

End-user software engineering approach: improve spreadsheets capabilities using Python-based user-defined functions

Tamer Bahgat Elserwy¹, Tarek Aly¹, Basma E. El-Demerdash²

¹Department of Software Engineering, Faculty of Graduate Studies for Statistical Research (FGSSR), Cairo University, Giza, Egypt

²Department of Operations Research and Management, Faculty of Graduate Studies for Statistical Research (FGSSR), Cairo University, Giza, Egypt

Article Info

Article history:

Received May 24, 2024

Revised Oct 31, 2024

Accepted Nov 11, 2024

Keywords:

End-user software engineering

Excel user-defined functions

Standalone Python with excel integration

Visual studio tools office

ABSTRACT

End-user computing enables non-developers to handle data and applications, boosting collaboration and productivity. Spreadsheets are a key example of end-user programming environments that are extensively utilized in business for data analysis. However, the functionalities of Excel have limitations compared to specialized programming languages. This study aims to address this shortcoming by proposing a prototype that integrates Python's features into Excel via standalone desktop Python-based user-defined functions (UDFs). This method mitigates potential latency concerns linked to cloud-based solutions. This study employs Excel-DNA (dynamic network access) and IronPython; Excel-DNA facilitates the creation of custom Python functions that integrate smoothly with Excel's calculation engine, while IronPython allows these Python UDFs to run directly within Excel. Core components include C# and visual studio tools office (VSTO) add-ins, which enable communication between Python and Excel. This approach grants users the chance to execute Python UDFs for various tasks such as mathematical computations and predictions — all within the familiar Excel environment. The prototype showcases seamless integration, enabling users to invoke Python-based UDFs just like built in Excel functions. This study enhances the capabilities of spreadsheets by harnessing Python's strengths within Excel. Future work may focus on expanding the Python UDF library and examining user experiences with this innovative approach to data analysis.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Tamer Bahgat Elserwy

Department of Software Engineering, Faculty of Graduate Studies for Statistical Research (FGSSR)

Cairo University

Giza, Egypt

Email: Tamer.elserwy@gmail.com

1. INTRODUCTION

End-user computing provides a centralized and standardized method for managing applications, devices, and data, leading to enhanced collaboration, scalability, and operational efficiency. A significant trend in software technology is the development of interactive applications by individuals who are not professional developers but possess expertise in other fields, working towards goals facilitated by computational tools. According to statistics from global market insights, the end-user computing (EUC) market reached USD 10.3 billion in 2022 and is expected to grow at a compound annual growth rate (CAGR) of 11% from 2023 to 2032 [1]. The ongoing digital transformation across various sectors has increased the demand for end-user computing solutions. Additionally, research indicates that spreadsheets can be viewed

as a form of code, with spreadsheet users serving as key examples of end-user programmers [2]. Excel spreadsheets are among the most widely used end-user programming environments today, and their significance in the business sector continues to grow [3]. Recently, Microsoft's announcement of integrating Python into Excel marks a noteworthy advancement that aligns with the trend of enhancing data analysis capabilities within familiar software platforms. This integration has the potential to simplify various tasks by utilizing Python's powerful programming features within Excel's intuitive interface. The synergy of Python and Excel creates new opportunities for users to conduct complex data analyses, perform statistical calculations, and even build predictive models seamlessly. However, when evaluating the integration of Python into Excel 365 within Microsoft's Cloud architecture, certain limitations must be considered. A major challenge is latency, which refers to delays in data processing caused by the time required for information to travel between the client and the cloud server [4]. Additionally, the cloud approach does not always guarantee improved performance; in some instances, response time measurements indicate that cloud database performance can be inferior to traditional systems [5]. This paper explores the drawbacks of integrating Python in Excel through cloud-based solutions, particularly focusing on latency issues that can occur due to data transfer between the client and server. We propose a standalone desktop application as an alternative solution. This method employs Python-based user-defined functions (UDFs) created using Excel-DNA and IronPython [6]. By maintaining functionalities locally, our prototype seeks to address potential latency problems and provide a more seamless user experience for data analysis in Excel. User-defined functions in Excel are custom functions crafted by users to perform specific calculations. These functions can be implemented in various ways, such as through add-ins that add new functions based on particular statistical distributions or through sheet-defined functions that enable users to create functions directly within their Excel sheets [7]. Additionally, this prototype is built on the architecture of visual studio tools for office (VSTO) add-ins. These add-ins can monitor activities within the office environment and respond to user actions, such as clicking a button that was added through the add-in. A consistent methodology was employed throughout the study to create a managed code assembly that is loaded by a Microsoft Office application [8]. Once the assembly is loaded, the VSTO add-in can react to events generated within the application. It can also interact with the object model to automate tasks and enhance the application's capabilities, utilizing any classes from the NET framework. The assembly communicates with the application's COM components via its primary interop assembly. This functionality makes VSTO add-ins valuable tools that expand what users can achieve with standard Office applications.

The paper utilized IronPython and Excel-DNA open-source software to meet the research objectives. Excel-DNA facilitates the creation of custom Python-based user-defined functions that integrate smoothly with Excel's calculation engine, while IronPython allows these Python-based functions to run directly within Excel. The primary aim of the prototype is to enhance spreadsheet capabilities through Python-based user-defined functions. In this study, an experiment was conducted to develop a proposed prototype for integrating Python's features into Excel via standalone desktop Python-based UDFs. This method addresses potential latency challenges associated with cloud-based solutions. The remainder of the paper is structured as follows: section 2 literature review and previous works and technologies utilized, section 3 details the methodology of the prototype and describes how the experiment was conducted, and section 4 discusses and analyzes the results of the experiments.

2. LITERATURE REVIEW

Prior studies investigate adapting Excel spreadsheets as a programming environment. Spreadsheets are widely recognized as a favored tool for end-user programming languages [9]. They are frequently employed for tasks such as data organization, the creation of custom functionalities, and even educational purposes [10]. Although spreadsheets are versatile and user-friendly applications, this study explores the use of Excel as a Turing-complete functional programming environment [11]. It emphasizes the potential for new functionalities within Excel and how these advancements could transform the way we develop spreadsheet solutions [12]–[15]. The paper also examines how to move away from the informal end-user practices commonly associated with traditional spreadsheets, advocating for approaches that align more closely with formal programming methods. The overall contribution of this paper is to investigate emerging trends stemming from innovative work within the Excel community and their potential effects on the business and engineering sectors. The goals of this paper are to illustrate that, with Excel 365, it is now feasible to develop solutions for problems that bear little resemblance to previous spreadsheet solutions. Also, it argues that semantically meaningful coding practices could give more reliable results. This research has made significant contributions to expanding Excel's capabilities. These works explore programming within Excel by utilizing programming paradigms to build robust spreadsheet solutions. In the data science field, Python has a wide range of uses. There exist different projects and libraries that aim to help spreadsheet users transfer data into

Python and aid in doing data analysis and statistics [16]. In addition to advantages like convenience and accessibility. There exist different projects and libraries that aim to help spreadsheet users transfer data into Python and aid in performing data analysis and statistics. One of those libraries is Pandas [17] widely used for loading spreadsheets into Python as a form of dataframe. To manipulate spreadsheets in Python, there are a wide range of tools to aid in. For instance, xlutils [18], openpyxl [19], and xlswriter [20] are among the tools available for reading and writing spreadsheets. However, it is important to note that while these tools simplify the process of handling spreadsheet data, they do not offer analysis assistance. Moreover, there are various research methodologies that have sought to integrate Python into Excel, allowing users to call Python functions directly within a spreadsheet setting. One example of this is PyXLL, which allows for the creation of Excel add-ins using Python instead of VBA [21]. This paper explores Python integration by simplifying data analysis by facilitating data transfer and analysis with Python libraries. In the oil industry, there is a study highlighting the effectiveness of IronPython in streamlining analysis through the development of shortcuts and automation. This aligns with the goal of integrating IronPython with Excel, where automation plays a crucial role in improving productivity and efficiency. To integrate IronPython with Excel and spreadsheets, using its capabilities for automation and data analysis, one can draw inspiration from the use of IronPython scripts in the oil industry for reservoir management. By utilizing IronPython scripts, similar to how they were employed in the oil industry for data analysis and workflow optimization, using custom solutions for Excel and spreadsheet integration. These scripts can be tailored to work with Excel, allowing them to manipulate data, carry out calculations, and automatically generate reports [22]. This paper explores, automation with Python and IronPython and enhancing productivity by automating tasks within Excel. This work paves the way for innovative approaches to Excel. It builds upon this foundation by conducting an experimental investigation. The goal is to develop a prototype for integrating Python with Excel through a standalone desktop application, emphasizing a desktop-based solution. This involves using Python-based UDFs to harness Python for custom functionalities within Excel. The integration leverages key technologies such as Excel-DNA and IronPython. Overall, this paper contributes to the ongoing efforts to enhance Excel's capabilities through tailored programming solutions.

3. METHOD

In this section, our prototype presents an end-user engineering approach to empower users with custom Python-based UDFs. The objective is to present a standalone desktop application as an alternative to cloud-based architecture, maintaining local functionalities. This prototype seeks to address potential latency problems and provide a more seamless user experience for data analysis within Excel compared to cloud solutions.

This approach utilizes Python-based UDFs developed with Excel-DNA and IronPython Figure 1 shows our standalone desktop architecture. To achieve this objective, there are elements in consideration. Firstly, C# language programming will be employed to build the core components of the prototype, encompassing the integration layer and any necessary back-end functionality. Additionally, VSTO add-ins will seamlessly integrate Python-based UDFs into the Excel environment. These add-ins will facilitate communication between Python scripts and Excel. Moreover, IronPython Excel spreadsheet integration directly executes Python-based UDFs within the spreadsheet environment. Furthermore, Additionally, it is feasible to develop new worksheet functions that work seamlessly with Excel's calculation framework. Python functions will be created to perform various operations, including mathematical calculations such as sum, minimum, and maximum values within Excel spreadsheets. Importantly, the collection of Python-based UDFs is flexible, allowing for the incorporation of any general-purpose function in the future to tackle new challenges as they arise. Lastly, Excel-DNA will facilitate the integration of .NET components, including C# and IronPython code, into Microsoft Excel, supporting the deployment of the prototype and the management of add-ins. In the following subsections, this paper will outline the underlying architecture of VSTO and explain how this architecture enables communication between the Excel environment and Python-based user-defined functions.

3.1. Underlying architecture

This paper explores the potential of the VSTO add-in architecture. VSTO add-ins serve as a link between software developers and end-users within Microsoft Excel. These plug-ins can monitor actions occurring in the office environment and respond to user interactions, such as clicking a button added via the add-in. Additionally, the VSTO add-in architecture enables smooth communication between the user interface and the UDFs "engine," effectively converting user requests into executable functions within the add-in. This allows users to directly engage with UDFs in the application interface, inputting data into cells and utilizing UDFs for advanced calculations and task automation in their familiar office setting. The VSTO

add-in architecture also provides a solid development framework for building the foundational logic of UDFs. Using visual studio and .NET languages, developers can define UDF functionality while accessing the Office Application's object model to perform tasks that exceed standard capabilities. Moreover, the VSTO add-in framework enhances UDF logic design, allowing customization to precisely address user requirements. Integration within this architecture is facilitated as Excel uses a manifest to load the VSTO add-in assembly [23]. Subsequently, Figure 2 illustrates how the VSTO addins assembly initiates seamless communication with Excel through object model calls, events, and callbacks, ensuring a cohesive and integrated experience for users.

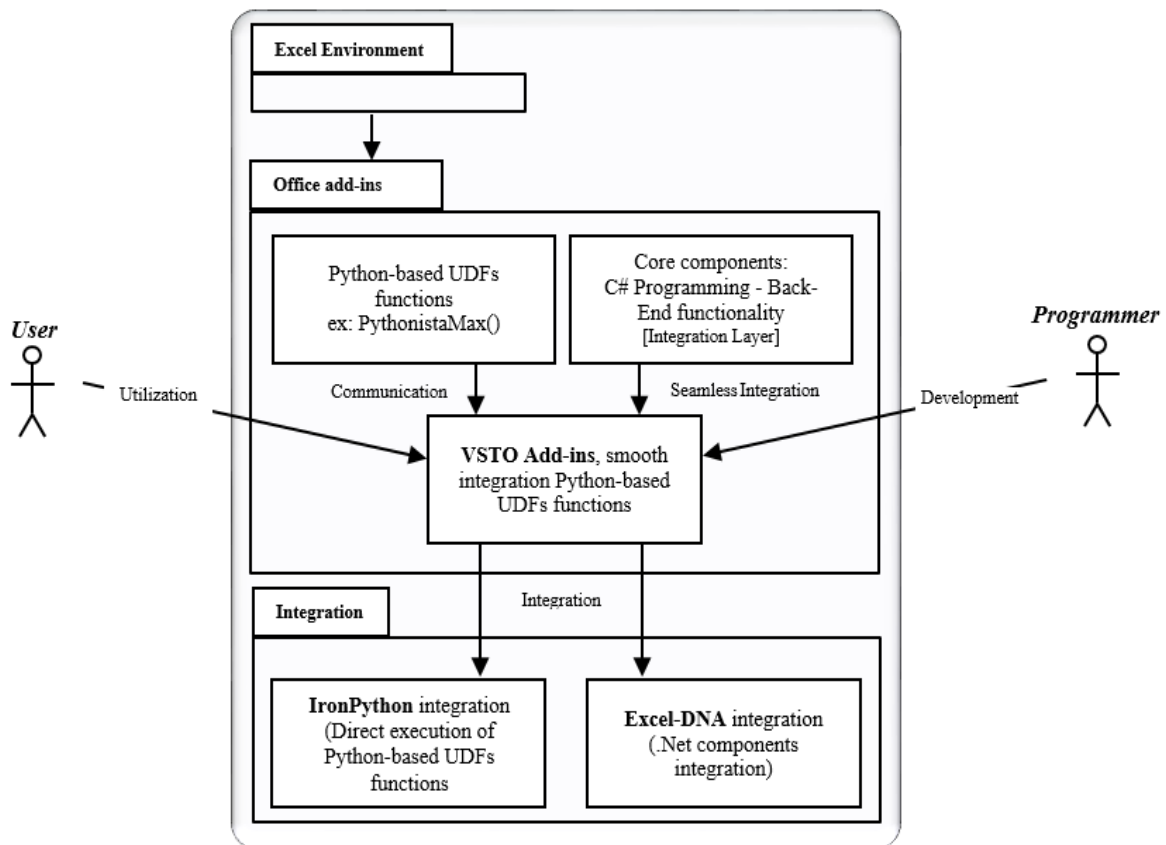


Figure 1. Empowering Excel with Python-based UDFs: An on-premises desktop architecture overview

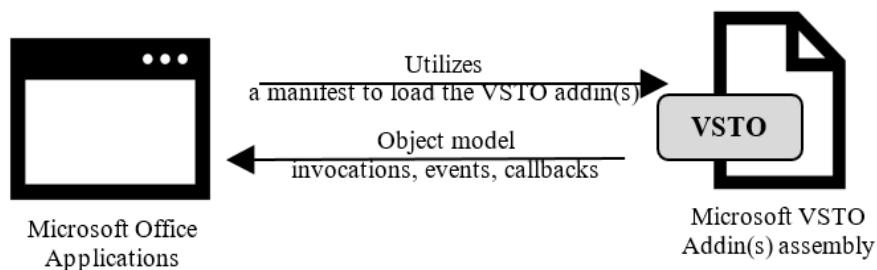


Figure 2. Excel utilizes a manifest to load the VSTO add-in assembly, enabling smooth communication via object model invocations, events, and callbacks

In this context, the integration of VSTO add-in architecture enables a smooth workflow where Python scripts are embedded in Excel as custom Python-based UDFs tailored to meet specific user requirements. End users take advantage of a seamless incorporation of these Python-based UDFs within the familiar Excel interface, enhancing Excel's capabilities and allowing them to undertake more complex tasks.

When an end user launches a Microsoft Office application, it utilizes both the deployment manifest and the application manifest to find and load the latest version of the VSTO add-ins assembly [24]. The VSTO add-in assembly is injected into the application's process space. After it is loaded, the VSTO add-in assembly can communicate with the application via its object model. This interaction enables the VSTO add-in to enhance the application's functionality and offer extra features to users. For instance, a VSTO add-in designed for Microsoft Excel may introduce a new button on the ribbon that enables users to automatically generate a table of contents. Additionally, a VSTO add-in for Excel could offer a custom function that performs calculations for complex mathematical formulas.

3.2. The experiment setup

This revised section details the experimental setup, including the necessary references required within the C# project to integrate Python-based UDFs with Excel using visual studio and VSTO add-ins. i) development environment: visual studio, the development environment remains the same. Ensure visual studio is installed and configured properly; ii) VSTO add-in: project creation, it start by creating a new VSTO add-in project in visual studio then navigate to Office/SharePoint, select Excel, and choose the Excel add-in template. References: a) Microsoft.Office.Interop.Excel, this reference is automatically added when creating a VSTO add-in for Excel. It provides access to the Excel object model; b) IronPython, use NuGet package manager to search for and install the "IronPython" package. This will add the necessary references for interacting with the IronPython interpreter; c) Excel-DNA is similarly, search for and install the "Excel-DNA" package. This will provide the required references for integrating .NET components into Excel; iii) Python interpreter: IronPython, the IronPython package installed in step 2 ensures that the Python interpreter is embedded within the VSTO add-in; iv) Excel-DNA: NuGet Package, the Excel-DNA package installed in step 2 provides the necessary infrastructure for integrating .NET components into Excel; v) Excel Spreadsheet: no additional references required, the Excel spreadsheet is the end-user interface and does not require any specific references within the VSTO add-in.

The following steps outline the flow of execution when a user calls a Python-based UDF within Excel as illustrated in Figure 3: i) user calls a Python-based UDF, the user enters the Python-based UDF name and specifies the required parameters within an Excel cell; ii) VSTO add-in receives call, the VSTO add-in intercepts the function call and receives the necessary data from the Excel spreadsheet; iii) data passed to Python interpreter, the VSTO add-in passes the received data to the IronPython interpreter for execution; iv) Python script execution, the IronPython interpreter interprets and executes the Python script, performing the specified calculations or operations; v) results returned to VSTO add-in, the results of the Python script execution are returned to the VSTO add-in; and vi) results displayed in Excel, the VSTO add-in displays the results within the Excel spreadsheet, making them available for further analysis or use.

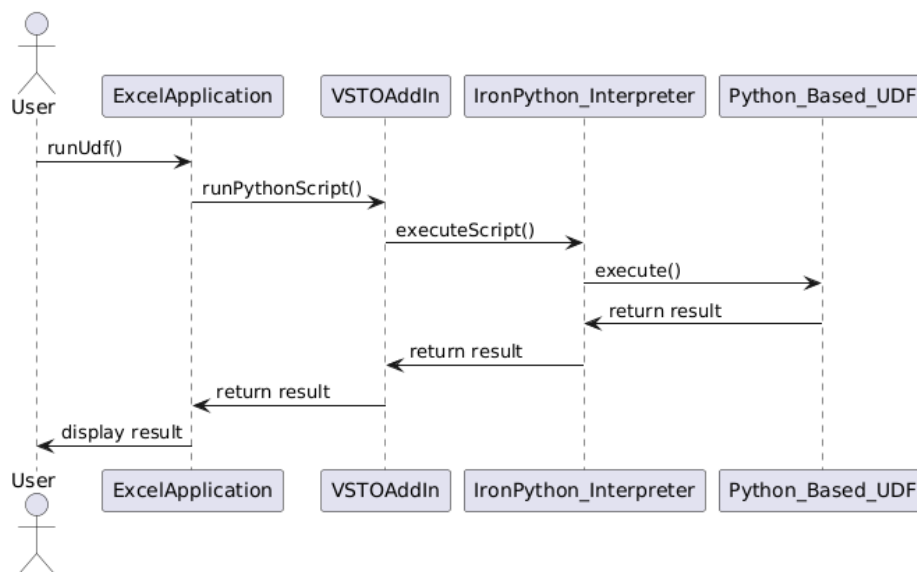


Figure 3. The sequence diagram outlines the flow of execution when a user calls a Python-based UDF within Excel

3.3. Development

In this context, we created the Pythonista set of functions, which includes functions such as PythonistaAverage(), PythonistaMax(), and PythonistaMIN(). As a prototype, we will focus on the PythonistaMax(range) function. This function computes the maximum value from specified ranges of cell values within an Excel environment established and integrated by Excel-DNA. Additionally, it allows for the creation of new worksheet functions that work seamlessly with Excel's calculation model [25]. The interaction within the described system involves a series of critical steps for executing a Python-based UDF in Excel. Initially, during the initialization phase, a MaxFunctions object is instantiated. Within its constructor, the IronPython interpreter is utilized to create a Python engine instance (referred to as 'engine') and a scope instance (denoted as 'scope') through the methods Python.CreateEngine() and engine.CreateScope(), respectively. Subsequently, when the PythonistaMax function is called, a new MaxFunctions object (designated as 'python_function') is instantiated. The range data provided by the user is assigned to the previously established Python scope. The system then executes the script file named calculate_range_max.py utilizing the engine alongside its corresponding scope. During this execution, it retrieves the calculate_max function from within the Python scope. The calculate_max function performs several operations: it scans for non-empty cells within the specified range, computes the maximum value among those cells, and subsequently returns this computed result. To further elucidate this process, an algorithm is presented in listing 1, which outlines the conceptual framework and workflow involved in integrating this Python-based UDF for calculating the maximum value of a specified range in Excel.

Algorithm 1. Integrates a Python-based user defined function in Excel

Require: Calculate the maximum value of a selected range of cells in Excel sheet.

Input:

range: A string representing the Excel range of cells (e.g., "A1:B5").

Output:

max_value: The maximum value found within the specified range of cells.

error_message: A string message indicating an error.

Class MaxFunctions:

1. Declare engine as ScriptEngine

2. Declare scope as ScriptScope

Constructor MaxFunctions():

3. engine <- Create new Python Engine

4. scope <- Create new Scope using engine

Function PythonistaMax(range):

5. Declare python_function as new MaxFunctions object

6. Set range variable in Python scope

7. Declare script File as path to script file

8. Execute script File using engine within scope

9. Declare calculate_max as function from scope

10. Return result of calculate_max function

End Class

Python script calculates_range_max.py:

Function calculate_max(cells):

1. If cells are not empty then

2. Calculate max value of cells

3. Return maximum value

4. Else

5. Return "Cells are empty, cannot calculate maximum value."

End Class

Furthermore, when Excel invokes RequestComAddInAutomationService, a new instance of the AverageFunctions class generated and returned. During add-in initialization, the InternalStartup method registers event handlers ThisAddIn_Startup and ThisAddIn_Shutdown. Overall, this integration allows users to call the Python-based UDF directly within Excel, leveraging Python's functionalities for data analysis within the familiar Excel interface.

4. RESULTS AND DISCUSSION

With the methodology and experiments established, we now discuss the debugging process employed to verify the accuracy and reliability of the implemented prototype of the Pythonista function set. To begin, users should start Microsoft Excel and run the PythonistaMax(range), a Python-based UDF. Step-by-step execution: i) select a new empty cell (e.g., C2); ii) type=PythonistaMax() to invoke the function, similar to standard Excel functions; and iii) specify the range by entering=PythonistaMax(A1:B3) as shown in Figure 4, then press enter to calculate the values.

This testing process validated the successful implementation of Pythonista functions as UDFs within the Excel prototype. Users can directly invoke the PythonistaMax(range) function in their spreadsheets.

By entering the name of the Python-based UDF and selecting a desired range in a cell (e.g., C2), users can trigger the execution of the corresponding Python script. The VSTO add-in architecture facilitates this interaction by directing data to the Python function within the add-in and returning calculated results to the user's spreadsheet. For example, it displays the calculated maximum value directly in Excel. The experiments confirm that we adhered to a consistent methodology throughout this study. Initially, we developed a managed code assembly that integrates with Microsoft Office applications. Once loaded, the VSTO add-in actively responds to events generated within Excel.

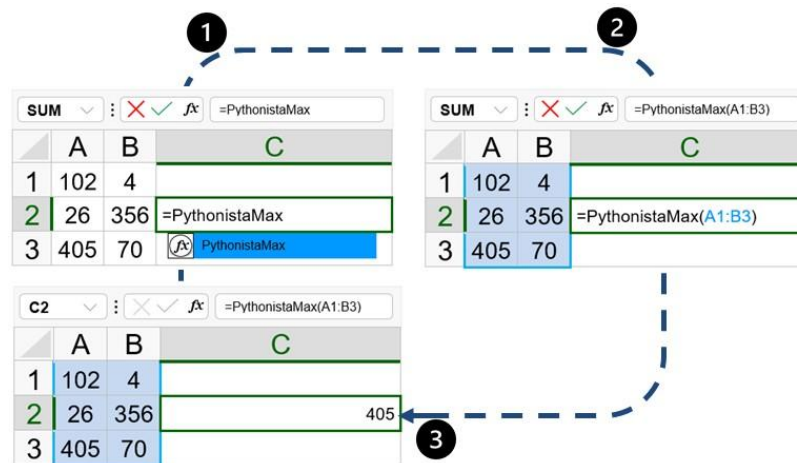


Figure 4. A python-based UDF appears similar to any typical Excel function

This add-in utilizes the object model to automate and enhance application features, with full access to .NET Framework classes for further customization. It interacts effectively with COM components, enabling calls to Python-based UDFs while directing data to custom functions that operate using Python scripts. As such, VSTO add-ins are essential tools for enhancing capabilities within Office Applications. This paper presents a novel method for integrating Python-based UDFs into Excel, addressing latency issues associated with cloud-based integrations. The C# and VSTO-built standalone desktop application allows seamless interaction between Python scripts and Excel, enabling users to perform operations such as data analysis and visualization directly within spreadsheets. The findings confirm the efficacy of this methodology, highlighting how VSTO add-ins can transform standard Office applications into powerful tools. This paper advances end-user engineering and paves the way for future studies on expanding available Pythonista functions and examining user experiences related to data analysis approaches within Excel. While demonstrating the feasibility of integrating Python UDFs in Excel, further research is necessary to evaluate performance impacts on large-scale script execution and scalability for complex datasets. Future directions may include expanding available Pythonista functions for data visualization and machine learning, as well as developing platforms for custom function creation. In conclusion, this paper emphasizes an innovative method for incorporating Python-based UDFs into Excel, empowering users with enhanced data analysis capabilities while setting a foundation for ongoing exploration in this area.

5. CONCLUSION




In conclusion, this paper introduces an innovative method for improving spreadsheet functionalities by integrating python-based UDFs within the Excel environment, successfully tackling the latency challenges linked to cloud-based Python integration. This paper demonstrated a standalone desktop application, developed with C# programming and VSTO add-ins, facilitates smooth interaction between Python scripts and Excel, allowing users to execute a variety of tasks directly within their spreadsheets. The successful implementation of Pythonista functions as python-based UDFs within the Excel prototype, as confirmed by our testing process, demonstrates the practical application of this approach. Users can now call the `PythonistaMax(range)` function directly in their spreadsheets, similar to any typical Excel function, revolutionizing the way they interact with data analysis. Moreover, this paper experiments confirm the efficacy of the methodology, demonstrating how VSTO add-ins can convert standard Office applications into robust tools. This research marks an important advancement in end-user engineering, setting the groundwork

for future studies to investigate the expansion of available Pythonista functions and to examine user experiences with this novel method of data analysis in Excel. To further enhance the capabilities of this approach, future research could focus on expanding Pythonista functions by implementing a wider range of Python functions as python-based UDFs. This would include functions for data visualization, machine learning, and statistical analysis. Additionally, custom function development could be facilitated by providing a platform or framework that allows users to create their own custom python functions as UDFs, tailored to specific needs and domains.




REFERENCES

- [1] Global Market Insights, "End-user-computing-EUC-market," 2023. [Online]. Available: <https://www.gminsights.com/industry-analysis/end-user-computing-euc-market>. (accessed Mar. 04, 2024).
- [2] J. Borghouts, A. D. Gordon, A. Sarkar, and N. Toronto, "End-user probabilistic programming," pp. 3–24, 2019, doi: 10.1007/978-3-030-30281-8_1.
- [3] M. Tallis, R. Waltzman, and R. Blazer, "Adding deductive logic to a COTS spreadsheet," *Knowledge Engineering Review*, vol. 22, no. 3, pp. 255–268, 2007, doi: 10.1017/S0269888907001166.
- [4] Microsoft, "Announcing python in excel combining the power of python and the flexibility of Excel," *Techcommunity.Microsoft.Com*, 2023. [Online]. Available: <https://techcommunity.microsoft.com/t5/excel-blog/announcing-python-in-excel-combining-the-power-of-python-and-the/ba-p/3893439-combining-the-power-of-python-and-the/ba-p/3893439>. (accessed: Mar. 05, 2024).
- [5] R. Gyorodi, M. I. Pavel, C. Gyorodi, and D. Zmaranda, "Performance of OnPrem versus azure SQL server: a case study," *IEEE Access*, vol. 7, pp. 15894–15902, 2019, doi: 10.1109/ACCESS.2019.2893333.
- [6] A. Harris, "Introduction to IronPython," *Pro IronPython*, pp. 1–14, 2009, doi: 10.1007/978-1-4302-1963-7_1.
- [7] T. Turk, "SDFunc: modular spreadsheet design with sheet-defined functions in Microsoft Excel," *Software - Practice and Experience*, vol. 52, no. 2, pp. 415–426, 2022, doi: 10.1002/spe.3027.
- [8] Microsoft, "Architecture of VSTO add-ins," 2023. [Online]. Available: <https://learn.microsoft.com/en-us/visualstudio/vsto/architecture-of-vsto-add-ins?view=vs-2022> (accessed Mar. 03, 2024).
- [9] C. Chambers, M. Erwig, and M. Luckey, "SheetDiff: A tool for identifying changes in spreadsheets," in *Proceedings - 2010 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2010*, 2010, pp. 85–92, doi: 10.1109/VLHCC.2010.21.
- [10] A. A. Bock and F. Biermann, "Puncalc: Task-based parallelism and speculative reevaluation in spreadsheets," *Journal of Supercomputing*, vol. 76, no. 7, pp. 4977–4997, 2020, doi: 10.1007/s11227-019-02823-8.
- [11] F. Hermans, M. Pinzger, and A. Van Deursen, "Supporting professional spreadsheet users by generating leveled dataflow diagrams," in *Proceedings - International Conference on Software Engineering*, 2011, pp. 451–460, doi: 10.1145/1985793.1985855.
- [12] P. Bartholomew, "Excel as a turing-complete functional programming environment," 2023, *arXiv:2309.00115*.
- [13] R. Abraham, M. Burnett, and M. Erwig, "Spreadsheet programming," *Wiley Encyclopedia of Computer Science and Engineering*, pp. 2804–2810, 2009, doi: 10.1002/9780470050118.ecse415.
- [14] M. A. T. T. Mccutchen, J. Borghouts, A. D. Gordon, S. I. M. O. N. Peyton Jones, and A. Sarkar, "Elastic sheet-defined functions: Generalising spreadsheet functions to variable-size input arrays," *Journal of Functional Programming*, vol. 30, 2020, doi: 10.1017/S0956796820000234.
- [15] K. T. Klasson, "QXLA: Adding upper quantiles for the studentized range to excel for multiple comparison procedures," *Journal of Statistical Software*, vol. 85, 2018, doi: 10.18637/jss.v085.c01.
- [16] A. Nassereldine, P. Chen, and J. Xiong, "Excel spreadsheet analyzer," 2022, *arXiv:2211.06333*.
- [17] W McKinney, "Pandas: a foundational Python library for data analysis and statistics," *Python for high performance and scientific computing*, 2011.
- [18] T. B. Elserwy, A. T. N. E.-D. Raslan, T. Ali, and M. H. Gheith, "Building custom spreadsheet functions with Python: end-user software engineering approach," *Journal of Software Engineering and Applications*, vol. 17, no. 5, pp. 246–258, 2024, doi: 10.4236/jsea.2024.175014.
- [19] J. Hunt, "Working with Excel files," in *Advanced Guide to Python 3 Programming*, 2019, pp. 249–255.
- [20] C. R. Harris *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020, doi: 10.1038/s41586-020-2649-2.
- [21] O. Ed-daymouni *et al.*, "Efficient KPIs analysis: Harnessing the power of Excel and VBA programming for data visualization and analysis," in *International Conference on Digital Technologies and Applications*, 2024, pp. 23–31, doi: 10.1007/978-3-031-68675-7_3.
- [22] R. R. Fatra, E. A. Flodin, C. Bawono, M. I. Arshanda, F. Rivano, and D. Rachmanto, "Teasing insight out of reams of data using advanced data visualization and analytics software for improved reservoir management, rokan light oil, Indonesia," 2019, doi: 10.2118/196318-ms.
- [23] K. McGrath and P. Stubbs, *VSTO for mere mortals: a VBA developer's guide to microsoft development using visual studio 2005 tools for office*. Addison-Wesley Professional, 2006.
- [24] L. Wang, J. Chen, C. Zheng, and J. Feng, "Application research on table structure recognition and information extraction in sci-tech academic journals based on visual studio tools for Office technology," *Editing Practice*, vol. 1, 2023, doi: 10.54844/ep.2023.0412.
- [25] DNAKode, "Excel-DNA," 2023. [Online]. Available: <https://excel-dna.net/docs/introduction> (accessed Mar. 15, 2024).




BIOGRAPHIES OF AUTHORS

Tamer Bahgat Elserwy    is a seasoned professional with over 10 years of experience in the software field and quality management. He holds a master of science degree in software engineering from the Department of Software Engineering, Faculty of Graduate Studies for Statistical Research, Cairo University, Egypt. He has demonstrated his proficiency in developing and implementing web-based applications using C# and Microsoft visual studio .Net. As a Microsoft certified solutions developer (MCSD), he has shown his expertise in designing and developing solutions using Microsoft technologies such as .NET framework and SQL server. He excels in software development with agile experience (scrum master) and quality assurance (certified auditor). He can be contacted at email: tamer.elserwy@gmail.com.



Tarek Aly    is a respected member of the Computer Science Department at the Faculty of Graduate Studies for Statistical Research, Cairo University, located in Giza, Egypt. He possesses a broad range of skills and expertise in various areas of computer science and information technology. His areas of proficiency include knowledge management, user experience, prototyping, interaction, information management, IT project management, information system management, design thinking, information technology, and designing. Throughout his career, he has made significant contributions to his field. He has published 6 research papers, which have been read 1,882 times by scholars and researchers worldwide. His contributions have significantly advanced the field of computer science and continue to influence researchers and practitioners alike. He can be contacted at email: Tarekmmmt@pg.cu.edu.eg.



Basma E. El-Demerdash    is an associate professor at the Department of Operations Research and Management, Faculty of Graduate Studies for Statistical Research (FGSSR), Cairo University. She graduated with a B.Sc. in Operations Research and Decision Support from the Faculty of Computers and Artificial Intelligence, Cairo University, in 2007. Her outstanding academic performance earned her a position as a teaching assistant in the department. She then went on to receive her M.Sc. degree from the same department in 2011. Her thesis focused on evaluating higher education systems in Egypt using data envelopment analysis. Finally, she went on to receive her Ph.D. degree from the same department in 2019. Her thesis focused on developing a data envelopment analysis model under different uncertainty environments with different orientation types and return-to-scale types. Her academic career began as a teaching assistant in 2007. Since then, she has accumulated 17 years of teaching experience at the Faculty of Computers and Artificial Intelligence, Cairo University. She can be contacted at basma.ezzat@cu.edu.eg.