to the development of tools related to the processing of spatial data with the use of GIS public.

Acknowledgements

This article has been prepared as part of the work on the project carried out at WASAT Sp. z o.o. with headquarters in Gdańsk, Trzy Lipy 3, Poland. Special thanks I would like to address to Wasat's Navigation and Geoinformation Department Manager – Daniel Zinkiewicz – for the substantive support, all comments to the text and tips on the directions of development of the described library.

References

[1] Aalst W.M.P., Bichler M., Heinzl A.: *Responsible Data Science*. Business & Information Systems Engineering, vol. 59, 2017, pp. 311–313. https://doi.org/10.1007/s12599-017-0487-z.

[2] Janowski A., Szulwic J., Tysiąc P.: *Spatial Modelling in Environmental Analysis and Civil Engineering*. Applied Sciences, vol. 11(9), 2021, 3945. https://doi.org/10.3390/app11093945.

[3] Bacao F., Santos M.Y., Behnisch M.: *Spatial Data Science*. ISPRS International Journal of Geo-Information, vol. 9, 2020, 428. https://doi.org/10.3390/ijgi9070428.

[4] Gibert K., Horsburgh J.S., Athanasiadis I.N., Holmes G.: *Environmental Data Science*. Environmental Modelling & Software, vol. 106, 2018, pp. 4–12. https://doi.org/10.1016/j.envsoft.2018.04.005.

[5] Cheng Y., Zhou K., Wang J., Yan J.: *Big Earth Observation Data Integration in Remote Sensing Based on a Distributed Spatial Framework*. Remote Sensing, vol. 12, 2020, 972. https://doi.org/10.3390/rs12060972.

[6] Mattmann C.: *A vision for data science*. Nature, vol. 493, 2013, pp. 473–475. https://doi.org/10.1038/493473a.

[7] Janowski A., Bobkowska K., Szulwic J.: *3d Modelling of Cylindrical-Shaped Objects from Lidar Data – an Assessment Based on Theoretical Modelling and Experimental Data*. Metrology and Measurement Systems, vol. 25(1), 2018, pp. 47–56. https://doi.org/10.24425/118156.

[8] Janowski A., Szulwic J., Ziółkowski P.: *Combined Method of Surface Flow Measurement Using Terrestrial Laser Scanning and Synchronous Photogrammetry*. [in:] *2017 Baltic Geodetic Congress (Geomatics): BGC Geomatics 2017: proceedings: 22–25 June 2017, Gdansk University of Technology, Poland*, IEEE, Piscataway 2017, pp. 110–115. https://doi.org/10.1109/BGC.Geomatics.2017.54.

[9] Ossowski R., Przyborski M., Tysiac P.: *Stability Assessment of Coastal Cliffs Incorporating Laser Scanning Technology and a Numerical Analysis*. Remote Sensing, vol. 11(16), 2019, 1951. https://doi.org/10.3390/rs11161951.

[10] Tysiac P.: *Bringing Bathymetry LiDAR to Coastal Zone Assessment: A Case Study in the Southern Baltic*. Remote Sensing, vol. 12(22), 2020, 3740. https://doi.org/10.3390/rs12223740.

[11] Guo H., Nativi S., Liang D., Craglia M., Wang L., Schade S., Corban C., He G., Pesaresi M., Li J., Shirazi Z., Liu J., Annoni A.: *Big Earth Data science: an information framework for a sustainable planet*. International Journal of Digital Earth, vol. 13(7), 2020, pp. 743–767. https://doi.org/10.1080/17538947.2020.1743785.

[12] Dumitru C.O., Schwarz G., Castel F., Lorenzo J., Datcu M.: *Artificial Intelligence Data Science Methodology for Earth Observation*. [in:] Soofastaei A. (ed.), *Advanced Analytics and Artificial Intelligence Applications,* IntechOpen, London 2019. https://doi.org/10.5772/intechopen.86886.

[13] Artiemjew P., Chojka A., Rapiński J.: *Deep Learning for RFI Artifact Recognition in Sentinel-1 Data*. Remote Sensing, vol. 13(1), 2021, 7. https://doi.org/10.3390/rs13010007.

[14] Janowski A., Renigier-Biłozor M., Walacik M., Chmielewska A.: *Remote Measurement of Building Usable Floor Area – Algorithms Fusion*. Land Use Policy, vol. 100, 2021, 104938. https://doi.org/10.1016/j.landusepol.2020.104938.

[15] Soille P., Burger A., De Marchi D., Kempeneers P., Rodriguez D., Syrris V., Vasilev V.: *A versatile data-intensive computing platform for information retrieval from big geospatial data*. Future Generation Computer Systems, vol. 81, 2018, pp. 30–40. https://doi.org/10.1016/j.future.2017.11.007.

[16] Stromann O., Nascetti A., Yousif O., Ban Y.: *Dimensionality Reduction and Feature Selection for Object-Based Land Cover Classification based on Sentinel-1 and Sentinel-2 Time Series Using Google Earth Engine*. Remote Sensing, vol. 12, 2020, 76. https://doi.org/10.3390/rs12010076.

[17] Müller M., Bernard L., Brauner J.: *Moving code in spatial data infrastructures–web service based deployment of geoprocessing algorithms*. Transactions in GIS, vol. 14, 2010, pp. 101–118. https://doi.org/10.1111/j.1467-9671.2010.01205.x.

[18] Gomes V.C.F., Queiroz G.R., Ferreira K.R.: *An Overview of Platforms for Big Earth Observation Data Management and Analysis*. Remote Sensing, vol. 12(8), 2020, 1253. https://doi.org/10.3390/rs12081253.

[19] Kadiyala A., Kumar A.: *Applications of Python to evaluate environmental data science problems*. Environmental Progress & Sustainable Energy, vol. 36(6), 2017, pp. 1580–1586. https://doi.org/10.1002/ep.12786.

[20] Raschka S., Patterson J., Nolet C.: *Machine Learning in Python: Main Developments and Technology Trends in Data Science, Machine Learning, and Artificial Intelligence*. Information, vol. 11(4), 2020, 193. https://doi.org/10.3390/info11040193.

[21] Hao J., Ho T.K.: *Machine Learning Made Easy: A Review of Scikit-learn Package in Python Programming Language*. Journal of Educational and Behavioral Statistics, vol. 44(3), 2020, pp. 348–361. https://doi.org/10.3102/1076998619832248.

[22] Bisong E.: *Google Colaboratory*. [in:] *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, Apress, Berkeley 2019, pp. 59–64. https://doi.org/10.1007/978-1-4842-4470-8_7.

[23] Pimentel J.F., Murta L., Braganholo V., Freire J.: *A Large-Scale Study about Quality and Reproducibility of Jupyter Notebooks*. [in:] *MSR 2019: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories: proceedings: 26–27 May 2019, Montreal, Canada*, IEEE Computer Society, Conference Publishing Services (CPS), Los Alamitos 2019, pp. 507–517. https://doi.org/10.1109/MSR.2019.00077.

[24] Cook J.: *Docker for Data Science: Building Scalable and Extensible Data Infrastructure around the Jupyter Notebook Server*, Apress, Santa Monica 2017. https://doi.org/10.1007/978-1-4842-3012-1.

[25] Rapiński J., Bednarczyk M., Zinkiewicz D.: *JupyTEP IDE as an Online Tool for Earth Observation Data Processing*. Remote Sensing, vol. 11, 2019, 1973. https://doi.org/10.3390/rs11171973.

[26] Fernández L., Hagenrud H., Zupanc B., Laface E., Korhonen T., Andersson R.: *Jupyterhub at the ESS. An Interactive Python Computing Environment for Scientists and Engineers*. [in:] *Proceedings of the 7th International Particle Accelerator Conference (IPAC2016), Busan, Korea, 8–13 May 2016*, pp. 2778–2780. https://doi.org/10.18429/JACOW-IPAC2016-WEPOR049.

[27] Zuhlke M., Fomferra N., Brockmann C., Peters M., Veci L., Malik J., Regner P.: *SNAP (Sentinel Application Platform) and the ESA Sentinel 3 Toolbox*. [in:] Ouwehand L. (ed.), *Sentinel-3 for Science Workshop*, ESA Special Publication, vol. 734, Venice 2015.

[28] Grizonnet M., Michel J., Poughon V., Inglada J., Savinaud M., Cresson R.: *Orfeo ToolBox: open source processing of remote sensing images*. Open Geospatial Data, Software and Standards, vol. 2, 2017, 15. https://doi.org/10.1186/s40965-017-0031-6.

[29] Warmerdam F.: *The Geospatial Data Abstraction Library*. [in:] Hall G.B., Leahy M.G. (eds.), *Open Source Approaches in Spatial Data Handling*, Advances in Geographic Information Science, Springer-Verlag Berlin Heidelberg 2008, pp. 87–104. https://doi.org/10.1007/978-3-540-74831-1_5.

[30] Unofficial Jupyter Notebook Extensions: https://jupyter-contrib-nbextensions.readthedocs.io [access: 30.05.2021].

[31] 3liz/qgis-nbextension: https://github.com/3liz/qgis-nbextension [access: 30.05.2021].

[32] Geospatial Python Tutorial – Install Jupyter Notebook in QGIS3: https://lerryws.xyz/posts/Install-Jupyter-Notebook-in-QGIS3 [access: 30.05.2021].

[33] PyQGIS Developer Cookbook: https://docs.qgis.org/3.22/en/docs/pyqgis_developer_cookbook/index.html [access: 30.05.2021].

[34] Pandas website: https://pandas.pydata.org [access: 30.05.2021].

[35] Statistics Poland: https://stat.gov.pl [access: 30.05.2021].

[36] Renigier-Biłozor M., Janowski A., Walacik M.: *Geoscience Methods in Real Estate Market Analyses Subjectivity Decrease*. Geosciences, vol. 9(3), 2019, 130. https://doi.org/10.3390/geosciences9030130.

[37] Zydroń A., Walkowiak R.: *Analiza atrybutów wpływających na wartość nieruchomości niezabudowanych przeznaczonych na cele budowlane w gminie Mosina* [*Analysis of Factors Affecting Value of Undeveloped Plots Allocated for Buildings Development in Mosina Municipality*]. Rocznik Ochrona Środowiska, t. 15, cz. 3, 2013, pp. 2911–2924.

[38] Kucharska-Stasiak E.: Odwzorowanie cech nieruchomości w cenach i skutki dla procesu wyceny [*Reflection of Real Estate Attributes in Prices and Consequences for Valuation Process*]. Studia i Materiały Towarzystwa Naukowego Nieruchomości, t. 18, nr 3, 2010, pp. 7–16.

[39] Muratorplus: *Ceny mieszkań w Polsce – prognozy na 2021*. https://www.muratorplus.pl/inwestycje/inwestycje-mieszkaniowe/ceny-mieszkan-w-2018-r-nowe-mieszkania-mocno-podrozaly-gdzie-ceny-mieszkan-wzrosly-najbardziej-aa-BJAZ-hHVK-d4wc.html [access: 30.05.2021].

[40] Pracuj.pl: *Ceny mieszkań w 2020 – ile średnich pensji potrzeba, aby kupić własne lokum?* 6.11.2020, https://zarobki.pracuj.pl/raporty-i-trendy-placowe/ceny-mieszkan-2020-ile-srednich-pensji-potrzeba-aby-kupic-wlasne-lokum/ [access: 30.05.2021].