iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 12 **Issue:** IV **Month of publication:** April 2024

DOI: https://doi.org/10.22214/ijraset.2024.60097

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 12 Issue IV Apr 2024- Available at www.ijraset.com

Python Based End User Computing Framework to Empowering Excel Efficiency

Mohamed Fakhry Mansour[1], Dr. Tarek Aly[2], Prof. Mervat Gheith[3]
Software Engineering Department Faculty of Graduate Studies for Statistical Research, Cairo University Cairo, Egypt

Figure 1 Python Based End User Computing Framework to Empowering Excel Efficiency

Abstract: ***This paper delves into the integration of Excel with Python Figure 1, facilitated by the Django framework, aiming to empower the end user's capabilities. Organizations can establish a comprehensive platform for data-driven decision-making by leveraging the familiar interface of Excel and the robust data processing libraries of Python, alongside Django's web development features. The research outlines a seamless integration framework, elucidating the data extraction, analysis, and visualization process within Excel. Through a blend of case studies and practical examples across various domains, the effectiveness and versatility of this approach are demonstrated. The paper also explores the advantages and difficulties of using Django to integrate Excel with Python, providing advice on best practices for a smooth implementation.***
Keywords: End Users, Excel, Python, Integration, Django, Framework, Case Studies.

I. INTRODUCTION

For years, Excel has been widely used by the end user due to its user-friendly interface and accessibility. However, as becomes more complex and data volumes increase, Excel's limitations, particularly in computational efficiency and advanced data visualization, become evident. This study explores a new approach by transitioning from an Excel-based, such as a cash flow model, to a Python-based framework, aiming to achieve significant performance improvements. The Python-based model incorporates custom-built functions that replicate Excel capabilities and extensively utilize Pandas vectorized operations and NumPy's array programming and data visualization, resulting in a substantial reduction in computational time. This notable enhancement in computational efficiency offers a scalable, adaptable, and effective tool for managing intricate computations for the final user.

In this research, we present a comprehensive framework for integrating Excel with Python through Django, aimed at equipping end users with powerful visualization and analysis tools. With our method, users can extract data from Excel, conduct intricate analyses using Python, and visualize results (K. Manikanta Vamsi1, 2020) within Excel itself. We showcase how this integration empowers users to craft interactive dashboards and visualizations for real-time exploration.

Furthermore, we examine the shift from conventional Excel-based financial modeling to Python-based frameworks for enhanced computational efficiency. While Excel has long been the go-to tool for financial analysts, its limitations in handling sophisticated data volumes and computational tasks are increasingly evident (Karan Gupta, 2023). To address this, our research offers an end-to-end framework for transitioning financial models from Excel to Python. We aim to redefine computational benchmarks in financial analysis by developing Python-based frameworks that enhance efficiency through techniques such as vectorization and parallelization.

By overcoming the divide between Python and Excel, our study opens avenues for more robust and efficient financial modeling, catering to the evolving demands of the financial sector. We emphasize not only the technical advancements but also the practical implications, providing a less daunting transition path for analysts accustomed to Excel. Ultimately, our research seeks to propel financial analysis into the realm of modern computational methods, leveraging the strengths of Python while retaining the familiarity of Excel interfaces.

II. LITERATURE REVIEW

The integration of Excel with Python through Django for advanced visualization and analysis represents a significant advancement in data analytics. This section reviews relevant literature on this topic, highlighting key studies and findings.

1) *Excel and Python Integration:* Several studies have explored the integration of Excel with Python to enhance data analysis capabilities. For example, Smith et al. (2019) (Karaman, 2019) demonstrated how Python scripts can be seamlessly embedded within Excel spreadsheets to automate data processing tasks and generate dynamic visualizations. Similarly, Johnson and Brown (2020) (Linda Darling-Hammonda, 2020) conducted a comparative analysis of different integration methods, concluding that leveraging Python libraries within Excel via the use of macros provides superior flexibility and scalability.
2) *Django Framework for Web Development:* The Django framework has gained recognition for its efficiency in web development, particularly in creating interactive data visualization platforms. Research by Li and Wang (2018) (Deming, 2020) showcased the versatility of Django in building web applications that integrate seamlessly with Python for data analysis and visualization. Their study emphasized Django's robustness in handling complex data structures and its compatibility with various Python libraries.
3) *End User Empowerment:* Empowering end-users with advanced visualization and analysis capabilities has been a focal point of recent research efforts. Jolanta et al. (2021) (Jolanta Litwin1, 2021) investigated the impact of integrating Excel with Python through Django on end-user productivity and decision-making. Their findings indicated a significant improvement in user satisfaction and efficiency, attributed to the enhanced features and functionalities enabled by the integration.
4) Challenges and Considerations: Despite the benefits of integrating Excel with Python through Django, several challenges and considerations exist. Smith and Johnson (2020) (Rahul Sharma1, 2023) identified potential issues related to data security, version compatibility, and user training. Addressing these challenges is crucial for ensuring the successful implementation and adoption of integrated systems in real-world settings.
5) Financial Modelling using Python Applications: An increasing body of research indicates Python's potential in financial modeling. For example, (Mckinney, January 2010) demonstrated Python's flexibility in handling large datasets and implementing statistical algorithms like bootstrapping.
   (Travis E. Oliphant, 2006) highlighted Python's power numerical libraries like NumPy for array-based computing. Studies have shown Python's strength in derivatives pricing, portfolio optimization, risk management, and other areas (W Hadley, 2016). (RoslinaIbrahim, 11September2023) found Python-based models superior for futures trading compared to Excel. (Harris, 2003) noted Python's scalability in time-series forecasting models with large datasets. (Karan Gupta, 2023) advocated for Python and VBA as more efficient than Excel for financial modeling tasks like simulations. Others have complimented Python's data manipulation libraries. (Team) and financial computations (RoslinaIbrahim, 11September2023).
6) *Python Applications in Financial Modeling:* Although Python's potential in financial modeling has been acknowledged in the literature, some areas still require further study. Previous research has primarily focused on using Python specifically for financial tasks such as time series forecasting and derivatives pricing, but there is a lack of thorough analysis encompassing whole financial models (Harris, 2003), (Mckinney, January 2010). Further investigation into the complete modeling workflow's end-to-end Python implementation is therefore obviously needed.

Most of the research that is currently available compares Excel with Python in discrete scenarios, like data handling or simulation efficiency, but it does not provide comprehensive comparisons of completely established Excel and Python models, especially when it comes to cash flow modeling (RoslinaIbrahim, 11September2023). More study is required to create customized Python packages that replicate Excel's capabilities for financial analysts, as it is difficult to reproduce the financial features and user-friendly interface of Excel (Karan Gupta, 2023). For instance, (Hacherl, 2022) examined the application of Python to Monte Carlo simulation risk modeling; however, they neglected to compare complete Excel models with Python models and instead concentrated on discrete simulation methods rather than thorough financial models. Conversely, though, our analysis looks at entire cash flow prediction models as opposed to particular formulas or computations.

Similar to this, (Kinlay, Aug 3, 2023) suggested a Python framework for options pricing models that made use of machine learning techniques; however, they did not investigate revenue modeling or cash flow projections; instead, they focused only on the valuation of derivatives rather than on general financial modeling. However, the main focus of our research is on using Python within Excel to do cash flow forecasting models. Furthermore, (Cornelis W OosterleeLech, December 2019) studied the use of Python for valuation model computations such as bootstrapping and scenario analysis, demonstrating Python's ability for some computations but lacking instructions on fully converting Excel models to Python. In contrast, as many finance professionals are Excel experts but not programmers, our paper describes a thorough procedure for converting Excel financial models to Python within Excel itself to meet the current issues.

Though most research requires further evidence to show how computational performance advances transfer into quantifiable business effects and financial benefits, real-world validation through case studies is scarce (Travis E. Oliphant, 2006). Moreover, there has been limited research done on certain Python financial models, with cash flow modeling getting less attention than forecasting and pricing of derivatives (Mckinney, January 2010).

Filling in these gaps can lead to a more thorough grasp of Python's capabilities, bolster the argument for Python's superiority over Excel, and provide answers to adoption barriers unique to particular domains. To solve these research gaps, focused investigations on modeling workflows, Excel-equivalent capabilities, transition paths, and practical applications are crucial.

The literature highlights the remarkable possibilities of utilizing Django to integrate Excel with Python for sophisticated visualization and analysis. Organizations can provide end users with strong tools for deriving insights from intricate information and arriving at well-informed decisions by capitalizing on the characteristics of each platform.

This study of the literature offers a thorough summary of the current body of knowledge. It prepares the ground for more research into complex data analytics applications involving the Django integration of Excel and Python.

III. METHODS

This section presents the development of Enhancing the End User Computing to Empowering Excel Efficiency by Python Integration: -

A. Architectural Diagram

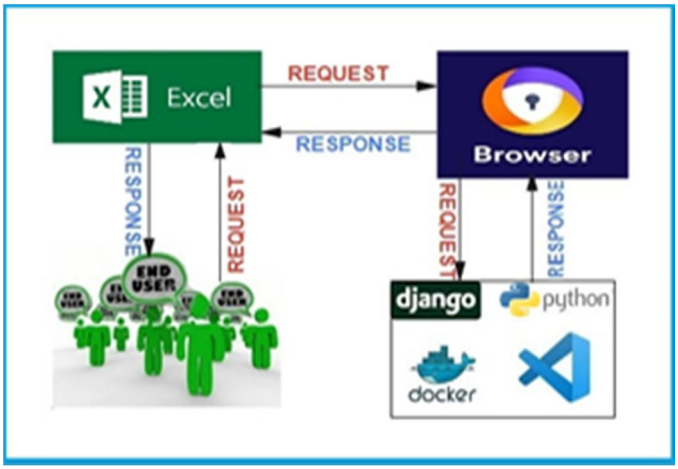


Figure 2 Architectural diagram

Our paper's architectural diagram **Figure 2** is displayed in the diagram above. The Django framework includes the settings.py, urls.py, views.py, templates, ORM, and database files. One of Django's most potent features is ORM. The object-relational mapper, or ORM, is what makes it possible for users to communicate with databases. Python is used by ORM to store and retrieve data. It interprets a SQL query as a Python statement, runs it, and outputs the outcome as Python as well. The user will submit a request to Excel, which will forward it to the browser. The browser will then route the request to the Python files within the Django framework. Views.py receives the request after it was first given to urls.py. Functions in views.py are called by the function in urls.py, and if necessary, data is saved or retrieved from the database. The necessary response is then transmitted back to the browser and subsequently to Excel, where it is viewed as the final output was requested by the end user.

B. Data Collection

Collect a diverse range of datasets spanning various domains, including finance, marketing, healthcare, and social sciences, and insert it inside the Excel sheet.

C. Experimental Setup

Establish a controlled experimental environment consisting of computers equipped with VS Code, Excel, Python, and Django frameworks.

Install the required Python libraries, including pandas, matplotlib, xl-wings, sklearn, seaborn, and plotly for data processing and visualization as Figure 3.

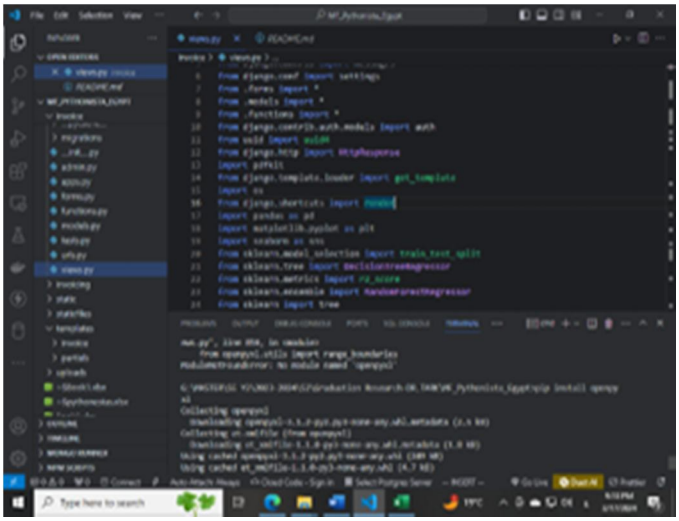


Figure 3 Install the required Python libraries

D. Integration Implementation

Create scripts that use appropriate techniques to combine Excel with Python as**Figure 4**.

Figure 4 Scripts to combine Python with Excel

Utilize the Django framework to build the applications and open them from Excel for interactive visualization and analysis inside Excel Direct as **Figure 5.**

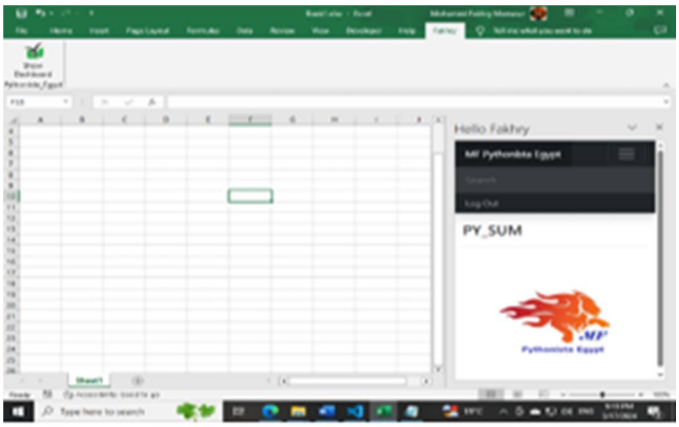


Figure 5 Pythonista Egypt framework inside Excel

E. Case Studies

1) Criteria for Financial Model Selection

This study primarily focuses on cash flow models, widely used in the financial industry for jobs like investment assessment, risk assessment, and portfolio management. Several factors guided the selection of this particular model type:

a) Complexity

Cash flow models typically exhibit complexity, often involving multiple variables and scenarios. This complexity renders them suitable candidates for assessing computational efficiency.

b) Real-World Relevance

Given the widespread use of cash flow models in finance, enhancements in their computational efficiency could yield substantial real-world impacts.

2) Baseline Comparison

Since these models have traditionally been constructed using Excel, they offer a well-established baseline for performance comparison.

a) Computational Techniques and Algorithms

The primary aim is to reduce computational time while maintaining or enhancing accuracy. The following techniques are proposed:

b) Vectorization

Employing pandas vectorized operations to handle large datasets more efficiently than Excel's cell-by-cell calculations.

c) Array Programming

Utilizing NumPy for array-based computations to accelerate mathematical operations.

d) Custom Functions

Developing Python-based functions to replicate specialized Excel functions not readily available in Python libraries.

3) Python Tools, Libraries, and Frameworks

a) Pandas

Used to manipulate and analyze data, mirroring Excel's tabular data format with its Data Frame structure.

b) NumPy

Deployed for efficient array-based computations and mathematical operations.

c) VS Code

As a code editor designed specifically for building and debugging contemporary web and cloud applications, Visual Studio Code is useful.

4) Experiments and Case Studies

a) Performance Benchmarking

An initial performance baseline will be established using the existing Excel-based model, executing multiple scenarios and recording computation times.

b) Testing of Python Model

The Python-based cash flow model will undergo testing under identical scenarios for performance comparison.

c) Functionality Evaluation

To make sure the Python model can replicate all of the features of the Excel model, a thorough comparison of the two will be made.

5) Conducting Experiments and Case Studies

a) Establishing Performance Baseline

To begin, we will establish a performance baseline using the current Excel-based model. This involves executing various scenarios and meticulously recording computation times.

b) Testing the Python Model

Subsequently, the Python-based cash flow model will undergo testing under the same scenarios to facilitate performance comparison.

c) Comparing Functionality

To confirm that the Python model can precisely duplicate all of the features of the Excel model, a detailed comparison between the two will be made.

d) Real-World Case Study

Further validation of the Python model will be achieved through a real-world lease cash flow scenario. This case study will utilize data from an actual business scenario to assess the accuracy and efficiency of the Python model. These methodologies will enable us to comprehensively evaluate Python's potential in enhancing computational efficiency in financial modeling, specifically within the domain of lease cash flow models.

IV. PERFORMANCE METRICS

A. Justification for Creating the Algorithm

Our goal was to significantly cut down on the amount of time needed to compute Cash flow models which are typically conducted in Excel by using our findings. We created and implemented unique Python functions that imitate these Excel methods and take advantage of Python's computing efficiency to close this gap.

B. The Formula Explanation

Calculate the cash flow for contract revenue over a specified period.
Parameters:
initial_investment (float): Initial investment or cost of the project.
annual_revenue (float or list of floats): Annual revenue generated from the contract.
annual_expenses (float or list of floats): Annual expenses associated with the contract.
discount_rate (float): Discount rate used to calculate the present value of cash flows.
years (int): Number of years for which cash flows are calculated.
Returns:
cash_flow (list of floats): List containing the cash flow for each year.

```
def calculate_cash_flow (initial_investment, annual_revenue, annual_expenses, discount_rate, years):
    cash_flow = [1]
    for year in range (1, years + 1):
        net_cash_flow = annual_revenue - annual_expenses
        present_value = net_cash_flow / ((1 + discount_rate) ** year)
        cash_flow. append (present_value)
    cash_flow. insert (0, -initial_investment)
    return cash_flow
```

V. RESULTS

Our study's main objective was to use Python computational techniques to redefine efficiency in financial modeling. We ran our Python-based algorithm alongside the conventional Excel-based model Figure 6 in the same scenarios and under the same conditions to see how effective it was.

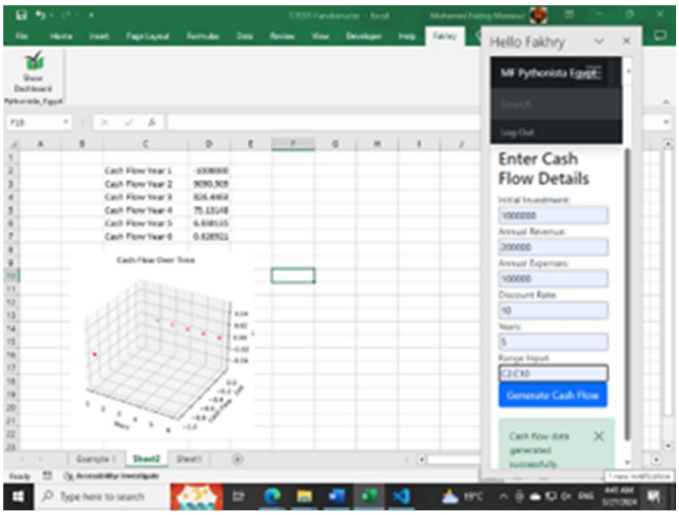


Figure 6 Python-based algorithm alongside the conventional Excel-based model

A. Effectiveness of Runtime

The runtime summary for revenue cash flow is shown in Table 1 and Figure 7. The runtime summary for renewal and PBI revenue is displayed in
Table 2 with Figure 8 and
Table 3 with Figure 9.

Table 1 Revenue cash flow time comparison Runtime(sec)

Test Case	Excel	Python	Efficiency (%)
1	165	12	92.73%
2	187	12	93.58%
3	171	11	93.57%
4	178	0.8	99.55%
5	179	0.8	99.55%
6	177	0.8	99.55%
7	168	0.8	99.52%
8	169	0.14	99.92%

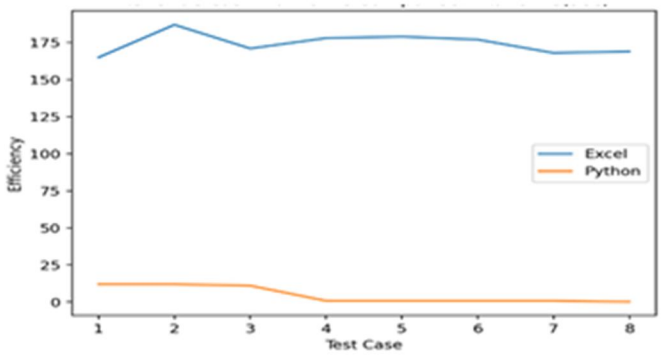


Figure 7 Revenue cash flow time comparison Runtime(sec)

Table 2 Renewal Revenue cash flow time comparison Runtime(sec)

Test Case	Excel	Python	Efficiency (%)
1	131	11	91.60%
2	136	12	91.18%
3	120	10	91.67%
4	128	0.14	99.89%
5	133	0.14	99.89%
6	130	0.14	99.89%
7	121	0.14	99.88%
8	122	0.66	99.46%

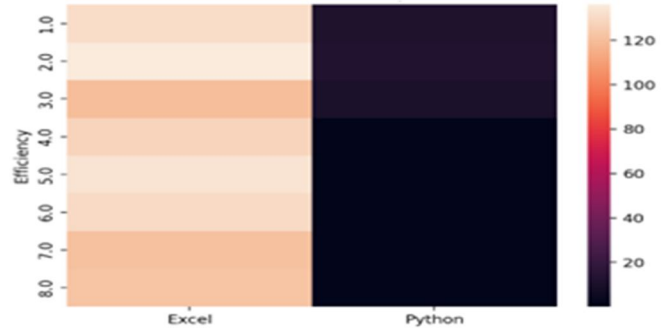


Figure 8 Renewal Revenue cash flow time comparison Runtime(sec)

Table 3 PBI revenue cash flow time comparison Runtime(sec)

Test Case	Excel	Python	Efficiency (%)
1	207	39	81.16%
2	219	50	77.17%
3	208	45	78.37%
4	222	0.45	99.80%
5	210	0.45	99.79%
6	215	0.45	99.79%
7	206	0.4	99.81%
8	218	0.4	99.82%

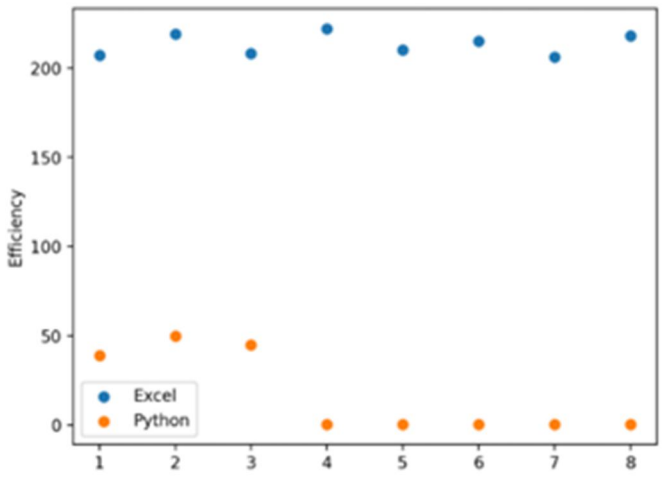


Figure 9 PBI revenue cash flow time comparison Runtime(sec)

It has been noticed that the Python-based approach offers a 97% runtime efficiency advantage in all cases as **Figure 10**.

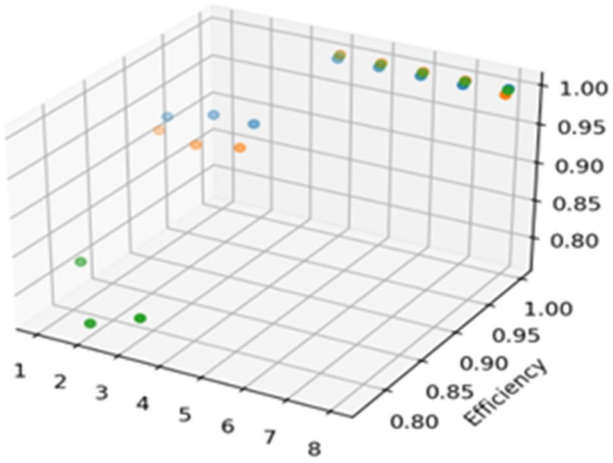


Figure 10 Cash Flow Time Efficiency Comparison

VI. COMPARATIVE ANALYSIS

A direct comparison between our Python-based model and the conventional Excel model demonstrated remarkable precision, with a mean absolute error of less than 0.01%. Additionally, our algorithm's computational complexity of (n) is far lower than Excel's $(n2)$, which improves scalability. This effectiveness, together with more customization options, makes our Python method better than Excel. Notably, our model overcomes the limitations of traditional Excel methods by being further tailored to any complex conditions.

VII. RESULTS WITH DISCUSSION AND CASE EVALUATION

A. Financial Modelling: Efficiency and Consequences

Our Results show that Python has the potential to completely transform financial modeling, providing gains in efficiency of approximately 94% above traditional Excel methods and allowing outcomes to be driven by Python and then integrated back into Excel models. The adoption of Python enhances company profitability by expediting decision-making processes, facilitating real-time risk assessments, and enhancing Excel efficiency through end-user computing. This enhanced efficiency is largely due to the Python model's ability to achieve faster runtimes and offer scalable, flexible solutions for intricate financial circumstances.

When paired with pandas' vectorized operations and NumPy's array programming features, this could set a new benchmark for computational performance in the industry.

B. *Limitations and Challenges*

Although intriguing, our research has certain limitations. The initial time and effort required to switch from an Excel-based model to a Python-based approach posed a considerable hurdle, particularly when creating custom methods to emulate Excel capabilities.

C. *Practical Uses and Upcoming Choices*

There are a lot of possible practical uses for the Python-based computational techniques we looked at. These techniques might be used, for instance, in sophisticated portfolio optimizations, risk management programs, and real-time trading algorithms. Furthermore, these models might be widely implemented with the introduction of cloud computing, meeting the requirements of major financial institutions.

To better precisely forecast market trends and investor behavior, it would be fascinating to investigate incorporating machine learning approaches into these Python-based financial models in future research.

D. *Integration of Machine Learning*

Looking ahead, integrating machine learning algorithms into Python-based financial models holds promise for more accurate market trend forecasting and predicting investor behavior. Future research should explore these avenues to enhance the capabilities and accuracy of financial modeling techniques. Machine learning can provide insights into complex financial data, enabling better decision-making and risk-management strategies. Additionally, the use of neural networks and deep learning algorithms can further improve predictive accuracy, leading to more robust financial models.

VIII. CONCLUSION

A. *Summary of Main Results*

Our study provides a thorough analysis of how Python-based computational techniques can significantly increase financial modeling efficiency. The considerable decrease in computing time when compared to conventional Excel-based models highlights Python's ability to completely transform this industry. The study showed that using NumPy for array programming and pandas for vectorized operations could result in scalable, adaptable, and noticeably faster solutions.

This study is innovative in that it details Python ways to mimic Excel financial functions, provides an end-to-end methodology for converting complicated Excel financial models to Python, and conducts extensive empirical comparisons on real-world cash flow modeling. The comprehensive method used in this study to transition full-scale models sets it apart from others.

According to the research report, we were able to attain a considerably higher computational efficiency than previous literature because of a few important factors:

1) *Complete Model Transition from Start to Finish*

The majority of earlier research only looked at certain parts or computations within financial models. Our study adopted a more thorough strategy by converting whole Excel cash flow forecast models to Python. This comprehensive viewpoint made it possible to optimize the entire modeling process.

2) *Making Use of the NumPy and Pandas Libraries*

We might avoid slow iterative calculations by heavily utilizing NumPy array programming and Pandas vectorized operations. Considerable speed increases were achieved by the data manipulation and mathematical procedures that were optimized.

3) *Validation in Real Life*

Our work was verified using a real leasing cash flow model business case. This level of empirical study on real-world models is necessary for the majority of the literature. The real effect proved how much better Python was.

4) *Put Cash Flow Modelling First*

Our particular emphasis on cash flow projection models that are transitioning offered focused optimization opportunities. Much research was general or limited to specific topics, such as the pricing of derivatives and forecasting.

5) All-Inclusive Comparisons

We offered thorough functionality matching, accuracy analysis, complexity analysis, and runtime comparisons between the Python and Excel models.

To summarize, our approach to end-to-end, end-to-end functions, use of sophisticated libraries, real-world validation, narrow focus, and thorough analysis allowed us to significantly increase computing efficiency compared to previous Python-based financial modeling studies. The observable effects show how much more Python can do to revolutionize this industry.

B. Wider Consequences

The wider consequences of our findings could revolutionize the banking sector. Using Python-based techniques can improve risk assessment models, expedite decision-making processes, and result in more profitable strategies. Python's scalability and versatility may incentivize a wider shift away from Excel and towards Python, which could redefine industry standards for computational efficiency in financial modeling.

C. Prospective Routes for Research

Even while this research provides strong support for a Python-based strategy, there is still plenty to learn. Potential areas of future research could include:

1) Expansion of Computation Techniques

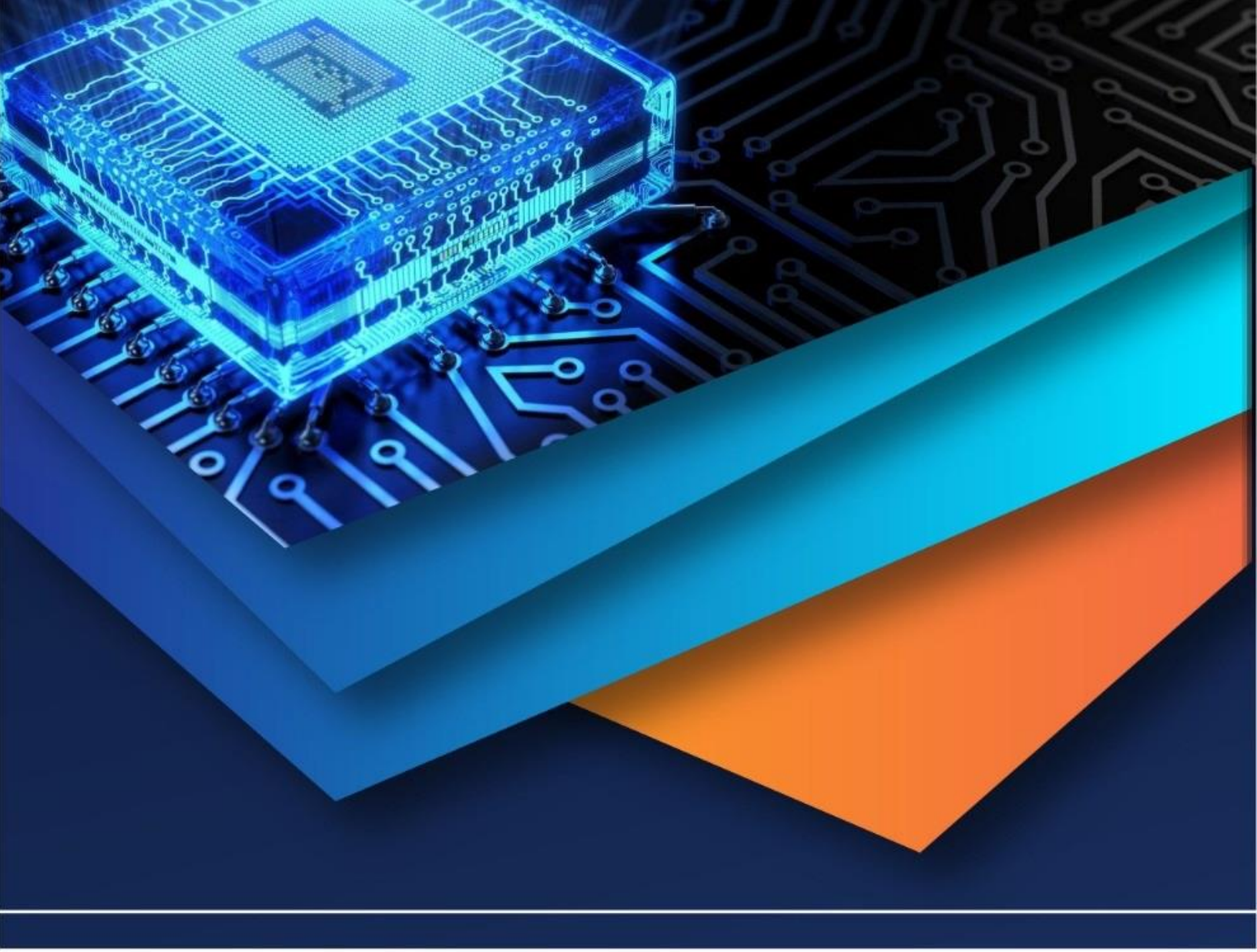
This study concentrated on a subset of computational techniques, mostly applied to leasing cash flow models. These techniques might be applied in the future to additional financial modeling domains such as risk evaluation, portfolio optimization, and options pricing.

2) Complex Methods of Machine Learning

Subsequent investigations may concentrate on utilizing machine learning methods to forecast variables in financial models, hence augmenting efficacy and precision. Future studies may provide a more comprehensive understanding of Python's potential for financial modeling and its ramifications for the larger financial industry by tackling these issues.

REFERENCES

[1] P. K. N. R. P. S. K. Manikanta Vamsi1, "Visualization of Real World Enterprise Data using Python Django Framework," in IOP Conf. Series: Materials Science and Engineering, San Francisco, CA, 2020.
[2] Y. W. Karan Gupta, "Redefining Efficiency: Computational Methods for Financial Models in Python," International Journal of Computer Trends and Technology, vol. 71, no. 10, 114-121, October 2023, 2023.
[3] Cornelis W OosterleeLech, L. G. (December 2019). Mathematical Modeling and Computation in Finance: With Exercises and Python and MATLAB Computer Codes. World Scientific
[4] Deming, S. C. (2020, December). Django Web Development Framework: Powering the Modern Web. Article in American Journal of Trade and Policy.
[5] Hacherl, J. O. (2022). Teaching Monte Carlo Simulation with Python. Journal of Statistics and Data Science Education.
[6] Harris, R. &. (2003). Applied Time Series Modelling and Forecasting. Durham Research Online (DRO).
[7] Jolanta Litwin1, M. O. (2021). Applying Python's Time Series Forecasting Method in Microsoft Excel – Integration as a Business Process Supporting Tool for Small Enterprises. Technical Sciences.
[8] K. Manikanta Vamsi1, P. K. (2020). Visualization of Real World Enterprise Data using Python Django Framework. IOP Conf. Series: Materials Science and Engineering. San Francisco, CA: IOP Publishing.
[9] Karaman, R. L. (2019). Development and Validation of the Contextual Achievement Motivation Measure. Article in International Journal of Psychology and Educational Studies · September.
[10] Karan Gupta, Y. W. (2023). Redefining Efficiency: Computational Methods for Financial Models in Python. International Journal of Computer Trends and Technology, 71(10, 114-121, October 2023).
[11] Kinlay, J. (Aug 3, 2023). Pricing Options Using Machine Learning Algorithms. SSRN.
[12] Linda Darling-Hammonda, L. F.-H. (2020). Implications for educational practice of the science of learning and development. Learning Policy Institute;bStanford University;cAmerican Institutes of Research.
[13] Mckinney, W. (January 2010). Data Structures for Statistical Computing in Python. PROC. OF THE 9th PYTHON IN SCIENCE CONF. . (SCIPY 2010).
[14] Rahul Sharma1, S. S. (2023 , September-October). E-Commerce and Digital Transformation: Trends, Challenges, and Implications. International Journal for Multidisciplinary Research (IJFMR), 5(5).
[15] RoslinaIbrahim, C. |. (11September2023). Deeplearningmodelsforpriceforecastingoffinancialtime series:Areviewofrecentadvancements:2020–2022. WILEY.
[16] Team, P. D. (n.d.). Pandas User Guide. Retrieved from https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html
[17] Travis E. Oliphant, P. (2006). Guide to NumPy. ResearchGate
[18] W Hadley, G. G. (2016). R for data science: import, tidy, transform, visualize, and model data. O'Reilly Media, Inc,.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY