UCC

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

FINAL-YEAR PROJECT REPORT

BSc IN COMPUTER SCIENCE

A new Python API for Visualisation of Sets

Nathan Crowley
Department of Computer Science

Supervised by
Dr Rosane Minghim
Department of Computer Science

April 2022

Abstract

This project involves creating a Python Package to be used by students or academic professionals, with the aim of visualising sets. By using the approach of *InteractiVenn*[1] to Set Visualisation, to allow a greater understanding of given sets, and their unions and intersections. The Python package developed should allow users to visualise up to six input sets of data as an interactive Venn diagram.

The developed Python package should take into account that the users may have little or no technical skills, this should direct the design towards being as easy as possible to use for the target audience. The documentation of the Python package must have clear and concise information on its functionality, along with concrete examples displaying to the user the operations that can be performed.

The final year project includes; Development of a Python package for Set Visualisation, experimental research conducted on previously established similar Set Visualisation systems; *InteractiVenn*[1] *and UpSet*[2], evaluation of the Python package created in contrast to the earlier specified established system; *UpSet*[2] to highlight the advantages and disadvantages of each package given a number of examples.

Declaration of Originality

This is to certify that the work I am submitting is my own and has been done by me solely and not in consultation with anyone else. Neither I nor anyone else have submitted this work for assessment, either at University College Cork or elsewhere. I have read and understood University College Cork's exam regulations, plagiarism policy and Code of Honour. I understand that breaches of this declaration are serious issues and can incur penalties.

Nathan Crowley

Signature of Student: *Nathan Crowley*

Date: Monday 25th April, 2022

Acknowledgements

This report and research would not have been possible without the guidance and support of my supervisor Dr Rosane Minghim. Dr. Minghim’s supervision has been critical from the beginning of the project to the conclusive report. Her knowledge and management showcased in bi-weekly meetings, manifested in meeting reports as well as a continuous logbook of the project’s progress, helping keep the project and myself on track to achieve the goals set for the project.

I would like to thank Frank Boehme, the administrator of the Final Year Project process for the BSc Computer Science degree program. Mr. Boehme’s lecture at the beginning of the academic calendar assisted in many aspects of the report, such as report structure, key dates and preferred language, along with the information necessary for submission in a timely manner.

Contents

1. Introduction

1.1. Overview of Python Package

A Python *package* is a reusable portion of Python instructions that can be downloaded and imported into a project, allowing a user to take advantage of the created functionality. This allows users of all technical levels the ability to efficiently import code to their project that may have either been: excessively time-costly to implement or above their technical abilities. The Python package is suitable for the project as Python is widely regarded as an entry-level programming language, taking into account that the users may have little or no technical skills.

1.2. Project Aims

The goal of the project is to create a Python *package* to be used by students or academic professionals with the aim of visualising sets. *InteractiVenn*[1] is an existing Venn Diagram visualisation tool which provides interactive Venn diagrams that mirror certain aspects of the functionality required. As a result, *InteractiVenn* will serve as a standard for the Python package generated. Specifically the package should include the ability to allow users to input up to six sets of data to be visualised as an interactive Venn diagram with functionality that: Allows users to visualise the union and intersections of the given complex input sets, along with allowing users to interact with the generated output of the Python package.The Python package aims to provide users with a more accessible approach to set visualisation.

1.3. Initial Deliverables

The project was completed by achieving specific project landmarks, known as *Deliverables Requests[3]*, at set intervals over the project's lifecycle. The specific goals include the *Outline Document, Extended Abstract* followed by the *Final Project Report,* deliverables expected by University College Cork. Furthermore continuous development with help of bi-weekly appointments accompanied by the project supervisor, Dr. Rosane Minghim, ensured the completion of the project.

1.3.1. Outline Document - October 2021.

A short analysis of the project and a broad plan of the steps to complete the work, ensuring that the student understood the assigned project brief along with building a calculated structure of work needed to complete the project.

1.3.2. Extended Abstract - February 2022.

A summary of the essential work completed over the project's lifecycle. Ensuring that the student had gained valuable experience and technical knowledge as a result of the work completed on the project.

2. Analysis

2.1. Venn Diagram

Venn diagrams are an illustrative method of set visualisation, widely used to identify set relationships: such as set intersections as well as set unions. Venn diagrams were introduced in 1880 by *John Venn* in a paper entitled *On the Diagrammatic and Mechanical Representation of Propositions and Reasonings[4]*. Venn diagrams depict set elements as points on a visualised plane together with sets themselves as enclosed areas of space on the given plane. This fundamental property allows Venn diagrams to exceed at set visualisation.

2.2. Set Members

Elements, displayed as points, within a set, shown as an enclosed shape labelled *S,* represent the elements belonging to a set *S.* For set A if x is an element of A, read as "element x belongs to set *B*", this is written as:

$$x \in B$$

Elements excluded from the enclosed shape *S* are not elements belonging to the set *S.* For an element *y* that does not belong to set B,read as "element y does not belong to set *B*", this is written as:

$$y \notin B$$

The null set (*or empty set*) is the unique set with the property that it has no set members. Null set denoted by the symbols:

$$\phi \; or \; \{\}$$

2.3. Subsets

Set Intersection: given two sets, A and B, visually represented by *enclosed shape A* and *enclosed shape B*. The intersection of *set A* and *set B*, logically denoted by:

$$A \cap B = \{x: x \in and\, x \in B\}$$

that is the mutual elements of *set A* and *set B*, is effectively visualised by the overlapping areas of *enclosed shape A* and *enclosed shape B.*

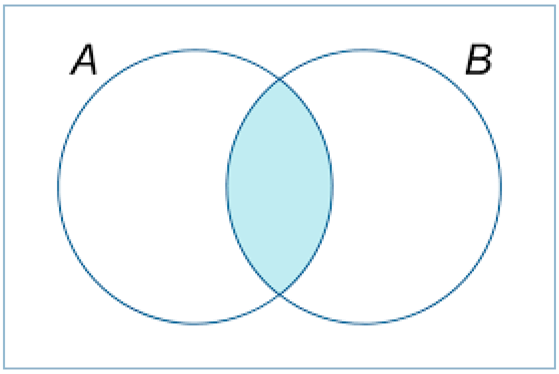


Figure 1: The intersection of A and B.

Set Union: Given two sets, *A* and *B*, visually represented by *enclosed shape A* and *enclosed shape B*. The union of *set A* and *set B*, logically denoted by:

$$A \cup B = \{x : x \in A \text{ } or \text{ } x \in B\}$$

are all elements entirely enclosed by the area of *set A* and *set B*.

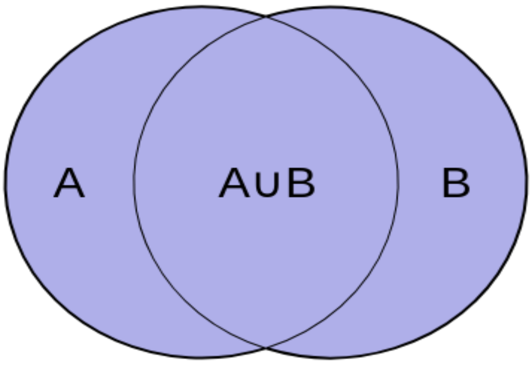


Figure 2: The union of A and B.

In Venn diagrams all possible logical relationships between sets of finite elements are visualised by all the overlapping areas of the plotted graph. Venn diagrams have been used to effectively demonstrate *set theory[5]* in a wide range of fields, including Computer Science.

2.4. Set Equality

Sets are a collection of mathematical objects, termed elements, of any type. Sets have two expected properties: Set order - the ordering of the elements in a given set which is not important, and set duplicates - sets containing the same element twice which is not allowed. A set may contain a finite or infinite number of elements. For two sets, *A* and *B*, *set A* and *set B* can only be regarded as equal if and only if the two sets are subsets of each other. Denoted by:

$$A \subseteq B \text{ } and \text{ } B \subseteq A$$

2.5. Current Set Visualisation Systems

2.5.1. InteractiVenn

InteractiVenn[1] consists of a web based interface for the visualisation and analysis of sets through Venn diagrams. Users can upload files, such as the model supplied for testing: *test_model.ivenn*, along with the ability to directly create sets of given elements. *InteractiVenn* requires the user to click the *Start* button, found under the navigation bar on the web interface in order to visualise their set input. Users can select to display up to six set relationships as an interactive Venn diagram model. Further functionality is included to allow users in real-time to hover over a given set, highlighting the set for improved visualisation.

2.5.2. UpSet

UpSet[2] expresses set relationships as a matrix of set intersections. *UpSet* is mentioned to be comfortably suited for inputs of up to 30 sets. Set, along with set size is displayed above the relationship matrix, the set relationship matrix itself is centred, with colour encoded spheres to indicate inclusion of elements in several sets. The set cardinality, a measure of the set's size, can be positioned either side of the set relationship matrix. UpSet provides an interactive web API allowing users to upload datasets, or choose from pre-existing datasets, for the purpose of testing and evaluation.

3. Design

3.1. Project Structure

The structure of the Python package created was developed continuously over the package's lifecycle. The Python package structure can be found at the public *Github repository for the Final Year Project* *[6]*. The base directory "/FinalYearProject" contains the source code and metadata belonging to the Python package. The contents of the Python package structure is listed below:

3.1.1. Extended Deliverables

The base directory regarding the expected work, from University College Cork, completed throughout the project's development. The expected deliverables can be found at Dr.Frank Boehme's *University College Cork's final year project website* *[7]*. This base directory "/Deliverables" contains four subdirectories:

OutlineDocument: Paper containing early project overview accompanied by the required steps to complete the project in the required development window.

Extended_Abstract: Compacted summary of the entire final year project's development, including work completed to date as well as future evolution needed if the project was to continue.

Open_Day: Directory containing a short project presentation, *FYP_video.mp4*, showcasing the final year project for use in *School of Computer Science and IT - Project Open Day 2022*. The open day was attended by fellow University College Cork students and professors along with commercial sponsors of the open day.

Final_Project_Report: The primary submittable report regarding the final year project. This includes all aspects of the project, a detailed description of each element of the work completed, concrete examples of the project's implementation and evaluation, as well as concluding remarks summarising the project's development.

3.1.2. FYP Drive

Directory containing detailed summaries of bi-weekly meetings hosted on Microsoft Teams, accompanied by the final year project's supervisor, Dr. Rosane Minghim. Furthermore a 'Deliverables_What_is_Success' file contains the discussion between the student and the project's supervisor on what is expected to be considered a successful final year project.

3.1.3. Report Tips

Directory containing constructive advice from the project's supervisor. This includes files supplied by the projects supervisor, such as *WritingTips_v8.pdf*, in conjunction with advantageous report writing information in regards to the final year project deliverables. Additionally the reports completed during the third year *Work Placement CS3300* module were consulted as a reference for report writing.

3.1.4. Research

Working directory of all experimental research completed over the final year project's development. This includes the source folder for the developed Python package along with sub-directories:

API research

Directory containing research conducted on Python web-based application programming interface's (API) examples implemented using the Python web framework *Flask[8]* , to implement and test a web-based API in Python. Following the research we concluded that the Python web-based API would not be a suitable direction to achieve success in the final year project.

Library research

Directory containing research conducted on Python package's as a technology for the final year project development. Included are sub-directories:

Examples: directory contains an example calculator Python packages used to experiment with the implementation as well as the download of a created Python package.

FYPlibrary: directory contains an experimental package structure that later influenced the final year project's Python package structure.

Upset

Directory for the experimental installation and testing of the Python package *UpSetPlot[9],* used as a Python implementation of the *UpSet[2] API. UpSet API* is described in the final year project's brief as a means to evaluate the Python package in contrast to a previously established set visualisation system.

Venn Diagram Research

Directory containing more advanced experimental research into a Python package for set visualisation. Contains experimental sub-directories along with the source directory for the final year project Python package created:

API: Source directory for the Python package created, *DrawVennDiagram[10]*. The source directory is expanded on in Chapter 3, section 3.2.

VennDiagramLibrary: Includes earlier development versions of the Python package created. During the experimental development in "/VennDiagramLibrary" a fundamental error occurred regarding Python package uploads to the *Python Package Index (PyPI)[11].* Concluding in a separate source directory */API* being created, hosting the current version of the package created.

3.1.5. TestAPI

Working test directory for the Python package created. Contains a *Jupyter Notebook[12]* testing file, 'FYP_TESTING.ipynb' used to import, from PyPI, the Python package created, to test the outputs generated and evaluate them against expected output. Additionally included is a sample data file retrieved from the *InteractiVenn[1]* web-based API, *"test_model.ivenn"*, used in testing and evaluating the Python package created.

3.1.6. Venv

Directory for the Python virtual environment*, venv[13],.* The virtual environment allows developers to isolate their working environment, enabling them to install packages locally, escaping the need to globally install packages that may cause system issues.

3.1.7. ReadME.md

Markdown file containing instructions on how to interact with a project, including concrete examples followed by expected outputs. A brief abstract is included giving a short overview of the project.

3.1.8. test_set.ivenn

Sample data file retrieved from the *InteractiVenn[1]* web-based API, *test_model.ivenn*, used to test and evaluate the Python package, conducted in the *Jupyter Notebook* test file, *FYP_TESTING.ipynb*.

3.2. Source Folder Structure

The Python package created source directory can be found at the public *Github repository for the Final Year Project [6]*. The base directory */API [14]* contains all source code and metadata related to the project's development.

3.2.1. Base Directory:

build: Directory created when the Python package is built. Contains the source code for the Python package.

distribution: Directory containing subdirectories of the numerous version releases of the Python package.

setup.py: Python file that is used to configure and build the Python package. States the *PyPI classifiers [16]*, containing information to categorise each Python package release which is needed to upload a built Python package to *PyPI*. States the PyPI set up data such as: Python package name, Python package version number, short description of the package, Python package author and author's email address, Python package licence as well as installation requirements for the Python package.

MANIFEST.txt: Text file consisting of commands, one per line, used in the building of the Python package which states what file types to include or exclude in the distribution (*dist*) directory.

LICENSES.txt: Text file confirming the copyright licences rights of the author for the Python package created.

3.2.2. Source Directory:

The source directory */DrawVennDiagram [15]* contains the source file. The *source file __init__.py [15]* contains the Python code for the package.

__init__.py: Source file for the Python package created. Contains the Python code of the package, this is expanded on in Chapter 3, section 3.2.3.

requirements.txt: Text file inside the source directory that allows the Python package to automatically install any dependencies required for the running of the Python package. This is a crucial step that will allow users of all programming skills to interact with the Python package created.

3.2.3. Source File:

The source file *__init__.py[16]* contains the Python code for the developed Python package. It comprises of several sections which are listed below:

Dependency import: At the beginning of the source file, the necessary dependency imports are made.

Constants: Constant values for the variables seen later in the source code. Variables which may not be set by the user are automatically initialised in this section.

drawVenn class: Python *Object-Oriented Programming* class that contains all the Python code in a *drawVenn class* that includes all necessary functions and encapsulation, including:

__init__ - Class initialising function

Necessary function in all Python Object-Oriented Programming classes to initialise the object. The parameters passed into the initialising function include the fundamental information needed for the Python package to operate.

__str__- Class string function

Overrides the Python string function to allow more control over the output of the Python package. The overridden string function automatically outputs the correct sized *Venn Diagram* based on the length of the object's *labels* variable.

Encapsulation functions

Getter function returns the value of a given variable. The setter function allows the user to adjust the object's variables through a function.

Inbuilt *read_data* function

Function built into the Python package that allows users to read in biological data files, with a similar format to the "*test_model.ivenn*" used for the testing of the Python package. The Python package initially only accepted integer arguments as set elements. This became an issue when the biological data set elements consisted of both integers as well as strings. The *"read_data"* function solves this issue by treating each set element as a string, then uniquely mapping each string to an integer representation. This unique integer representation is then used as an argument to the Python package therefore allowing the visualisation of biological sets.

3.3. Technologies

3.3.1. Programming Language - Python:

Python is a high level, general purpose programming language. Python is widely regarded as an accessible programming language to users of all programming skill levels. Python is therefore suitable for this final year project as the aim is to create a simple and user-friendly Python package for set visualisation.

3.3.2. Package Hosting:

PyPI: Python Package Index [11], offers developers the ability to host their Python package The Python package is uploaded to PyPI by the developer, using the tools mentioned below in Chapter 3, section 3.3.3 . The user of the project can import the Python package using the Python *pip[18]* package installer for Python or from the PyPI website.

Test PyPI [19] is a separate instance of PyPI. Test PyPI used to privately upload project versions and test importation prior to releasing a given public version on PyPI.

3.3.3. Project Tools:

Project structure hosting - GitHub

GitHub [20] is a version control system that allows developers to track the evolution of projects over time, throughout different versions using *Git[21]*. *GitHub* allowed the storing of the project structure, mentioned in Chapter 3, section 3.1, on the cloud in a public repository. This is advantageous as the project's life cycle was over the span of six months therefore uploading iterations of the project to *GitHub* using *Git* ensured that progress would be securely stored. *GitHub's* version control also allowed free development of the Python package with the protection that if a direction of development led to system issues, then the project could be easily reverted to an earlier state.

Virtual Environment - venv:

The Python virtual environment*, venv[13],.* allows developers to isolate their working environment, enabling developers to install packages locally in the working environment, escaping the need to globally install packages that may cause system issues.

3.3.4. Project Dependencies

Matplotlib:

Matplotlib is an object-oriented Python package for generating and outputting visual representations of data. *Matplotlib* classes *venn2* along with *venn3* were used to efficiently create two and three set Venn diagrams.

Matplotlib-venn:

Matplotlib-venn is a Python package that builds on the foundations of *Matplotlib* allowing developers to generate Venn Diagram outputs.

3.3.5. Package Upload & Build:

Wheel

Wheel is a Python package used for installation and management of Python packages. *Wheel* possesses similarities to the *ZIP* format however it is more applicable to Python projects.

Setuptools

Setuptools is a Python package that allows developers to easily build and deploy their packages.

Twine

Twine is a tool for developers to publish their Python packages to *PyPI*. Twine takes as an argument the Python package version from the "/dist" directory mentioned in Chapter 3, section 3.2.1.2 , uploading the given verison to PyPI.

4. Implementation

4.1. Functional Arguments

The Python package created expects essential input parameters in addition to optional input parameters, in order to operate. A functional argument or function input, is a non-local value that must be passed to the function in order to be used during the execution of the function.The functional argument can be of any data type. In the Python programming language a function can be passed an argument using the following syntax:

```
def function(expected_argument):
   <<function code>>
```

Figure 3: creation of a function with an expected argument

```
function(argument)
```

Figure 4: calling of the function created above

The Python package created expects Python set objects as function arguments to the drawVenn class. Python set objects are unordered collections of objects, set elements, in the package created the set object data type expected is of type Python integer. Python set elements are objects used to form a set object. Python integers are positive or negative whole numbers without a fractional component, including zero. The set objects contain the set element data that is to be visualised through the package created.

Additional expected function arguments include set labels, of Python tuple data type, that indicate the set name to generate as output. Python tuples are one of the four built in data types in Python, used to store multiple items in a single variable. A tuple is a collection that is ordered and unchangeable. The package created expects tuples containing Python strings. Python strings are a sequence of characters, enclosed in single or double quotation marks. A Python character is a single symbol, however Python does not provide a data type to directly store a character, instead a single character is stored as a string of length one.

Users of the package created can select between a weighted and unweighted visual representation of the set data. Users selected their preferred visual representation by assigning an optional functional argument, labelled unweighted, to either the boolean values *True*, generating an unweighted Venn diagram, or *False*, generating a weighted Venn diagram.

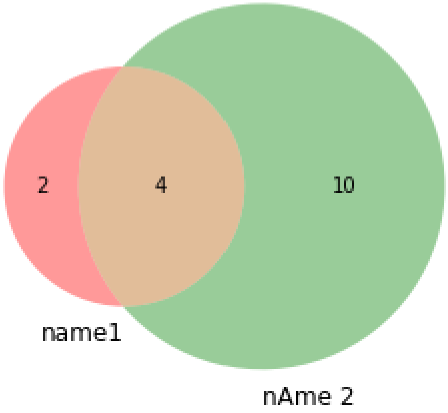


Figure 5: Output generated by setting *unweighted* parameter to *True*

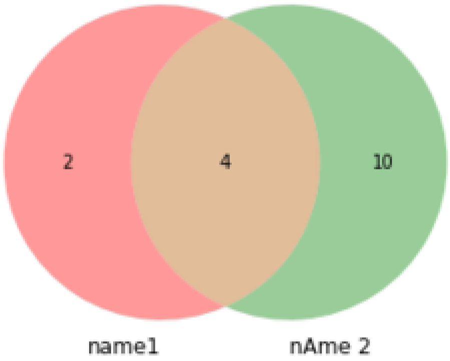


Figure 6: Output generated by setting *unweighted* parameter to *False*

Furthermore users of the package created can modify the colouration of the Venn diagram output generated. This modification is done by passing a Python tuple, consisting of Python strings, with values indicating the desired colouration of the generated Venn diagram.

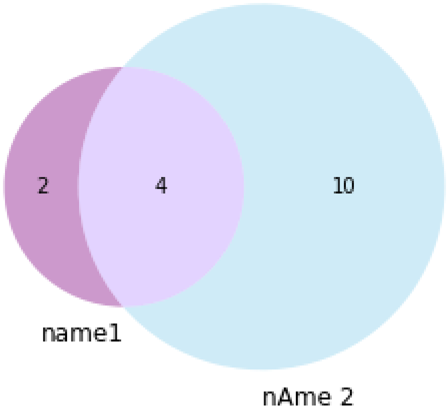


Figure 7: Output generated by setting the *colouration* of the Venn diagram to

4.1.1. In-built *read_data* function:

As mentioned above, in section 4.1, the package created expects a Python set object as a functional argument. The package expects set elements of Python integer data type. This initial fundamental property of the package resulted in the creation of the in-built package function, read_data, that expects a Python file input, for example the test file test_model.ivenn. This test file consists of a series of set objects, with each line of the file representing a single set. Each line of the file consists of a set label followed by the set elements. The set label and set elements, both of Python string data type, are separated by a colon, along with each set element separated by a comma.

The read_data function takes the file argument, opens the file using the Python method, with open(file) as f, to open and close the file, ensuring the file is closed in the correct manner if an exception was to prevent the input flow. Once the file input is open, represented by the variable f, contents are read to the function using the Python readlines method. Python's readlines method returns the entire input file's contents as an iterable list, with each line of the input file being an index of the iterable list object. The iterable list object can be iterated over using a Python for loop, with the input set label data being appended to a Python list variable, *labels*, along with the input set elements being appended to a Python list variable, *values*.

Once the raw input data, labels and elements, have been extracted, as Python string variables, and subsequently stored in the list variables created above. The raw input data, of type Python string, is required to be converted to data of type Python integer, to function as input to the Python package created, which only accepts input data of type integer.

The conversion consists of steps listed below:

1. **Clean the raw data**

   Firstly the raw input data is cleaned, removing the unimportant characters such as; colon and comma separators.

2. **Flatten the two-dimensional**

   Secondly the cleaned input data is converted from its original two-dimensional Python array format, to a one-dimensional Python array. This crucial step is to ensure that each set element value is uniquely mapped to an integer, as well as ensuring that given a raw set element value, x, will consistently be mapped to a given integer, y.

$$String\ element\ x\ \rightarrow\ Integer\ y$$

The *read_data* function uses a Python list compression, containing two nested Python loops, to create the larger one-dimensional array from the contents of the two-dimensional array.

3. Uniquely map string variables to integer variables

A Python dictionary is used, along with a list comprehension, to map each unique Python string element value. The flattened one-dimensional Python array is converted to a Python set object, this is to remove any duplicate element values.

```
set(flat_values)
```

The resulting set object is then sorted using the Python *sorted* function.

```
sorted(set(flat_values))
```

Once the set object is sorted, a Python *enumerate loop* is used to assign each raw input element value an integer representation.

```
for x,y in enumerate(sorted(set(flat_values)))
```

Finally to have the mapped integer values begin from the value one, rather than the default of zero, we must add a value of one to all index values.

```
x+1
```

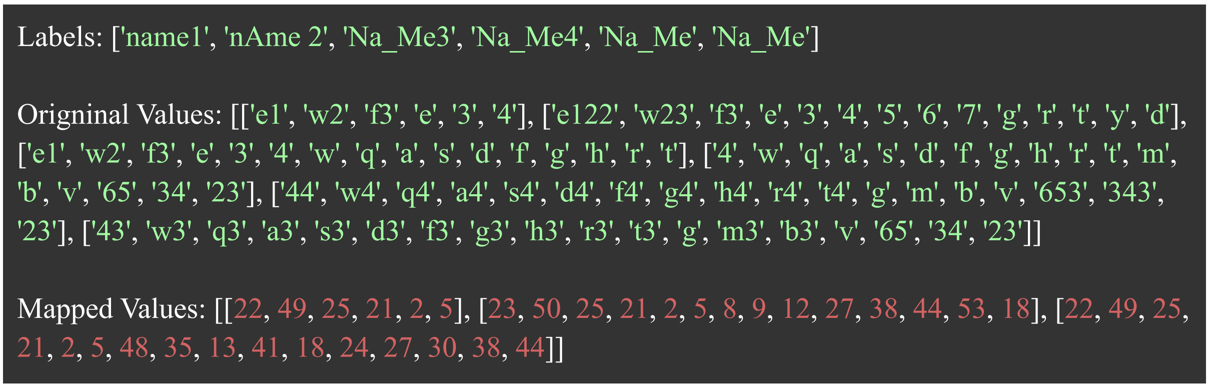
4. Slice the data to its original two-dimensional form

Following the above steps, the data is now a cleaned one-dimensional integer representation of the original raw string values. To return the data to a two-dimensional Python array format, a Python list comprehension is used in conjunction with Python list slicing to separate the mapped integer values, returning them to the original two-dimensional format and storing them in a Python dictionary.

```
dict([(y,x+1) for x,y in enumerate(sorted(set(flat_values)))] )
```

5. **Return the list variables**

   The function returns three Python lists; labels, mapped_values and values. *Labels* contain the extracted string set labels from the input file. *Mapped_values* contain each set's integer representation to be used for input to the Python package created. *Values* contain the original raw string set element values.



```
Labels: ['name1', 'nAme 2', 'Na_Me3', 'Na_Me4', 'Na_Me', 'Na_Me']

Origninal Values: [['e1', 'w2', 'f3', 'e', '3', '4'], ['e122', 'w23', 'f3', 'e', '3', '4', '5', '6', '7', 'g', 'r', 't', 'y', 'd'], ['e1', 'w2', 'f3', 'e', '3', '4', 'w', 'q', 'a', 's', 'd', 'f', 'g', 'h', 'r', 't'], ['4', 'w', 'q', 'a', 's', 'd', 'f', 'g', 'h', 'r', 't', 'm', 'b', 'v', '65', '34', '23'], ['44', 'w4', 'q4', 'a4', 's4', 'd4', 'f4', 'g4', 'h4', 'r4', 't4', 'g', 'm', 'b', 'v', '653', '343', '23'], ['43', 'w3', 'q3', 'a3', 's3', 'd3', 'f3', 'g3', 'h3', 'r3', 't3', 'g', 'm3', 'b3', 'v', '65', '34', '23']]

Mapped Values: [[22, 49, 25, 21, 2, 5], [23, 50, 25, 21, 2, 5, 8, 9, 12, 27, 38, 44, 53, 18], [22, 49, 25, 21, 2, 5, 48, 35, 13, 41, 18, 24, 27, 30, 38, 44]]
```

Figure 8: Variables created by *read_data* function

4.2. Research Directory

Working directory of all experimental research completed over the final year project's development. This includes the source folder for the developed Python package along with sub-directories:

4.2.1. UpSetPlot - Research

Directory containing research completed on a Python package implementation of *UpSet[2]* web-based application. A *Jupyter Notebook* was created to implement the UpSetPlot documentation to study the expected outputs of a similar Python package system. Furthermore careful research of the UpSetPlot's GitHub repository assisted with the construction of the package structure.

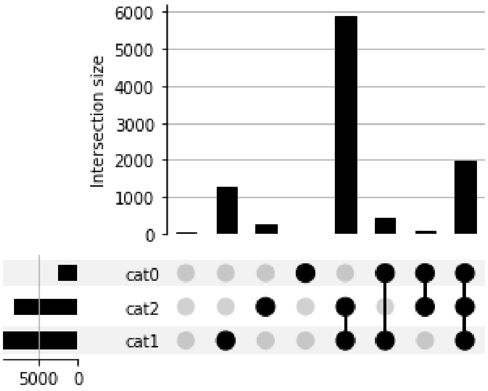


Figure 9: UpSet output generated by UpSetPlot package

4.2.2. Venn diagram - Research

Directory containing three sub-directories, that were used throughout the project's development as test cases for the implementation of the final package created, sorted in the *API* directory. *API* directory contains the Python package published to PyPI. The structure of the package is expanded upon in Chapter 3, section 3.1.

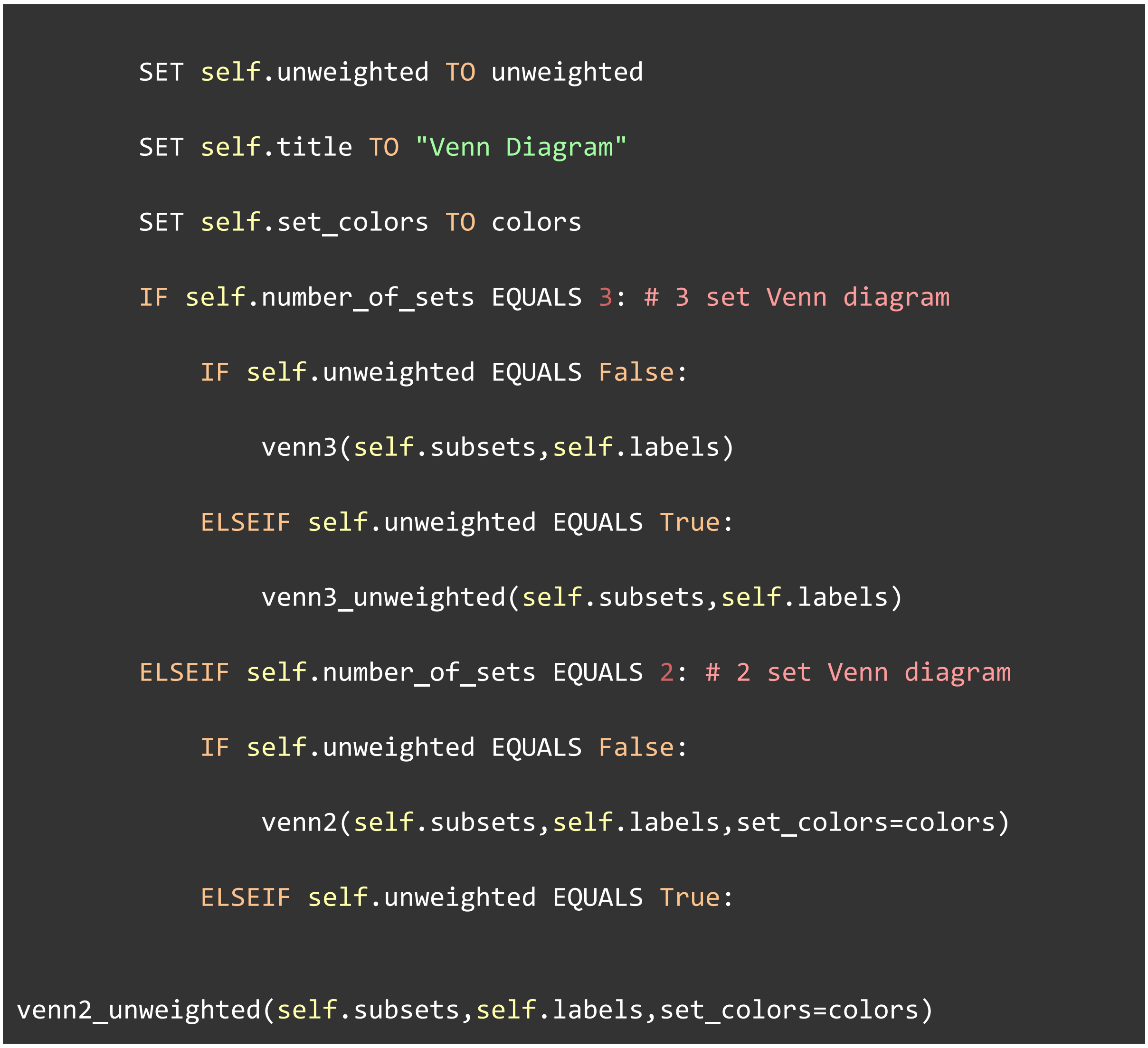
```
DEFINE CLASS drawVenn:

    DEFINE FUNCTION
__init__(self,subsets,labels,unweighted=False,colors=('r','g')):

        SET self.subsets TO subsets

        SET self.labels TO labels

        SET self.number_of_sets TO len(labels)
```



```
        SET self.unweighted TO unweighted
        SET self.title TO "Venn Diagram"
        SET self.set_colors TO colors
        IF self.number_of_sets EQUALS 3: # 3 set Venn diagram
            IF self.unweighted EQUALS False:
                venn3(self.subsets,self.labels)
            ELSEIF self.unweighted EQUALS True:
                venn3_unweighted(self.subsets,self.labels)
        ELSEIF self.number_of_sets EQUALS 2: # 2 set Venn diagram
            IF self.unweighted EQUALS False:
                venn2(self.subsets,self.labels,set_colors=colors)
            ELSEIF self.unweighted EQUALS True:

venn2_unweighted(self.subsets,self.labels,set_colors=colors)
```

Figure 10: Pseudocode demonstration of the package

Now that we have the folder structure and source code created, we can explore the process of uploading a Python package to PyPI.

4.3. Hosting of Package

4.3.1. PyPI

Once the folder structure and source code, shown in section 4.2 above, is created. We must build the Python package for publication on the PyPI website, to allow for users and/or developers to easily view and install the package. To build the package we ensured we were in the current working directory of the Python package, *API/*. Next we must run the *Linux* terminal commands shown below in Figure 11.

```
python setup.py bdist_wheel
twine upload dist/version_number
```

Figure 11: Linux terminal commands to build and upload Python package to PyPI

The package can be downloaded and installed conveniently from the PyPI repository using the Linux terminal commands shown below in Figure 12.

```
pip install DrawVennDiagram
from DrawVennDiagram import *
```

Figure 12: Linux terminal command to download and install a Python package from PyPI.

4.3.2. GitHub

Project versions were continuously published to *GitHub* in order to store project states throughout the project's lifecycle. This crucial process is necessary to ensure the project's code as well as additional files are kept protected and backed-up in case of an emergency. The process of publishing the project to the *GitHub* remote repository is as follows: First initialise repositories, both local and remote, to store the project. Enable the local virtual environment, *venv*, through the Linux terminal. Pull the most recent project version from *GitHub*. Output or update the current local repository status. Commit any current changes to the project's state. To allow the uploading of project versions directly to a *GitHub* account, Personal Access Tokens were used to verify the authentication of the client. The Personal Access Tokens are generated in the *GitHub* website by the project author. Publish committed changes to the public *GitHub* remote repository.

4.3.3. Python Package Documentation

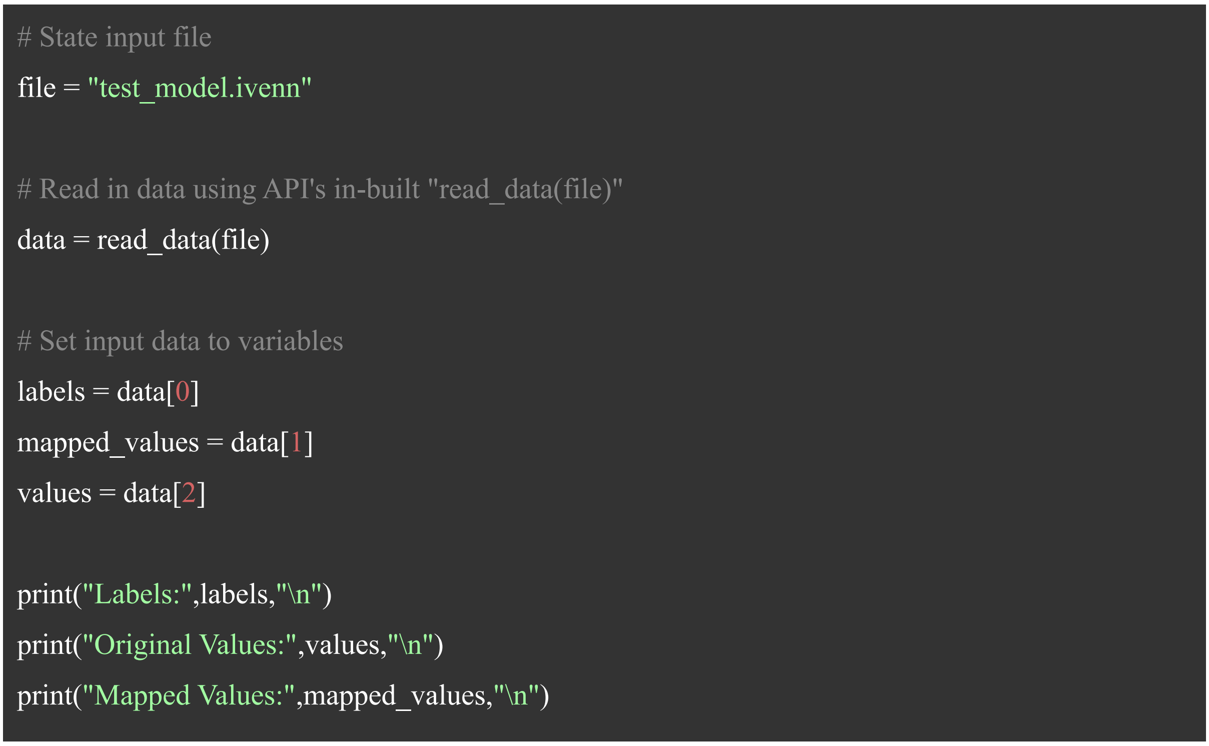
Development of a Python package includes the development of clear and concise documentation. The aim of package documentation is to explain the contents of the package created along with concrete examples and expected outputs that the package should generate, to ensure users of the package can understand the correct way to interact with the package.The documentation for the Python package created is stored in the *README.md* markdown file hosted on the project's *GitHub* remote repository.

5. Evaluation and Results

5.1. Testing of Package

For the purpose of testing the Python package, we developed a *Jupyter Notebook* to provide an intuitive, interactive testing interface for users to explore.
The *Jupyter Notebook* initially presents the user with a short abstract description of the Python package created along with information regarding the author of the package and the project's supervisor. The *Jupyter Notebook* contains a series of Python code cells followed by the generated output from the Python package.

Once the package is downloaded and successfully installed, you can begin using the package. The *Jupyter Notebook* uses a testing file, *test_model.ivenn*, as the input data to be visualised. The testing files contents are read into the *Jupyter Notebook* using the in-built *read_data* function, mentioned above in Chapter 4, section 4.1.1.



```
# State input file
file = "test_model.ivenn"

# Read in data using API's in-built "read_data(file)"
data = read_data(file)

# Set input data to variables
labels = data[0]
mapped_values = data[1]
values = data[2]

print("Labels:",labels,"\n")
print("Original Values:",values,"\n")
print("Mapped Values:",mapped_values,"\n")
```

Figure 13: Python code for reading input file

Instructions on how to interact with the Python package created are listed below in order:

Step 1: Create Python set objects with mapped values as inputs.

Step 2: Create a set tuple variable, which contains the set objects created in *Step 1*.

Step 3: State the labels you wish to display, as a tuple of strings.

Step 4: Create a *drawVenn* object, passing the set tuple. from *Step 2*, along with the labels stated in *Step 3* as functional arguments.

Step 5: Display the *drawVenn* object.

5.2. Appraisal of Completed System

The primary goal of the Python package created, in accordance with the final year project's description, was to produce a user-friendly Python package to visualise the correspondences between sets, such as unions and intersections. The result of the project is a Python package that satisfies this requirement for input sets of up to three sets, with development of four set inputs requiring further development mentioned in Chapter 6, section 6.2 below.

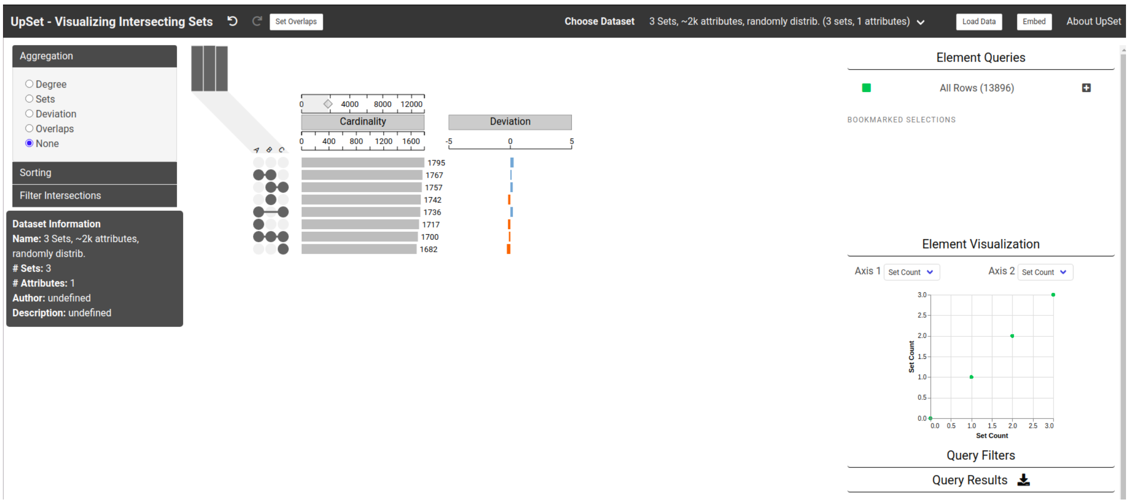
Following continuous bi-weekly meetings accompanied by the final year project's supervisor, Dr. Minghim, it was concluded that the primary target audience of the Python package created would include Biology scientists, including currently enrolled students along with working professionals. The Python package created, took this conclusion into account with reference to the *read_data* in-built function created, mentioned in Chapter 4, section 4.1.1. The *read_data* function automatically converts string data type set element values to mapped integer values, allowing the string inputs to be efficiently visualised using the Python package.

An essential component of the final year project involved the evaluation and comparison of the Python package created in regards to current set visualisation systems. This evaluation of such systems will be discussed below in Chapter 5, section 5.3.

5.3. Evaluating current Set Visualisation Systems

5.3.1. UpSet

For the testing of the *UpSet[2] API* an existing dataset was used, three sets*,* to effectively portray an expected output from the *UpSet API*. The dataset was chosen as it can be directly compared with the Python package created.



Choose Dataset 3 Sets, ~2k attributes, randomly distrib. (3 sets, 1 attributes)

Dataset Information
Name: 3 Sets, ~2k attributes, randomly distrib.
# Sets: 3
# Attributes: 1
Author: undefined
Description: undefined

Figure 14: Output generated from *UpSet* using three set inputs

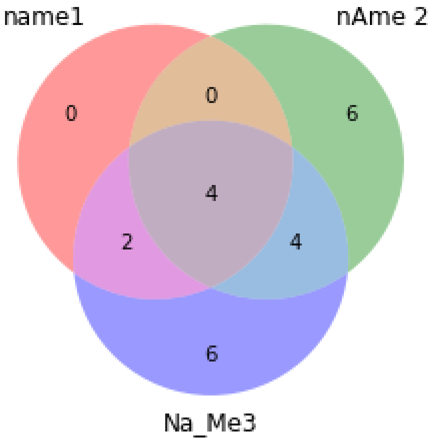


Figure 15: Output generated from package using three set inputs

As seen from the examples above, Figure 14 and Figure 15, for data sets of a low size, up to 3 sets, it is much more efficient to visualise the data through the Python package created. UpSet does produce outputs for sets.

5.3.2. InteractiVenn

For the testing of the *InteractiVenn[1]* the provided, *test_model.ivenn*, testing file acted as a control regarding the testing of both: *InteractiVenn[1]* as well as the package created. More detailed description of the dataset used can be found in Chapter 3, section 3.1.8.

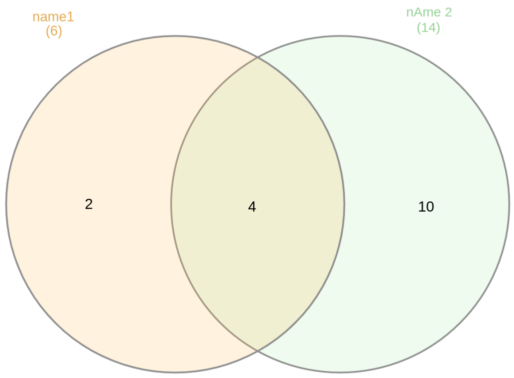


Figure 16: Output generated from *InteractiVenn* using two set inputs

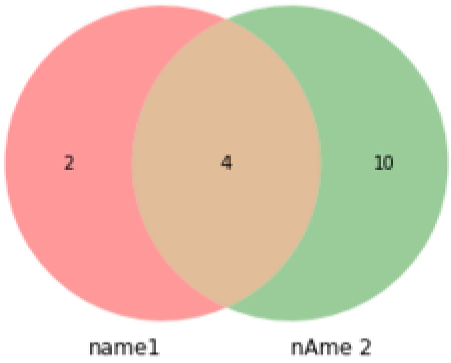


Figure 17: Output generated from package using two set inputs

As seen from the examples above, Figure 16 and Figure 17, *InteractiVenn[1]* produces a highly similar output to the package created. Where *InteractiVenn[1]* excels is in the expanding of input sets, up to six sets.

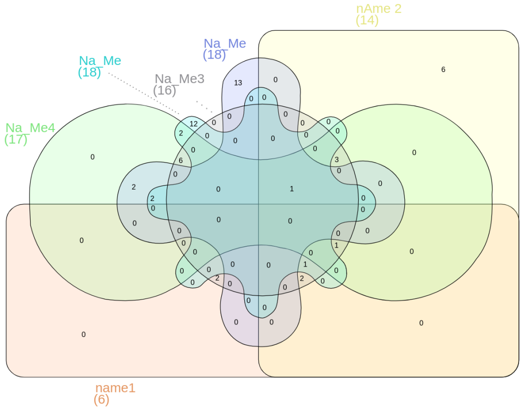


Figure 18: Output generated from *InteractiVenn* using six set inputs

This magnitude of inputs is currently not supported by the package created. Continued development of the package should address this weakness.

6. Conclusion

The result of the project is a Python package, *drawVenn*, created with the ability to visualise sets using *InteractiVenn's* approach as a foundation. Evaluation was completed comparing the package created with an existing similar visualisation system, *UpSet*. It is clear from the work completed that both systems have diverse approaches to a similar problem, set visualisation. *UpSet's* approach is efficient for graphically displaying set intersections of input set size above a trivial threshold, approximately six sets. The package created builds on the approach taken by *InteractiVenn*, displaying set relationships as a more conventional Venn diagram, however it takes advantage of the ease of use of the Python programming language.

6.1. How the Project was Conducted

The project's development was completed over two stages; beginning in semester one and completed in semester two. The first stage, semester one, involved careful evaluation of similar systems to the project as well as outlining the structure and hosting of the package. This stage contained the majority of the test implementations for package uploading, downloading, as well as testing the importing of a Python package.

Come semester two, the project had moved on to the implementation of the final versions of the package. Here the project was finalised and the testing stage could commence. Here, a *Jupyter Notebook* was created to organise the testing of the package in a clear and concise manner.

The project was finalised towards the end of semester two, this stage involved generating the *Final Project Report* deliverable, preparation of the project for the *Computer Science FYP Open Day* hosted on the 6th of April. 2022, along with final touches made to complete the project.

6.2. Future Work

Following the development of the Python package, the area which immediately requires further development is increasing the limit of set input sizes from the current three sets to a maximum of six sets. This package does not currently support such functionality. The issue lies with the assignment of set element values to more complicated graphical representations required to display four to six set Venn diagrams. A potential solution would be to assign labels to the unique areas of these larger Venn diagrams, which would also aid in the additional functionality mentioned below.

Following this, development of additional functionality, such as adding interactivity to the package to allow users to select set relationships and display the elements enclosed in a given set relationship. The issue lies with the use of the less maintained *Matplotlib-venn* as an output generation dependency. There does not seem to be a direct approach to adding interactive functionality to such an output graph. A potential solution would take advantage of the *x* and *y* coordinates of the output graph to understand the specific section the user wishes to interact with.

6.3. Reflection

The result of the development of the Python package demonstrated the effort required in order to produce an acceptable programming package. Before any implementation can begin, countless hours of research must be completed in order to understand the requirements of a programming package. It is critical to assess similar systems in order to understand the expected outcomes as well as user experience. Development of the project has expanded my knowledge for both the Python programming language along with programming packages and their clear advantages such as ease of download and importation. It was fascinating to experience the power a programming package can bring to computer literate users as well as users of less technical skill.

I would like to also mention the tremendous collaboration between myself and the project supervisor, Dr. Rosane Minghim. The continuous development and engagement from Dr. Minghim helped me keep on track with the development of the project over the months.The regular meetings and demonstrations of work completed to date, along with Dr. Minghim's professional suggestions assisted in the successful completion of the final year project.

7. References

1. Heberle, H.; Meirelles, G. V.; da Silva, F. R.; Telles, G. P.; Minghim, R. InteractiVenn: a web-based tool for the analysis of sets through Venn diagrams. BMC Bioinformatics 16:169 (2015). DOI: 10.1186/s12859-015-0611-3, http://www.interactivenn.net/
2. Alexander Lex, Nils Gehlenborg, Hendrik Strobelt, Romain Vuillemot, Hanspeter Pfister. UpSet: Visualization of Intersecting Sets IEEE Transactions on Visualization and Computer Graphics (InfoVis), 20(12): 1983--1992, doi:10.1109/TVCG.2014.2346248, 2014. https://upset.app/
3. Deliverables Requests listed on University College Corks final year project website, 2022, https://project.cs.ucc.ie/deliverable/index.
4. J. Venn M.A. (1880) I. On the diagrammatic and mechanical representation of propositions and reasonings , Philosophical Magazine Series 5, 10:59, 1-18, DOI: 10.1080/14786448008626877 To link to this article: http://dx.doi.org/10.1080/14786448008626877
5. Set theory Wikipedia page, 2022, https://en.wikipedia.org/wiki/Set_theory#Formalized_set_theory
6. GitHub 2022 remote public repository for Nathan Crowley final year project, https://github.com/NathanCrowley/FinalYearProject
7. Dr.Frank Boehme of University College Cork,2022 final year project website, https://project.cs.ucc.ie/deliverable/index
8. Grinberg, M. (2018). Flask web development: developing web applications with python, https://flask.palletsprojects.com/en/2.0.x/quickstart/
9. Alexander Lex, Nils Gehlenborg, Hendrik Strobelt, Romain Vuillemot, Hanspeter Pfister, UpSet: Visualization of Intersecting Sets, IEEE Transactions on Visualization and Computer Graphics (InfoVis ‘14), vol. 20, no. 12, pp. 1983–1992, 2014. doi: doi.org/10.1109/TVCG.2014.2346248, https://upsetplot.readthedocs.io/en/latest/
10. DrawVennDiagram 0.6.0, https://pypi.org/project/DrawVennDiagram/ , hosted on Python Package Index - PyPI. (n.d.). Python Software Foundation. Retrieved from https://pypi.org/
11. Python Package Index - PyPI. (n.d.). Python Software Foundation. Retrieved from https://pypi.org/
12. Kluyver, T., Ragan-Kelley, B., Fernando P&#x27;erez, Granger, B., Bussonnier, M., Frederic, J., … Willing, C. (2016). Jupyter Notebooks – a publishing format for reproducible computational workflows. In F. Loizides & B. Schmidt (Eds.), Positioning and Power in Academic Publishing: Players, Agents and Agendas (pp. 87–90).

13. Venv 3.3 - Creation of virtual environments, Python programming language (2021), https://github.com/python/cpython/tree/3.10/Lib/venv/
14. *Github repository for the Final Year Project base directory /API,* https://github.com/NathanCrowley/FinalYearProject/tree/main/Research/VennDiagram_research/API
15. *Github repository for the Final Year Project source directory /DrawVennDiagram, https://github.com/NathanCrowley/FinalYearProject/tree/main/Research/VennDiagram_research/API/build/lib/DrawVennDiagram*
16. *Github repository for the Final Year Project source file, https://github.com/NathanCrowley/FinalYearProject/blob/main/Research/VennDiagram_research/API/build/lib/DrawVennDiagram/__init__.py*
17. *Python package index, Classifiers, https://pypi.org/classifiers/*
18. Python pip 22.0.4, https://pypi.org/project/pip/
19. Test Python package publishing with the Test Python Package Index, https://test.pypi.org/
20. GitHub - https://github.com/
21. Git - https://git-scm.com/