


# Building Custom Spreadsheet Functions with Python: End-User Software Engineering Approach

Tamer Bahgat Elserwy<sup>1</sup> , Atef Tayh Nour El-Din Raslan<sup>2</sup>, Tarek Ali<sup>1</sup>, Mervat H. Gheith<sup>1</sup>

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

<sup>2</sup>Department of Information Systems, Higher Institute of Advanced Studies, Giza, Egypt

Email: Tamer.elserwy@gmail.com, Dr.Atef.Raslan@gmail.com, Tarekmmmt@pg.cu.edu.eg, Mervat\_gheith@yahoo.com

**How to cite this paper:** Elserwy, T.B., Raslan, A.T.N.E.-D., Ali, T. and Gheith, M.H. (2024) Building Custom Spreadsheet Functions with Python: End-User Software Engineering Approach. *Journal of Software Engineering and Applications*, 17, 246-258. <https://doi.org/10.4236/jsea.2024.175014>

**Received:** March 28, 2024

**Accepted:** May 25, 2024

**Published:** May 28, 2024

Copyright © 2024 by author(s) and Scientific Research Publishing Inc. This work is licensed under the Creative Commons Attribution International License (CC BY 4.0). <http://creativecommons.org/licenses/by/4.0/>



Open Access

## Abstract

End-user computing empowers non-developers to manage data and applications, enhancing collaboration and efficiency. Spreadsheets, a prime example of end-user programming environments widely used in business for data analysis. However, Excel functionalities have limits compared to dedicated programming languages. This paper addresses this gap by proposing a prototype for integrating Python's capabilities into Excel through on-premises desktop to build custom spreadsheet functions with Python. This approach overcomes potential latency issues associated with cloud-based solutions. This prototype utilizes Excel-DNA and IronPython. Excel-DNA allows creating custom Python functions that seamlessly integrate with Excel's calculation engine. IronPython enables the execution of these Python (CSFs) directly within Excel. C# and VSTO add-ins form the core components, facilitating communication between Python and Excel. This approach empowers users with a potentially open-ended set of Python (CSFs) for tasks like mathematical calculations, statistical analysis, and even predictive modeling, all within the familiar Excel interface. This prototype demonstrates smooth integration, allowing users to call Python (CSFs) just like standard Excel functions. This research contributes to enhancing spreadsheet capabilities for end-user programmers by leveraging Python's power within Excel. Future research could explore expanding data analysis capabilities by expanding the (CSFs) functions for complex calculations, statistical analysis, data manipulation, and even external library integration. The possibility of integrating machine learning models through the (CSFs) functions within the familiar Excel environment.

## Keywords

End-User Software Engineering, Custom Spreadsheet Functions (CSFs),

---

## Visual Studio Tools Office (VSTO) Add-Ins, Python with Excel Integration, On-Premises Desktop Applications

---

### 1. Introduction

End-user computing offers a centralized and standardized approach to managing applications, devices, and data. This translates to improved collaboration, scalability, and operational efficiency [1]. A significant trend in software technology is the rise of interactive applications built not by professional developers, but by domain experts leveraging computational tools to achieve their goals [2]. The global EUC market is estimated at USD 10.3 billion in 2022 and is projected for an 11% CAGR by 2032, reflecting the growing demand for EUC solutions driven by digital transformation across industries efficiency [1]. Research has shown that spreadsheets are a form of code, with spreadsheet users acting as end-user programmers [2]. Excel, specifically, remains one of the most popular end-user programming environments, with its importance in the business world continuing to rise [3]. Microsoft's recent announcement of Python integration within Excel signifies a major step towards enhanced data analysis capabilities within familiar software environments. This integration has the potential to streamline tasks by leveraging Python's robust programming capabilities within Excel's user-friendly interface [4]. The combination opens doors for users to perform complex data analysis, statistical computations, and even develop predictive models efficiently. However, cloud-based Python integration in Excel faces limitations, particularly concerning latency. Latency refers to the delay in data processing caused by the time it takes for information to travel between the client and the cloud server [5]. This paper addresses the limitations of cloud-based Python integration in Excel, particularly latency issues arising from client-server data transfer. This paper proposes an alternative solution: an on-premises desktop application approach to build custom spreadsheet functions with Python developed with Excel-DNA and IronPython [6]. By keeping functionalities local, to overcome potential latency issues and offer a smoother user experience within Excel. Custom spreadsheet functions (CSFs) are custom functions created by users to perform specific calculations within Excel. They can be implemented through various methods, such as add-ins introducing new functions based on specific statistical distributions, or sheet-defined functions allowing users to define functions directly within Excel sheets [7]. This prototype leverages the architecture of Visual Studio Tools Office (VSTO) add-ins. These add-ins can monitor user activity within Office applications and react to events, such as clicking buttons added by the add-in itself [8]. VSTO add-ins follow a consistent methodology: a managed code assembly is loaded by a Microsoft Office application. Once loaded, the add-in can respond to events raised within the application and call into its object model for automation and extension. Additionally, it can

access any .NET Framework classes and communicates with the application's COM components through the primary interop assembly [8]. This makes VSTO add-ins valuable tools for extending the functionalities of standard Office applications. This paper outlines the limitations of cloud-based Python integration in Excel and proposes an alternative approach using an on-premises desktop application.

This paper is organized through the relevant background information. The next section delves into contemporary works related to this field. To establish prototype infrastructure, the third section explores the existing architecture relevant to the study. Following that, the fourth section provides a detailed overview of the VSTO Add-Ins architecture. As this groundwork lays, the fifth section, methodology and experiment setup, outlines the paper's methods and procedures used in the research. Finally, the sixth section, results and discussions presents and interprets the paper's findings.

## **2. Background**

This section discusses related research areas relevant to the development of custom spreadsheet functions (CSFs) in Excel.

### **2.1. End-User Software Engineering**

This field focuses on empowering non-professional programmers to create software [9]. It explores adapting existing software development tools for collaboration between developers and non-programmers [10].

### **2.2. End-User Programming of Spreadsheets**

Spreadsheets are a popular tool for end-user programming languages [11]. They are commonly used for data organization, development of custom functionality, and even education [12], while spreadsheets are flexible and user-friendly applications [13].

### **2.3. Concepts in Spreadsheet Programming**

Spreadsheets consist of cells that can hold numbers, text, or formulas. These formulas can reference other cells to perform calculations automatically. The system for creating and modifying spreadsheets acts as a programming environment. Spreadsheets are typically built on a grid structure, and the underlying model may lack structure, especially for non-programmers who refine their model as they develop the spreadsheet [14].

### **2.4. Custom Spreadsheet Functions**

Custom spreadsheet functions (CSFs) are valuable when standard Excel functions are insufficient. They allow users to work with data, create modular designs, and improve spreadsheet reusability [15]. (CSFs) empower users to extend Excel's capabilities by creating custom functions for specific needs. These func-

tions can automate tasks, perform advanced calculations, and enhance the overall flexibility and efficiency of Excel [16].

### 3. Related Work

Several studies have explored using Excel as a programming environment. Research has investigated using Excel for complex programming tasks, highlighting its potential for more powerful spreadsheet solutions. This suggests a shift towards more formal programming practices within Excel. Other studies demonstrate how Excel 365 allows creating solutions beyond traditional spreadsheets solutions [17]. They advocate for using semantically meaningful code for reliable results, contributing to expanding Excel's capabilities. In the data science field, Python is widely used for data analysis and statistics [18]. Libraries like Pandas [19] help transfer data from spreadsheets into Python for further analysis. Additional tools like, xlutils [20], openpyxl [21], and xlswriter [22] simplify working with spreadsheets in Python. Research has also explored unifying Python with Excel, allowing users to directly call Python functions within Excel. PyXLL [23], for example, enables writing Excel add-ins in Python instead of VBA. This simplifies data analysis by leveraging Python libraries. A study in the oil industry demonstrates the effectiveness of IronPython for streamlining analysis through automation [24]. This aligns with the goal of integrating IronPython with Excel to improve productivity and efficiency. Similar to its use in the oil industry, IronPython scripts can be used to automate tasks within Excel, manipulate data, perform calculations, and generate reports.

This paper focuses on creating a desktop-based solution that integrates Python with Excel using an on-premises desktop application. This allows users to leverage Python for custom functionalities within Excel through Python-based custom spreadsheet functions (CSFs). This research utilizes Excel-DNA and IronPython technologies to achieve this integration. Overall, this work contributes to the ongoing exploration of enhancing Excel's capabilities through building custom spreadsheet functions with Python.

### 4. Underlying Architecture

This paper investigates the potential of the Visual Studio Tools for Office (VSTO) add-in architecture. This powerful functionality is enabled by a robust underlying architecture designed by Microsoft specifically for VSTO Add-ins. VSTO add-ins act as a bridge between software engineers and end-users in the realm of Microsoft Excel. The add-ins can listen to what's happening within the Office environment. For example, the add-in can react to users clicking on buttons they added themselves. The Visual Studio Tools for Office Add-in (VSTO Add-in) architecture also facilitates communication between the user interface and the custom spreadsheet functions (CSFs) engine. It effectively interprets user requests and converts them into executable functions within the add-in. This empowers users to directly interact with these (CSFs) within the application's in-

terface, inputting data into cells and leveraging them for complex calculations and task automation within their familiar Office environment.

However, the VSTO add-in architecture goes beyond a user interface. It provides a robust development framework for constructing the underlying logic of (CSFs). Using Visual Studio and .NET languages, engineers define the functionality of (CSFs), accessing the Office application's object model to perform tasks beyond standard capabilities. Moreover, the VSTO add-in framework enriches the design of the logic for (CSFs), tailoring them precisely to meet users' needs. The VSTO add-in architecture integration is facilitated by Excel utilizing a manifest to load the VSTO add-in assembly. Subsequently, **Figure 1** illustrates how the VSTO add-in assembly initiates integration communication with Excel through object model calls, events, and callbacks, ensuring a harmonious and integrated experience for users [25].

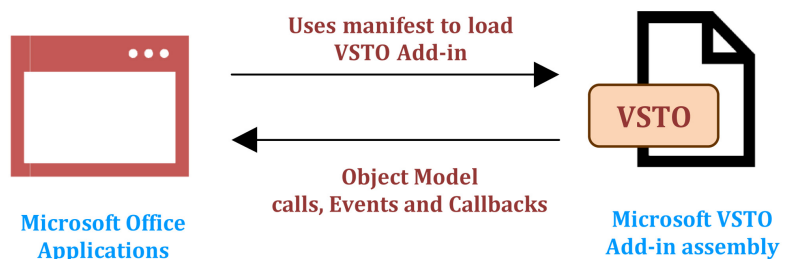
Integrating the VSTO add-in architecture fosters an effortless workflow, enabling users to leverage Python functions within Excel as if they were standard features. Python scripts embedded within Excel as custom (CSFs) can address specific user needs. End users benefit from a smooth integration of these (CSFs) within the familiar Excel interface, extending the capabilities of Excel and empowering them to perform more advanced tasks.

## 5. The VSTO Add-Ins Architecture

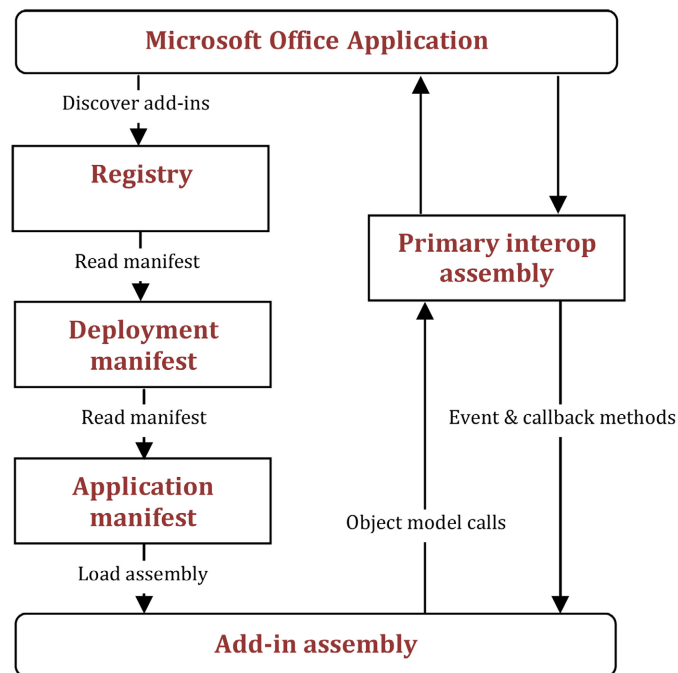
The VSTO Add-Ins, built using Visual Studio Tools for Office, extend the functionality of Microsoft Office applications like Word, Excel, and Outlook. This architecture empowers developers to create powerful customizations that blend effortlessly with the familiar Office environment. In the case of starting Microsoft office application by the end user, the application uses the deployment manifest and the application manifest to locate and load the most current version of the VSTO Add-In assembly as depicted in **Figure 2** which illustrates the basic architecture of these VSTO Add-ins.

## 6. Methodology

This paper adopts a structured methodology. The initial phase involves setting up the development environment. Next, custom spreadsheet functions (CSFs)



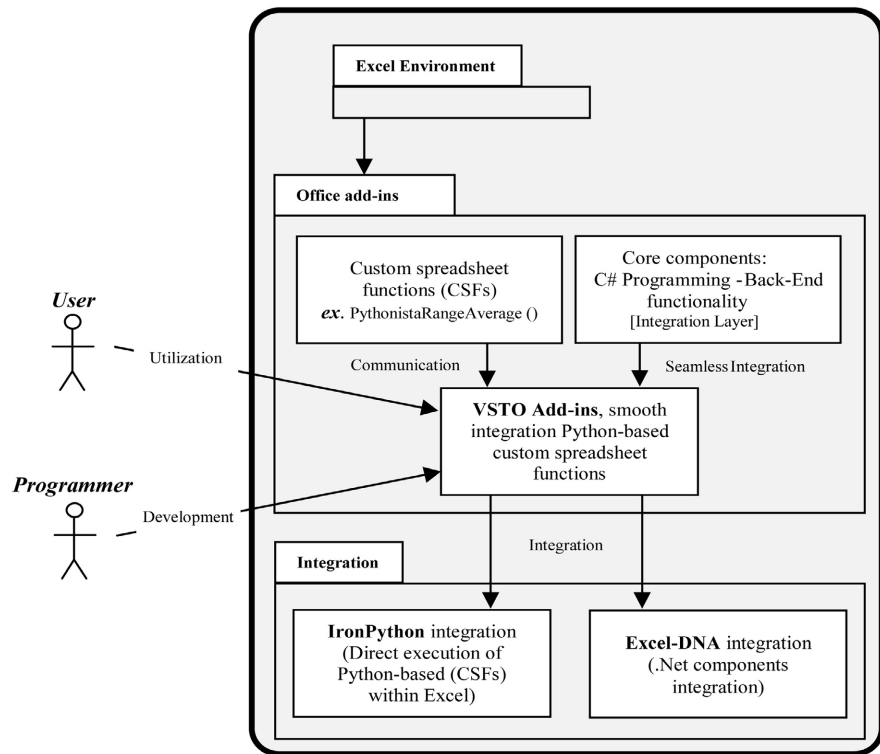
**Figure 1.** Excel utilizes a manifest to load the VSTO add-in assembly, enabling integration communication through object model calls, events, and callbacks (Source: Microsoft, 2023).



**Figure 2.** The basic VSTO add-ins architecture (Source: Microsoft, 2023).

are developed using C# programming language, incorporating Python for enhanced capabilities. Following development, a rigorous testing process is undertaken to ensure the effectiveness and reliability of the CSFs. Finally, the results section presents and interprets the paper findings, providing valuable insights into the performance of the custom spreadsheet functions. This paper investigates an alternative solution, an on-premises desktop application approach to build custom spreadsheet functions (CSFs) in Excel environment. This approach targets to reduce potential latency issues and provide a smoother user experience compared to cloud-based solutions. This paper presents a prototype that leverages a combination of technologies based on Visual Studio Tools for Office architecture. The prototype discuss foundational technologies to build such as Python-based functions, C# programming forms the foundation, building the core components and backend functionalities as shown in **Figure 3**. Microsoft's VSTO Add-ins bridge the gap between Python scripts and the Excel environment, enabling smooth communication. IronPython Excel Spreadsheet Integration allows for direct execution of Python-based (CSFs) within Excel spreadsheets. These Python-based (CSFs) integrate with Excel's calculation model and can handle various operations, with the potential for future expansion.

In a development context, a set of Python-based functions developed that integrate with Excel's calculations by combining these technologies. These Python-based functions handle various operations such as mathematical calculations (e.g., sum, average, and maximum) within Excel spreadsheets. Crucially, the set of Python-based (CSFs) is open-ended. This means that it is possible to use any general-purpose function in the future to address recent problems that



**Figure 3.** Build custom spreadsheet functions with python: An architectural overview. (Source: Author, 2024).

emerge. Finally, Excel-DNA will simplify the integration of .NET components, including C# and IronPython code, into Microsoft Excel, aiding in deploying the prototype and managing add-ins.

### 6.1. Experiment Setup

This subsection outlines the steps to establish a development environment for building custom Python-based spreadsheet functions with unique functionalities. It emphasizes the importance of experimental setup to ensure that the necessary Python interpreter and dependencies are installed. To experiment setup, follow these steps:

- 1) Start Visual Studio and initiate a new project.
- 2) Navigate to the “Create a new project” section and select “Visual C#” as a development language.
- 3) Delve into the “Office/SharePoint” category and choose “Excel” followed by the “Excel Add-in” project template. This sets up the project specifically for crafting Excel add-ins.
- 4) To empower Python functionality within add-ins, you need to reference the IronPython library.

- 5) Head over to the “Manage NuGet Packages” option within Visual Studio and search for IronPython packages. Install the appropriate version that aligns with the project requirements.
- 6) Similar to IronPython, Excel-DNA is essential for integrating .NET components into Excel add-in.
- 7) Access the “Manage NuGet Packages” window again and search for Excel-DNA. Locate and install the compatible version for the project. By following these steps, a well-equipped environment



ready to tackle building custom Python functions within Excel environment.

## 6.2. Coding

In this direction, we developed a set of Python-based custom spreadsheet functions (CSFs) to demonstrate this integration. For example, the `PythonistaRangeAverage()` function calculates the average of selected cell ranges within an Excel environment. The `AVERAGE` function calculates the average (arithmetic mean) of supplied numbers. `AVERAGE` can handle arguments such as cell references, ranges and numbers. This capability emphasizes that new worksheet functions can be effortlessly incorporated into Excel's existing calculation model [26]. To understanding the interaction process, the interaction between Python and Excel in this system follows these key steps.

### 6.2.1. Initialization

During startup, an Average Functions object is created. Within its constructor, the system utilizes the IronPython interpreter to create a Python engine instance (engine) and a scope instance (scope) using the Python. `Create Engine()` and `engine. Create Scope()` methods, respectively as shown in pseudocode **Listing 1**.

```
Class AverageFunctions:
  1.Declare engine as ScriptEngine
  2.Declare scope as ScriptScope
  Constructor AverageFunctions():
    3.engine <- Create new Python Engine
    4.scope <- Create new Scope using engine
  Function PythonistaRangeAverage(range):
    5.Declare pyFunction as new AverageFunctions 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_average as function from scope
    10.Return result of calculate_average function
End Class
```

**Listing 1.** Pseudocode shows flow of average calculation using Python scripting within the Average Functions class.

### 6.2.2. Calling the Python Function

When calling the Python-based custom spreadsheet `PythonistaRangeAverage()` function, an Average Functions object (pyFunctions) is created, and the provided range data is set within the Python scope. The script file `calculates_range_average.py` is then executed using the engine and associated scope. This script fetches the `calculate_average` function from the Python scope, see pseudocode **Listing 2**. This function checks for non-empty cells, computes the average, and returns the result.

```
Python script calculates_range_average.py:
Function calculate_average(cells):
  1.If cells are not empty then
  2. Calculate average of cells
  3. Return average
  4.Else
  5. Return "Cells is empty, cannot calculate average."
End Class
```

**Listing 2.** Pseudocode shows `calculate_average` function in `calculates_range_average.py` script for calculating cell averages.



In closing, this subsection showcased the integration of Python with Excel through a sample set of Python functions. The `PythonistaRangeAverage()` function serves as a concrete example of how new functionalities can be effortlessly incorporated into Excel’s existing calculation capabilities. The provided pseudo-code as shown in **Listing 1** and **Listing 2** outline the key steps involved in this interaction, including initialization, and calling the Python function. This approach paves the way for developers to extend Excel’s functionalities using Python such as enhancing its potential for data analysis and manipulation.

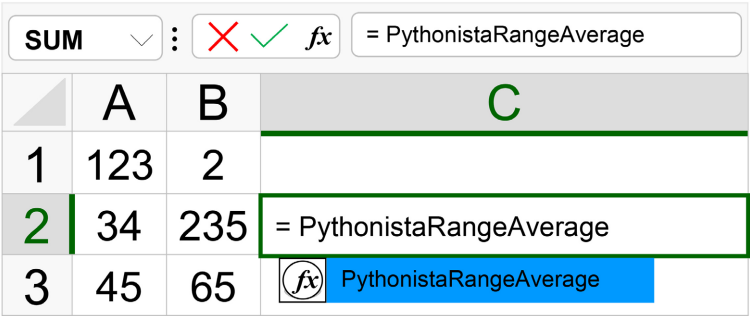
6.3. Testing the Python-Based Functions

This section discusses the testing process used to ensure the accuracy and robustness of the implemented custom spreadsheet function set. The testing process is critical for identifying and resolving issues that could impede the functionality of the (CSFs). However, development can also encounter challenges related to debugging. A common hurdle is version compatibility. Incompatibilities might arise between Excel-DNA, IronPython, and the .NET Framework versions used in the project. To overcome this, meticulous management of project dependencies is essential. Ensure all components, including Python, IronPython, and the .NET Framework, are fully compatible with each other. It might also be necessary to test with different version combinations to guarantee overall integration.

Testing the Custom Spreadsheet Functions (CSFs)

This subsection details how to utilize the custom spreadsheet functions (CSFs) within the Excel environment and here are steps to use it:

- 1) Start Excel and run the function by opening Microsoft Excel. Within the spreadsheet, type = `PythonistaRangeAverage()` in a new empty cell (e.g., C2) or inside the formula bar. This function behaves just like standard Excel functions, as shown in **Figure 4**.
- 2) Select the desired range of cells to calculate the average, as illustrated in **Figure 5**.
- 3) Press Enter to initiate the calculation. The result will be displayed in the cell where the formula was entered, as shown in **Figure 6**.



**Figure 4.** Use formula bar to use Python-based `PythonisatRangeAverage()` inside excel.

SUM <span>⌵</span> : <span>✗</span> <span>✓</span> <i>fx</i> = PythonistaRangeAverage (A1:B3)			
	A	B	C
1	123	2	
2	34	235	= PythonistaRangeAverage (A1:B3)
3	45	65	

**Figure 5.** Select range of cells to apply custom spreadsheet function PythonistaRangeAverage().

C2 <span>⌵</span> : <span>✗</span> <span>✓</span> <i>fx</i> = PythonistaRangeAverage (A1:B3)			
	A	B	C
1	123	2	
2	34	235	62.5
3	45	65	

**Figure 6.** The result of execution Python script inside excel environment.

In the previous example, the VSTO add-in architecture integrates flawlessly. It executes the (CSFs) call PythonistaRangeAverage() in the background. It then transmits the data from the selected range to the custom function implemented within the add-in.

In conclusion, the experiments verify that the PythonistaRangeAverage() function performs the average calculations and returns the result back to Excel environment, where it's displayed in the Excel spreadsheet.

## 7. Results and Discussions

As a result, the testing process confirmed the implementation Python-based custom spreadsheet functions (CSFs) within the Excel environment. Users can directly call (CSFs) functions in their spreadsheets, the prototype Pythonista RangeAverage() function just like any standard Excel function. Moreover, the intuitive integration and functionality allow users to initiate Python script execution simply by entering the (CSF) name and selecting the desired range within a cell to calculate the average. The VSTO Add-in architecture efficiently handles this call, passing the data to the Python function within the add-in, and promptly displays the calculated back in the user's spreadsheet. In this domain, future research could explore expanding the available (CSFs) functions and investigate more complex calculations such as explore enabling (CSFs) functions to handle more complex data analysis tasks. This could involve statistical functions, data manipulation tools, or integration with external libraries. Further-

more, investigate the possibility of integrating machine learning models written in Python into the Excel environment through (CSFs) functions.

## 8. Conclusion

In conclusion, this paper introduces a novel solution on-premises desktop application approach building custom spreadsheet functions with Python to overcome latency issues associated with cloud-based Python integration in Excel. It utilizes C# programming and VSTO add-ins to seamlessly connect Python scripts with Excel, allowing users to perform calculations directly within spreadsheets as any standard Excel function. This effective implementation empowers users and validates the effectiveness of the methodology, demonstrating how VSTO add-ins can transform Excel into a powerful data analysis tool. Future research could explore expanding data analysis capabilities by expanding (CSFs) functions for complex calculations, statistical analysis, data manipulation, and even external library integration. The possibility of integrating machine learning models through custom spreadsheet functions opens doors for even more powerful data analysis techniques within the familiar Excel environment.

## Acknowledgements

I would like to be grateful to our teaching staff at the department of software engineering, faculty of graduate studies for statistical research (FGSSR), Cairo University for their guidance and support.

## Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

## References

- [1] Global Market Insights (2023) End-User-Computing-EUC-Market. <https://www.gminsights.com/industry-analysis/end-user-computing-euc-market>
- [2] Borghouts, J., Gordon, A.D., Sarkar, A. and Toronto, N. (2019) End-User Probabilistic Programming. *Quantitative Evaluation of Systems*, Glasgow, 10-12 September 2019, 3-24. [https://doi.org/10.1007/978-3-030-30281-8\\_1](https://doi.org/10.1007/978-3-030-30281-8_1)
- [3] Tallis, M., Waltzman, R. and Blazer, R. (2007) Adding Deductive Logic to a Cots Spreadsheet. *The Knowledge Engineering Review*, **22**, 255-268. <https://doi.org/10.1017/s0269888907001166>
- [4] Microsoft (2023) Announcing Python in Excel: Combining the Power of Python and the Flexibility of Excel. <https://techcommunity.microsoft.com/t5/excel-blog/announcing-python-in-excel-combining-the-power-of-python-and-the/ba-p/3893439>
- [5] Györödi, R., Pavel, M.I., Györödi, C. and Zmaranda, D. (2019) Performance of OnPrem versus Azure SQL Server: A Case Study. *IEEE Access*, **7**, 15894-15902. <https://doi.org/10.1109/ACCESS.2019.2893333>
- [6] NET Foundation (2023) Ironpython. <https://ironpython.net/>

- [7] Turk, T. (2021) SDFunc: Modular Spreadsheet Design with Sheet-Defined Functions in Microsoft Excel. *Journal of Software: Practice and Experience*, **52**, 415-426. <https://doi.org/10.1002/spe.3027>
- [8] Microsoft (2023) Architecture of VSTO Add-Ins. <https://learn.microsoft.com/en-us/visualstudio/vsto/architecture-of-vsto-add-ins?view=vs-2022>
- [9] Dwivedi, V. (2022) Halo: A Framework for End-User. Carnegie Mellon University, Pittsburgh.
- [10] Ko, A.J., Abraham, R., Beckwith, L., Blackwell, A., Burnett, M., Erwig, M., Scaffidi, C., Lawrance, J., Lieberman, H., Myers, B., Rosson, M.B., Rothermel, G., Shaw, M. and Wiedenbeck, S. (2011) The State of the Art in End-User Software Engineering. *ACM Computing Surveys*, **43**, Article No. 21. <https://doi.org/10.1145/1922649.1922658>
- [11] Chambers, C., Erwig, M. and Luckey, M. (2010) Sheetdiff: A Tool for Identifying Changes in Spreadsheets. 2010 *IEEE Symposium on Visual Languages and Human-Centric Computing*, Leganes, 21-25 September 2010, 85-92. <https://doi.org/10.1109/VLHCC.2010.21>
- [12] Bock, A. and Biermann, F. (2019) Puncalc: Task-Based Parallelism and Speculative Reevaluation in Spreadsheets. *The Journal of Supercomputing*, **76**, 4977-4997. <https://doi.org/10.1007/s11227-019-02823-8>
- [13] Hermans, F., Pinzger, M. and Deursen, A. (2011) Supporting Professional Spreadsheet Users by Generating Leveled Dataflow Diagrams. *Proceedings of the 33rd International Conference on Software Engineering*, Honolulu, 21-28 May 2011, 451-460. <https://doi.org/10.1145/1985793.1985855>
- [14] Abraham, R., Burnett, M. and Erwig, M. (2009) Spreadsheet Programming. In: Wah, B.W., Ed., *Wiley Encyclopedia of Computer Science and Engineering*, John Wiley & Sons, Hoboken. <https://doi.org/10.1002/9780470050118.ecse415>
- [15] McCutchen, M., Borghouts, J., Gordon, A., Jones, S. and Sarkar, A. (2020) Elastic Sheet-Defined Functions: Generalising Spreadsheet Functions to Variable-Size Input Arrays. *Journal of Functional Programming*, **30**, e26. <https://doi.org/10.1017/S0956796820000234>
- [16] Klasson, K.T. (2018) QXLA: Adding Upper Quantiles for the Studentized Range to Excel for Multiple Comparison Procedures. *Journal of Statistical Software*, **85**, 1-9. <https://doi.org/10.18637/jss.v085.c01>
- [17] Bartholomew, P. (2023) Excel as a Turing-Complete Functional Programming Environment. arXiv:2309.00115. <https://doi.org/10.48550/arXiv.2309.00115>
- [18] Nassereldine, A., Chen, P. and Xiong, J. (2022) Excel Spreadsheet Analyzer. <https://arxiv.org/abs/2211.06333>
- [19] McKinney, W. (2011) pandas: A Foundational Python Library for Data Analysis and Statistics. <https://docslib.org/doc/4231522/a-foundational-python-library-for-data-analysis-and-statistics>
- [20] Xlutils Documentation. <https://xlutils.readthedocs.io/en/latest/#>
- [21] <https://openpyxl.readthedocs.io/en/stable/tutorial.html>
- [22] <https://xlsxwriter.readthedocs.io/contents.html>
- [23] Introduction to PyXLL. <https://www.pyxll.com/docs/introduction.html>
- [24] Fatra, R., Flodin, E., Bawono, C., Arshanda, M., Rivano, F. and Rachmanto, D. (2020) Teasing Insight Out of Reams of Data Using Advanced Data Visualization

and Analytics Software for Improved Reservoir Management, Rokan Light Oil, Indonesia. *SPE/IATMI Asia Pacific Oil & Gas Conference and Exhibition*, Bali, 29-31 October 2019. <https://doi.org/10.2118/196318-MS>

- [25] iFour Technolab (2016) Office Add-in Development: VSTO Add-ins vs JavaScript API. <https://www.ifourtechnolab.com/blog/office-add-in-development-vsto-add-ins-vs-javascript-api>
- [26] Excel-DNA Free and Easy. NET for Excel. <https://excel-dna.net/>