World Journal of Advanced Research and Reviews
eISSN: 2581-9615 CODEN (USA): WJARAI
Cross Ref DOI: 10.30574/wjarr
Journal homepage: https://wjarr.com/

(Research Article)

Program-Based Construction of Scientific Visualization

Monday Eze [1, *], Charles Okunbor [2], Abel Samuel [1] and Oluwatobi Akinmerese [1]

[1] *Department of Computer Science, Babcock University, Ogun State, Nigeria.*
[2] *Department of Computer Science, Admiralty University, Delta State, Nigeria.*

World Journal of Advanced Research and Reviews, 2021, 11(03), 413–423

Publication history: Received on 20 August 2021; revised on 27 September 2021; accepted on 29 September 2021

Article DOI: https://doi.org/10.30574/wjarr.2021.11.3.0488

Abstract

Scientific Visualization remains an integral and inevitable part of every meaningful scientific, industrial and academic research. The focus of this study is to demystify the evolution, design and programmatic construction of scientific visualizations. Real life demonstrations have been achieved in this work using Python Programming Language. This work begins by exploring the programming environment based on a Python Integrated Development Environment (IDE) – the Anaconda. The IDE usage was shown in a chronological sequence with accompanying visual outcomes. The use of plot libraries was discussed, and implemented in real life. One of the demonstration projects is the Zig-Zag plot. Studies were also done on sub-plotting, and how it is used in scientific visualizations, especially where there is necessity to generate variations of outputs originating from a singular dataset. The work also explores the use of pie charts for presentations. A real-life case of how this could be used to visualize the halls of a residential university was demonstrated, with each of the data components labelled in distinct colours. It is hoped that this work will serve as a foothold and useful guide to researchers and other practitioners involved in real life scientific visualization.

Keywords: Scientific Visualization; Python Programming; Anaconda; Integrated Development; Spyder

1. Introduction

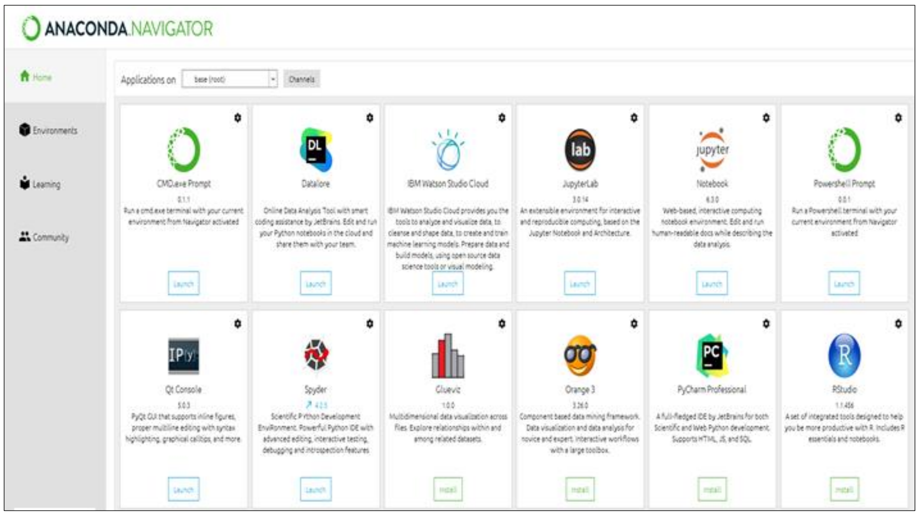
The strength of scientific visualization stems from the generally accepted norm, that a picture speaks more than a thousand words [1]. This cannot be overemphasized in scientific research, where the authenticity of theoretical outcomes needs to be re-enforced using corresponding visualizations. Having mentioned the need for visualizations, the ability of researchers and learners to do-it-by-themselves appears to be of utmost necessity. This work therefore lays an unambiguous foundation for delving into practical implementations of scientific visualizations, especially from the programmatic point of view. This work demonstrates this using Python Programming Language. A number of foundational concepts were covered, some of which are graphical plotting and sub-plotting. Other areas explored in this work are generation of pie charts, bar charts, among others. The next section of this work will focus on the development environment.

2. Exploring the Development Environment

This section begins with an exploratory study of the programming environment [2]. The laboratory cases of this study were carried out in a Python. First, the programming tools are installed accordingly. This is simplified by installing Anaconda [3], a popular open-source [4] Integrated Development Environment (IDE) [5]. The use of an efficient IDE, especially those that incorporate reusable modules no doubt, increase project completion speed in a significant way. The power of Anaconda IDE stems from the fact that it is fused with about a dozen or more tools that enhance software

* Corresponding author: Monday Eze
Department of Computer Science, Babcock University, Ogun State, Nigeria.

construction in a very professional manner. Some of such enabling tools are: Jupyter Notebook [6], Spyder [7], Powershell Prompt [8], among others as shown in the Anaconda Navigation screen in Fig.1.


ANACONDA.NAVIGATOR

Home
Environments
Learning
Community

Applications on base (root) Channels

CMD.exe Prompt
0.1.1
Run a cmd.exe terminal with your current environment from Navigator activated
Launch

Datalore
Online Data Analysis Tool with smart coding assistance by JetBrains. Edit and run your Python notebooks in the cloud and share them with your team.
Launch

IBM Watson Studio Cloud
IBM Watson Studio Cloud provides you the tools to analyze and visualize data, to cleanse and shape data, to create and train machine learning models. Prepare data and build models, using open source data science tools or visual modeling.
Launch

JupyterLab
3.0.14
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.
Launch

Notebook
6.3.0
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.
Launch

Powershell Prompt
0.0.1
Run a Powershell terminal with your current environment from Navigator activated
Launch

Qt Console
5.0.3
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.
Launch

Spyder
↗ 4.2.5
Scientific Python Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features
Launch

Glueviz
1.0.0
Multidimensional data visualization across files. Explore relationships within and among related datasets.
Install

Orange 3
3.26.0
Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.
Install

PyCharm Professional
A full-fledged IDE by JetBrains for both Scientific and Web Python development. Supports HTML, JS, and SQL.
Install

RStudio
1.1.456
A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.
Install

Figure 1 Anaconda Navigation Screen

After a successful installation and launching of the IDE, a number of steps are taken in order to access the programming environment, as outlined in the flowchart in Fig. 2. The flowchart [9] consists of a total of seven distinct chronological steps, labelled accordingly from 1 to 7. In that order, the user first invokes the IDE from command prompt, and when the navigator appears, the user clicks on Jupyter 6.3.0 icon. The terminal pull-down arrow [10] is clicked, and this is followed by the appearance of the power-shell dark screen. Note that this could take a while, thus the necessity to wait a while, as explicitly indicated in step 5. With the appearance of the dark screen, the user types the IDLE command, and presses ENTER key.

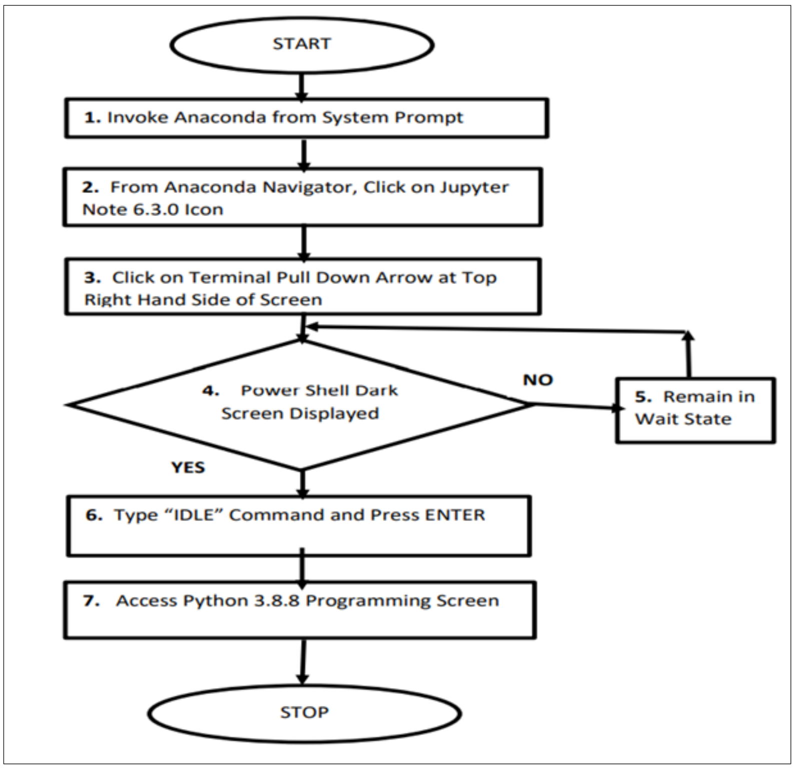
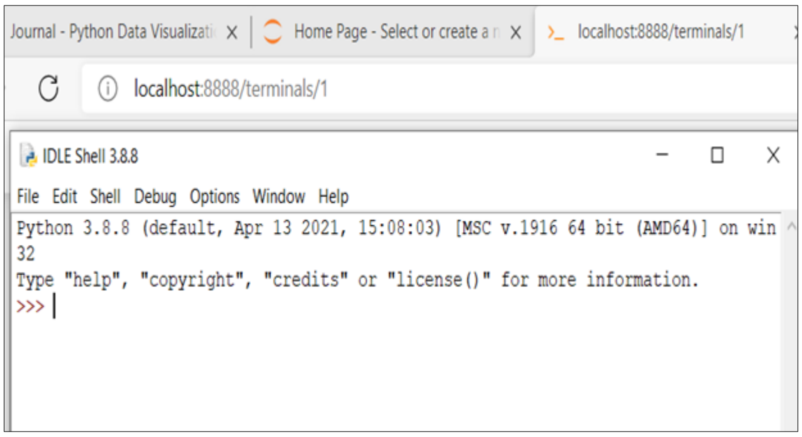

START
1. Invoke Anaconda from System Prompt
2. From Anaconda Navigator, Click on Jupyter Note 6.3.0 Icon
3. Click on Terminal Pull Down Arrow at Top Right Hand Side of Screen
4. Power Shell Dark Screen Displayed
NO
5. Remain in Wait State
YES
6. Type "IDLE" Command and Press ENTER
7. Access Python 3.8.8 Programming Screen
STOP

Figure 2 Flowchart for accessing the Python 3.8.8. Programming Screen

The final outcome in the outlined chronology of steps is the appearance of the Python 3.8.8 programming screen as indicated in step 7, signaling that actual system construction can now commence. A sample Python 3.8.8 screen is shown in Fig. 3. As shown, there is a unique prompt ">>>" where direct commands can be issued for system execution. There are also seven sub-menus [11] consisting of "File", "Edit", "Shell", "Debug", "Options", "Window" and "Help" which are useful for the actual system construction. These will not be discussed further in this research.

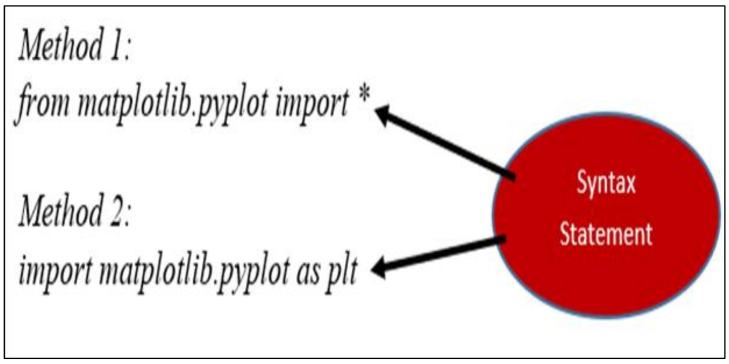

Journal - Python Data Visualizati
Home Page - Select or create a n
localhost:8888/terminals/1
localhost:8888/terminals/1
IDLE Shell 3.8.8
File Edit Shell Debug Options Window Help

```
Python 3.8.8 (default, Apr 13 2021, 15:08:03) [MSC v.1916 64 bit (AMD64)] on win
32
Type "help", "copyright", "credits" or "license()" for more information.
>>> |
```

Figure 3 Python 3.8.8 Programming Screen

3. The Plot Library

One of the key engines of visualization and graphics in Python is the plot library, commonly known as the Matplotlib [12]. The matplotlib drives diverse forms of scientific plots. One of the pre-requisites for using the matplotlib is that it has to first be imported. The import command [13] could be achieved using a number of methods, two of which are shown in the syntax statement in Fig. 4.



Method 1:
*from matplotlib.pyplot import **

Method 2:
import matplotlib.pyplot as plt

Figure 4 Matplotlib Syntax Statement

The first syntax specifies the asterisk "*", which by implication, imports all the contents of the matplotlib. On the other hands, the second option also imports the matplotlib, but in this case, uses an alias which is specified as "plt". Whichever case is used, the import statement must be issued before the actual plot comes into place. The syntax for the plot invocation is shown in STM1 (Statement 1) in Fig. 5, where x, y are tuples of equal cardinalities, and in some cases, [x,] is an optional component of the syntax [14]. Also, in line with the general syntax of the plot statements in STM1, it implies that the statement STM2 is a valid and acceptable plot command, where the first tuple is [1,2,3,4] and the second one is [4,2,6,3]. The tuples [15] can also be presented in indirect format, by first assigning the tuples to variables, and the variables then inserted as arguments to the plot invocation call, as shown in statement STM3.

It can also be deduced that since one of the arguments in STM1 is optional, it implies that statement STM4 is also a valid and acceptable plot invocation [16]. Finally, it is important to note that a number of Python installations require an explicit show statement in order to activate a visual presentation of the internal plots, as will be demonstrated in this work.

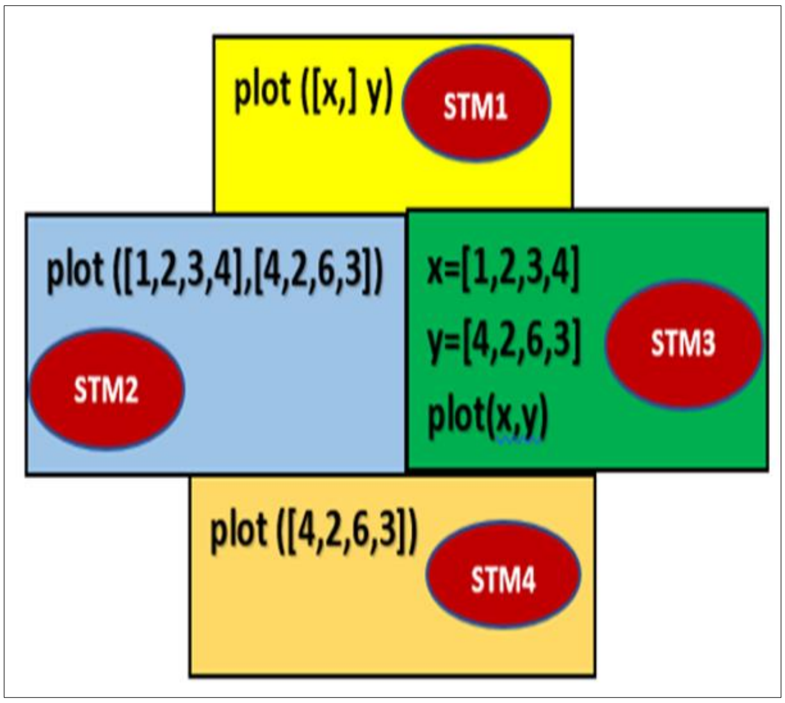


Figure 5 Plot Statement Variations

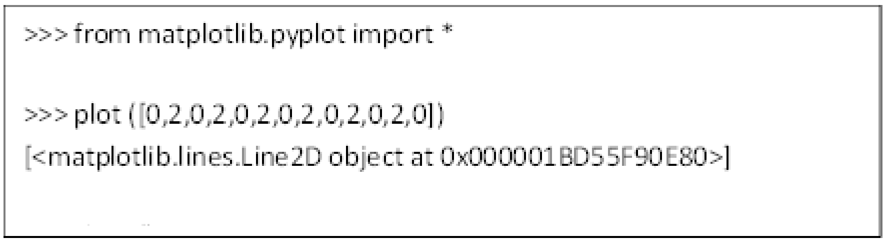
3.1. Illustration 1 - The Zig- Zag Project

The Zig-Zag Project [17] is the first practical implementation in this research. The plot statement makes use of an argument given by Equ (1).

$$Arg = [Z, M, Z, M, Z, M, \dots.] \ (1)$$

where Z = Zero, and M = uniform height.

Thus, a sample code for a zig-zag plot of uniform height M =2 is shown in Fig. 6.



```
>>> from matplotlib.pyplot import *

>>> plot ([0,2,0,2,0,2,0,2,0,2,0,2,0])
[<matplotlib.lines.Line2D object at 0x000001BD55F90E80>]
```

Figure 6 The Zig-Zag Code for Uniform Height M=2

The output after running the source code for the simple zig-zag project is shown in Fig. 7.

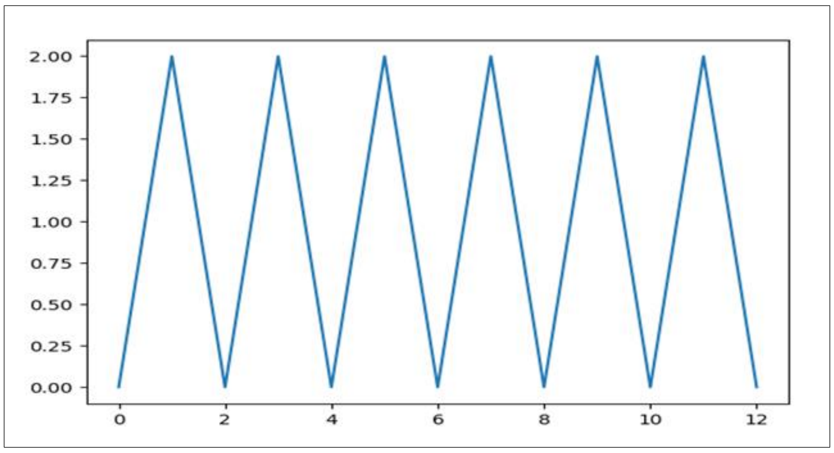


Figure 7 The Zig-Zag Code for Uniform Height M=2

4. Sub-Plot Generation

Sub-plots can be defined as a series of plots in an organized visualized matrix format [18]. Each of the cells in the sub-plots could represent same type, or different types of plots. For instance, one could visualize a singular set of data as a graph [19], a bar chart [20], a pie chart [21], a scatter diagram [22], among others, all within the subplot. A sample grid architecture [23] for a subplot is shown in Fig. 8, where Vxy represents a visual output in row x, column y.

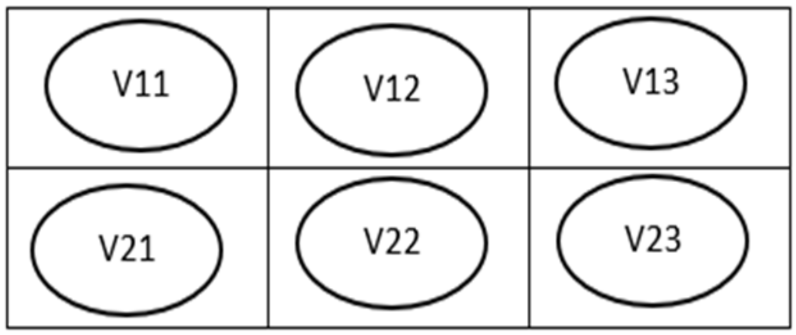


Figure 8 Subplot arrangement in visualized 2x3 matrix format

As shown in the subplot architecture, there are a total 2 x 3 which is 6 possible visual options that can be displayed. The general rule is that any grid x points to a subplot (**x), where the asterisks represent the rows and columns. Thus, in the example shown in Fig. 8, the grids are accessed using the integers x=1,…6 as narrated in Table 1.

Table 1 Grid access representation table

Grids	Exact Sub-Plot
Grid 1	Subplot (231)
Grid 2	Subplot (232)
Grid 3	Subplot (233)
Grid 4	Subplot (234)
Grid 5	Subplot (235)
Grid 6	Subplot (236)

A programmer begins by ensuring that all the sub-plots must be collated as a single entity. This is achieved using a Figure Command with the general syntax [24] given by figure (). An illustration will be given at the appropriate section of this work.

4.1. Illustration 2 - The Sub-Plots Project

The aim of this practical illustration is to create six subplots consisting of an ordinary graph, a bar graph, a horizontal bar chart, a stacked bar chart [25], a box plot [26], and a scatter plot respectively at the same time, using a singular piece of data. The system flow chart is shown in Fig. 9.

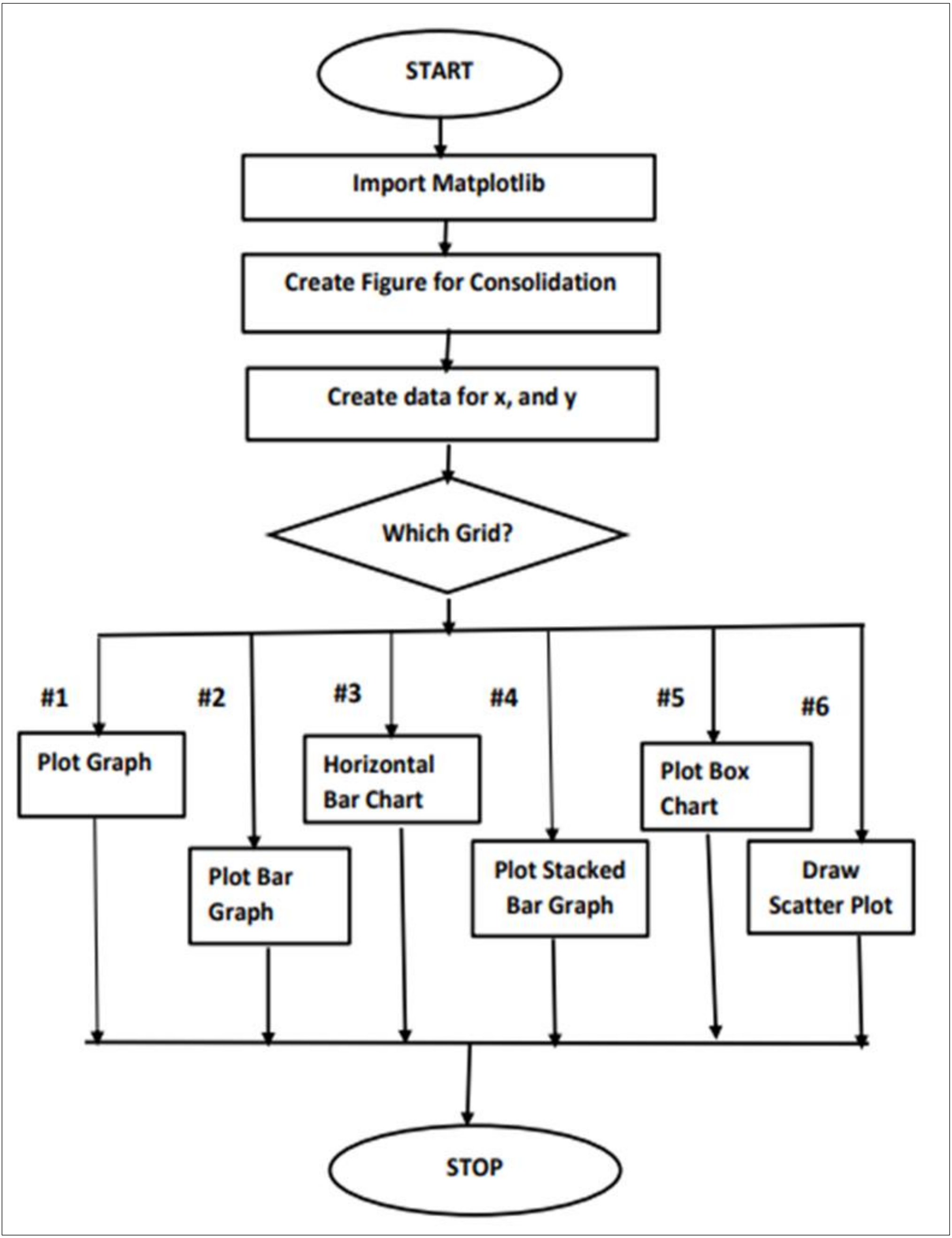
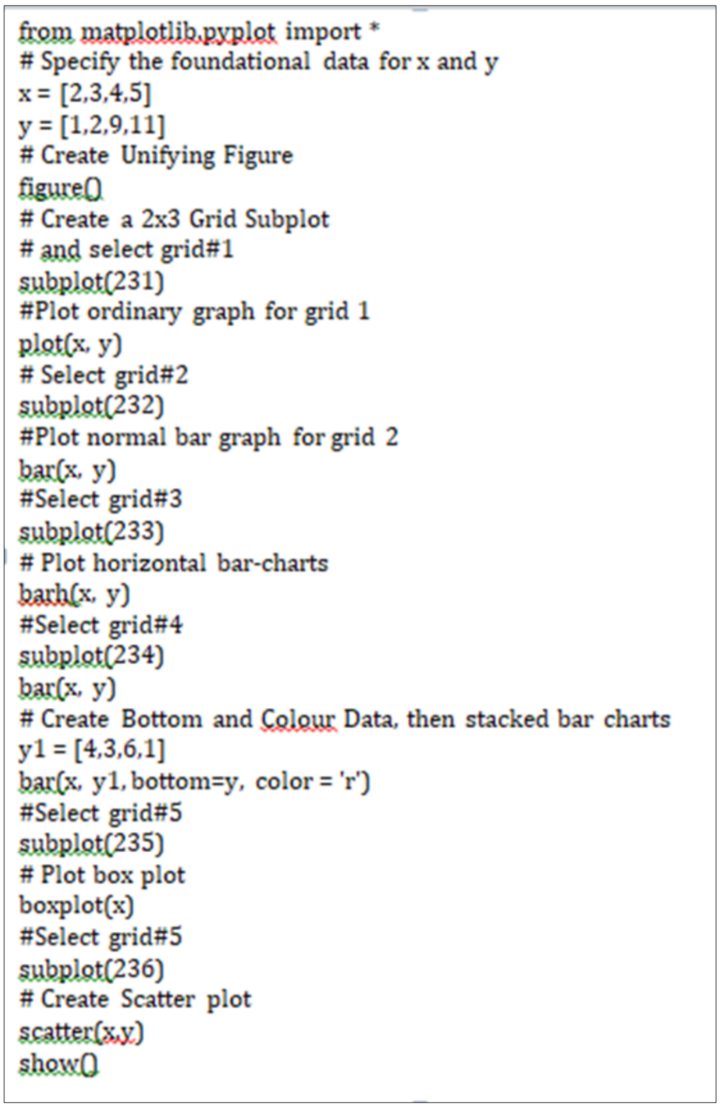


Figure 9 Sub-Plot Flowchart

As show in the flow diagram, the data used in building the visualization are presented using the variables x and y. Thereafter, a grid is created, and the exact value is decided in a form of decisional structure [27], though in actual programming, this follows a simple chronology. For instance, the sixth grid leads to the drawing of a scatter plot. The source code [28] is shown in Fig. 10.



```
from matplotlib.pyplot import *
# Specify the foundational data for x and y
x = [2,3,4,5]
y = [1,2,9,11]
# Create Unifying Figure
figure()
# Create a 2x3 Grid Subplot
# and select grid#1
subplot(231)
#Plot ordinary graph for grid 1
plot(x, y)
# Select grid#2
subplot(232)
#Plot normal bar graph for grid 2
bar(x, y)
#Select grid#3
subplot(233)
# Plot horizontal bar-charts
barh(x, y)
#Select grid#4
subplot(234)
bar(x, y)
# Create Bottom and Colour Data, then stacked bar charts
y1 = [4,3,6,1]
bar(x, y1, bottom=y, color = 'r')
#Select grid#5
subplot(235)
# Plot box plot
boxplot(x)
#Select grid#5
subplot(236)
# Create Scatter plot
scatter(x,y)
show()
```

Figure 10 Source Code for Sub-Plots

It is important to mention that it is very necessary to issue the show () statement, since this leads to the actual display of the visual output on the screen. The resulting output is shown in Fig. 11. The importance of sub-plotting cannot be overemphasized, as it makes it possible of researchers to present a particular piece of data, using various options, side by side. Invariably, such a step helps to demystify analytical complexities [29] as much as possible.

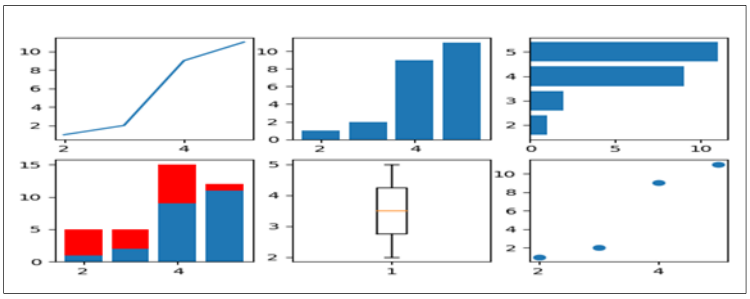


Figure 11 Resulting Output from Sub-Plots Code

5. Pie Chart Generation

Pie chart is used in a number of scientific visualizations. A typical pie chart consists of a number of sectors, whose sizes are determined by the enclosed angles of such sectors. It is important to state that drawing the chart follows an anti-clockwise direction [30]. The label parameter is used to insert labels to the components of the pie chart. An illustration will be presented here.

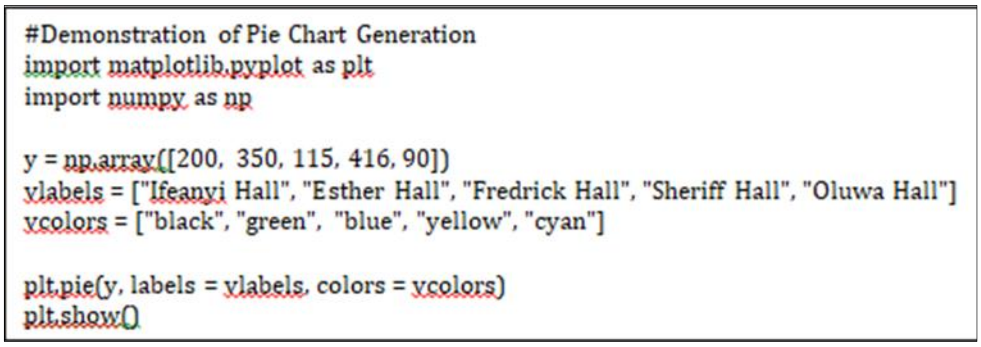
5.1. Illustration 3 – Pie Based Hall Visualization

Suppose a residential university has five students' hall, with a total of 1,171 students distributed as shown in Table 2. The goal is to programmatically visualize this using pie chart.

Table 2 School Hall Identification Data

Label	**Count**
Ifeanyi Hall	200
Esther Hall	350
Fredrick Hall	115
Sheriff Hall	416
Oluwa Hall	90

The source code is show in Fig. 12. In solving this problem, the numpy module was imported, and its special array facility utilized accordingly. As shown in the code, the different components of the pie chart were given distinct labels, being the names of the halls, as well as distinct colours. The colours are Ifeanyi Hall➔ Black, Esther Hall ➔ Green, Fredrick Hall ➔ Blue, Sheriff Hall➔ Yellow and Oluwa Hall ➔ Cyan respectively.



```
#Demonstration of Pie Chart Generation
import matplotlib.pyplot as plt
import numpy as np

y = np.array([200, 350, 115, 416, 90])
ylabels = ["Ifeanyi Hall", "Esther Hall", "Fredrick Hall", "Sheriff Hall", "Oluwa Hall"]
ycolors = ["black", "green", "blue", "yellow", "cyan"]

plt.pie(y, labels = ylabels, colors = ycolors)
plt.show()
```

Figure 12 Source Code for Pie Chart Generation

The output after program run is shown in Fig. 13. It is important to mention that there are other parameters [31] that could be specified in the process of decorating the pie visualization. One of such parameters is the "explode". When one specifies the explode, then the tagged sector of the pie chart is pulled out of the rest of the other sectors.

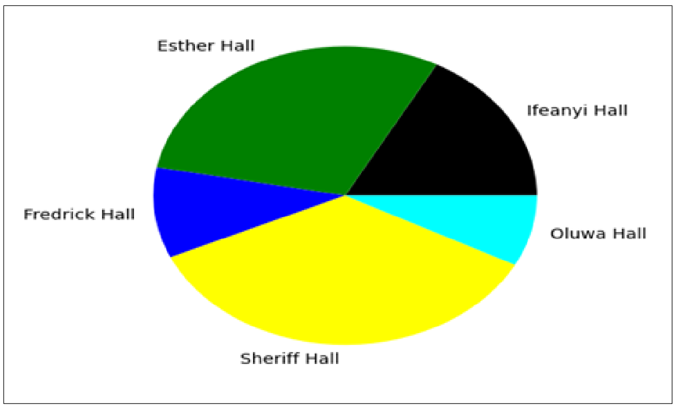


Figure 13 Source Code for Pie Chart Generation

For instance, to tag the explode option [32] on the Esther Hall, the following modifications should be included in the original code.

vexplode =[0, 0.2, 0,0,0]

plt.pie(y, labels = vlabels, colors = vcolors, explode=vexplode)

6. Conclusion

This work has presented a practical oriented study of scientific visualization. The importance of Scientific Visualization cannot be overemphasized. Most importantly is the necessity for computer scientists and researchers to be able to practically utilize the available tools, and possibly evolve new ones. This work is a stepping stone towards understanding and implementing scientific visualization in real life. It demonstrates how to use a popular Python Integrated Development Environment (IDE) known as Anaconda in building visualizations. Real life demonstrations of plotting and sub-plotting were demonstrated. Other outputs demonstrated are bar chart, pier chart, scatter plots, among others. It is hoped that this work will be very useful to researchers and other practitioners involved in real life scientific visualization. In order to enhance understanding of this subject matter, a number of source code segments were shown. There were also well annotated flow diagrams, among others. It is hoped that this work will be very useful to researchers and other stakeholders with interest in python-based scientific visualizations.

Compliance with ethical standards

Acknowledgments

The authors acknowledge themselves for a successful collaboration.

Disclosure of conflict of interest

There are no conflicts of interest in this work.

References

[1] Gordon B. A picture speaks a thousand words: A personal view using images in the teaching of the biological sciences. Bioscience Education. 2003; 1(1): 1-15.

[2] Marilyn T, Ann H, Michael F. Using a computer programming environment and an interactive whiteboard to investigate some mathematical thinking. Procedia – Social and Behavioural Sciences. 2010; 8(1): 561-570.

[3] Lene M, Kim M, Michael H and Alan L. Anaconda: A new tool to improve mortality and cause of death data. BMC Medicine. 2020; 18(1): 1-13.

[4] Zhaohui W. Research on the application of open-source software in digital library. Procedia Engineering. 2011; 15(1): 1662-1667.

[5] Sakila B, Kanakasabapathi V. An IDE for Android Mobile phones with extended functionalities using best developing methodologies. International Journal of Computer Networks and Communications. 2013; 5(4): 131-145.

[6] Kelvin M, Leighton P, Stacy R, David B. Towards Collaborative Open Data Science in Metabolomics using Jupyter Notebooks and Cloud Computing. Metabolomics. 2019; 15(10): 124-161.

[7] Damien R, Matt R, Thadde R, Kinsey C. Introduction to Anaconda and Python: Installation and Setup. Python for research in psychology. 2020; 16(5): s3-s11.

[8] Hitesh M, Subarna P, Amiya K, Edalatpanah S, Ranjan K. A tutorial on PowerShell Pipeline and its loopholes. International Journal of Emerging Trends in Engineering Research. 2020; 8(4): 975-982.

[9] Benjamin K, Jeroen M, Michel M. Modeling process flow using diagrams. Quality and Reliability Engineering. 2009; 26(1): 341-349.

[10] Malihe A, Matthew S. In the proceeding of 27th Conference on Computational Linguistics, Held in Santa Fe, Mexico. 2018; 3552-3563.

[11] Andreas H, Paivi T. Screen_Mgmt: A Python Module for Managing Screening Data. Journal of laboratory automation. 2015; 20(1): 56-59.

[12] Polina L. Python Libraries Matplotlib, Seaborn and Pandas for Visualization Geospatial Datasets Generated by QGIS. Scientific Annals of University of IASI. 2020; 64(1): 13-19.

[13] Sai M. Python and its libraries in data science and related fields. Data Science and Engineering. 2020; 1(1): 1-3.

[14] Norbert H. Basic operations: minimal syntax – semantics. Catalan Journal of Linguistics. 2009; 8(1): 113-139.

[15] Ekmekci B, McAnany C, Mura C. An introduction to programming for Bioscientists: A Python-Based Primer. PloS Computatiuonal Biology. 2016; 12(6): 1-43.

[16] Oulasvirta A, Dayama N, Shiripour M, John M, Karrenbauer A. Combinatorial Optimization of Graphical User Interface Design. In Proceedings of IEEE. 2020; 108 (3): 434-464.

[17] Deasy K, Ani R, Ngurah P. Student's Creative Zig-Zag Book: Improving their Concepts Understanding by using Project Based Learning. Journal of Primary Education. 2019; 8(2): 209-217.

[18] Ann T, Phil H, David W, Philip B, Bradbury M, Van Z. Identifying Decision Useful Information with the Matrix Format Income Statement, Journal of International Financial Management and Accounting. 2008; 19(2): 184-217.

[19] Susanti Y, Puspitasari YI, Khotimah H. On Total Edge Irregularity Strength of Staircase Graphs and Related Graphs Iranian Journal of Mathematical Sciences and Informatics. 2020; 15 (1): 1-13.

[20] Linda C, Felice S. The Effects of Data and Graph Type on Concepts and Visualizations of Variability. Journal of Statistics Education. 2010; 18(2): 1-16.

[21] Marcin K, James H, Agnieszka W, Małgorzata T. Multiple Pie Charts: Unreadable, Inefficient, and Over-Used. Journal of Scholarly Publishing. 2015; 46(3): 282-289.

[22] Michael F, Daniel D. The Early Origins and Development of the Scatterplot. Journal of the History of the Behavioral Sciences. 2005; 41(2): 103-30.

[23] Hai J, Deqing Z, Hanhua C, JianHua S, Song W. Fault-tolerant grid architecture and practice, Journal of Computer Science and Technology. 2003; 18(4): 423-433.

[24] Hulya M. The analysis of A1 level speaking exam in terms of syntax: The effect of general competence on syntax in A1 level speaking. Journal of Language and Linguistic Studies. 2017; 13(1): 27-40.

[25] Indratmo I, Lee H, Joyce B, Ben D. The efficacy of stacked bar charts in supporting single-attribute and overall-attribute comparisons. Visual Informatics. 2018; 2(3): 1-11.

[26] Marmolejo-Ramos F, Siva T. The shifting boxplot: A boxplot based on essential summary statistics around the mean. International Journal of Psychological Research. 2010; 3(1): 37-45.

[27] Kirti T, Alpika T, Shipra S, Vandana D. Merging of Data Flow Diagram with Unified Modeling Language. International Journal of Scientific and Research Publications. 2012; 2 (8): 1-6.

[28] Muhammad W, Lukman A, Rizaldi A, Abdus S, Ismail I. Source Code Library (SCL): Software Development Learning Application. International Journal of Scientific & Technology Research. 2020; 9(2): 175-182.

[29] Matthijs K. Perspectives on Complexity, Its Definition and Applications in the Field. Complicity: An International Journal of Complexity and Education. 2017; 14(1): 16-35.

[30] Tanaka K, Suzuki N, Sakuramoto K. Further Analysis of Clockwise Loops and Anticlockwise Loops Observed in a Stock-Recruitment Relationship. Open Access Library Journal. 2021; 8(1): 1-52.

[31] Tan C, Ryspek U, Amin H, Low K, Teh Y, Muhammad S. Parameters Investigation of Mathematical Model of Productivity for Automated Line with Availability by DMAIC Methodology. Journal of Applied Mathematics. 2014; (1): 1-7.

[32] Machado F, Malpica N, Borromeo S. Parametric CAD modeling for open source scientific hardware: Comparing OpenSCAD and FreeCAD Python scripts. Plos One. 2019; 14(12): 1-30.