Modelling the effects of domestication in Wheat through novel computer vision techniques

Author:	Mr. Nathan Hughes (nah26@aber.ac.uk)
Supervisor:	Dr. Wayne Aubrey (waa2@aber.ac.uk)
Degree Scheme	G401 (Computer Science)

Date:	April 25, 2018
Revision:	0.1
Status:	Draft

This report was submitted as partial fulfilment
of a BSc degree in Computer Science (G401)

Declaration of originality

I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Quality and Records Office (AQRO) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work I understand and agree to abide by the University's regulations governing these issues.

Name ....

Date ....

Consent to share this work

By including my name below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name ....

Date ....

Acknowledgements

This work is a product of all the things Aberystwyth University has taught me, unfortunately there is not enough space here for everyone who should be thanked and acknowledged.

However, in relation to this project it would be a great crime to not thank my project supervisor Dr. Wayne Aubrey for his help and guidance; Professor. John Doonan, who provided the project and trusted me with it; Dr. Candida Nibau who taught me practically everything I know about biology; Dr. Kevin Williams for allowing all my questions about maths and statistics;

My headphones, my books and all the art that kept me sane. Without J.K. Rowling, Arthur Conan-Doyle, Ben Howard and Bob Dylan my motivation would have dried up long ago.

Finally, and most importantly, my mother Barbara Hughes. Who believed in me, always.

Contents

List of Tables

List of Figures

List of Equations

List of Listings

Chapter 1

Introduction, Analysis and Objectives

This project aims to answer a biological research question through the use of computer science, whilst also creating a software suite which will enable further studies to be carried out with ease.

Primarily the focus has been on the data science elements of my degree, creating, cleaning and discerning meaning in it.

Using a population of genetically diverse wheat, several hypothesis and questions are explored in the hopes of contributing to the scientific understanding of domestication. A mixture of image analysis through three-dimensional micro-computed tomography and computational analysis are used to provide these much needed solutions.

Additionally, as this is very much multi-disciplinary research, specific terms and definitions have been outlined in the *glossary* (table:B.1).

Background

2 Western society and agriculture has been dominated by the ability to create successful crops for the past 10,000 years [2]. Of these crops wheat is considered to be one of the most vital and is estimated to contribute to 20% of the total calories and proteins consumed worldwide, and accounts for roughly 53% of total harvested area (in China and Central Asia) [3].

During domestication, the main traits selected for breeding were most likely plant height and yield. This meant that important non-expressed traits such as disease resistance and drought tolerance were often neglected and lost overtime.

Whilst the choices made for selective breeding were successful, effects are now being felt as it is estimated that as much as a 5% dip is observed yearly on wheat production [3]. This decrease in efficiency is attributed to climate change bringing in more hostile conditions, which these elite and domesticated genotypes are unprepared for.

Furthermore, with increasing populations and less arable land there is an even greater pressure for the optimisation of grain and spike characteristics. With studies showing that spikelet, the collective for seeds sharing the same node on a spike, count can be controlled by specific and sometimes recessive genes [4], which could drastically enhance overall yield, and a general public distrust towards genetically modification [5, 6, 7] the reliance on breeding programs for optimisation is further stressed.

Modern breeding programs have had some success in selecting primitive undomesticated genotypes and using them to breed back in useful alleles which would have been lost during domestication [8].

As such, there are questions still left open about how best to make selections for crop breeding. There is also a lack of formalised modelling of information which could be of use to these areas of research.

Biological Question and Materials

The driving question for this research asks "Can µ-CT data be used to model domestication in wheat?". Using an already grown and harvested range of genetically diverse wheat this project has generated a collection of 3D images, processed these images into raw phenotypic data and produced biologically significant information.

The genotypes used in this study are listed here, denoted by "*X* N" where *X* indicates the ploidy. 2N - Diploid; 4N - Tetraploid; 6N - Hexaploid. A *W* indicates wild genotypes, *D* domesticated ones.

- *T. boeoticum* (2N|W)
- *T. monococcum* (2N|D)
- *A. Tauschii* (2N|D)
- *T. durum* (4N|D)
- *T. dicoccum* (4N|D)
- *T. dicoccoides* (4N|W)
- *T. ispahanicum*(4N|D)
- *T. timopheevii* (4N|D)
- *T. spelta* (6N|D)
- *T. aestivum* (6N|D)
- *T. compactum* (6N|D)

Full species names are found in table:C.1.

Why use µ-CT image analysis?

In the past, science has been greatly limited by the amount of data which could be processed in an experiment. In the last few decades the inclusion of computer science has reduced this bottleneck. Now, the challenge for many fields of research is producing more data and this is often cited as the new limiting factor in creating robust studies [9].

Many experiments aim to meet the demand for data by using high-throughput automated imaging systems [10, 11, 12]. These systems have, in the last decade, become a standard and accepted tool for data generation. However, they will only produce 2-dimensional data on a per-plant basis. Image processing research has had success in modifying these automated systems in order to produce a pseudo 3-dimensional structure using stero-imaging [13]. Even so, these techniques require destructive harvesting of materials and do not provide information of internal structure.

For decades medical research has found success with X-Ray imaging technology [14]. From this, plant science has been able to benefit from the wealth of prior knowledge and more and more studies are being augmented with the use of X-Ray/µ-CT imaging [1, 15, 16, 17, 18].

In this study, µ-CT has enabled the study of individual seeds of wheat, which is the product that plant breeders, commercial growers and farmers are truly interested in. Other imaging techniques could not provide as much detail, whilst retaining developmental and positional information, or in such a high throughput or quality.

Extracted Data

These samples come from over 70 plants and provided in excess of 2000 seeds for analysis which data was created based on. The traits recorded are labelled in figure:1.1 and are as follows:

- Length
- Width
- Depth
- Volume
- Surface Area
- Crease Depth / Volume

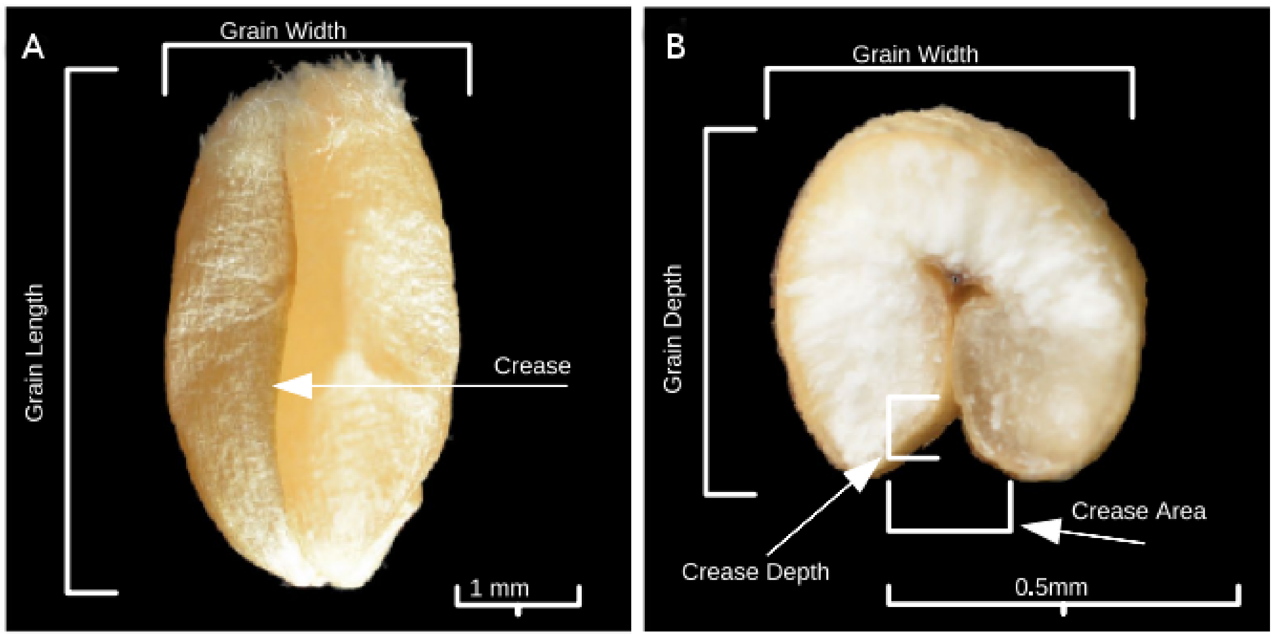


Figure 1.1: Wheat grain labelled (*left*), wheat grain cut in half (*right*), adapted from Hughes et al. [1]

Significance to Current Research

The biological interest in this area has been expressed in several areas of research [19], it is proposed that the key to unlocking diversity in the wheat genus lies in these ancestor, undomesticated species [20].

This research has the potential to be useful in several areas including: crop breeding; disease resistance; environmental stress. Each of these areas depend on making informed decisions in order to direct experiments. By producing information at an individual seed level, this study has been able to provide data that can offer suggestions of plant potential and behaviour.

Often, the most sought after traits are centred around thousand-grain-weight (TGW) as well as standard deviation of seed shapes. During harvesting, filters are used to only allow ideal shaped seeds through. This means that, potentially, despite a breed of wheat providing a high average volume of seed in reality much of it may go to waste if the shapes are not uniform. This research aims to alleviate this problem and provides low level information which is sorely required.

The individual images in figure:1.2 show, at a glance, the diversity and also the difference in the wild and cultivated (domesticated) species. This work allows for these differences to be quantified and evaluated into useful metrics for answering research based questions.

By better understanding the morphometric deviations in wheat species, more informed choices can be made when it comes to breeding wheat for the future and to fulfil ever-changing requirements.

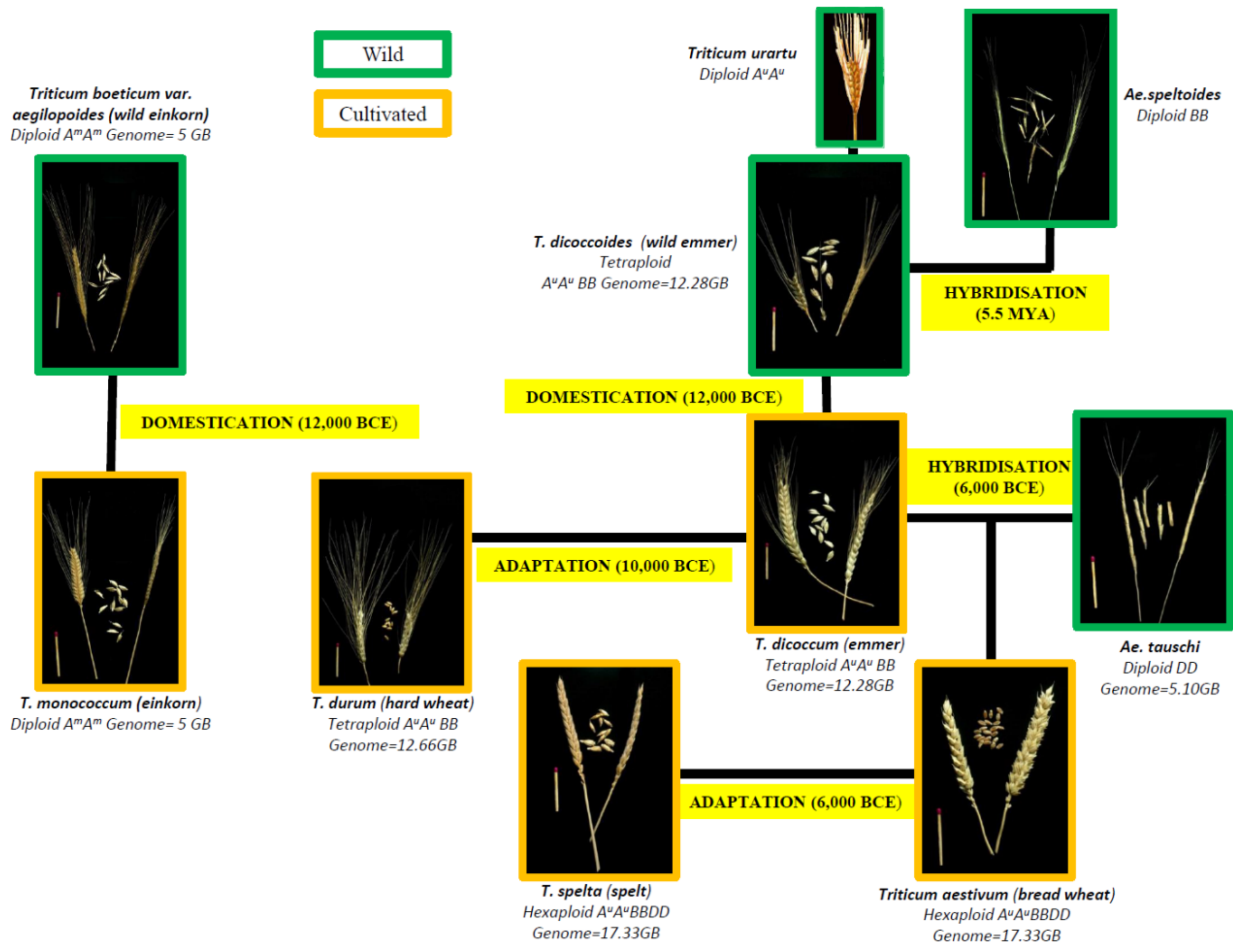


Figure 1.2: Phylogeny of wheat genotypes (Provided by Dr. Hugo Oliveira)

Aims and Objectives

The overarching aim of this project has been to create several pieces of software which aid in answering the biologically significant questions outlined. As well as to prove/disprove the hypothesis stated below.

The software created is robust in order to duplicate results and is flexible as to allow for further studies to be carried out and to use the same method.

Novel additions have been made to existing image analysis libraries in order to make them more flexible for this project. Figure:1.3 illustrates the range of diversity

Furthermore, the library written allows for easy data organisation and automation of otherwise difficult tasks such as concatenating data from multiple sources and graphing of information. Full documentation and integrated testing allows for a suite of tools which can be built upon in future and reduce the amount of effort required for similar studies to be carried out and analysed.

Hypothesis

To provide a full spectrum of analysis the null-hypothesis of this work is presented as investigating if there are morphometric differences in the seeds of several wheat varieties outlined in figure:1.2.

The comparison pairs are as follows (where W indicates wild genotypes and D domesticated.):

1. *T. monococcum* (2N|D) and *T. beoticum* (2N|W)
2. *T.dicoccum* (4N|D) and *T. dicoccoides* (4N|W)
3. *T. spelta* (6N|D) and *T. aestivum* (6N|D)
4. *T. dicoccum* (4N|D) and *T. durum* (4N|D)
5. *T. beoticum* (2N|W) and *T. dicoccoides* (2N|W)

These comparison groups where chosen, with help from researchers at the National Plant Phenomics Centre, based on their ploidy and also on their domestication grouping. This maximises the potential of this research by isolating features and attributes of wheat based on domestication status.

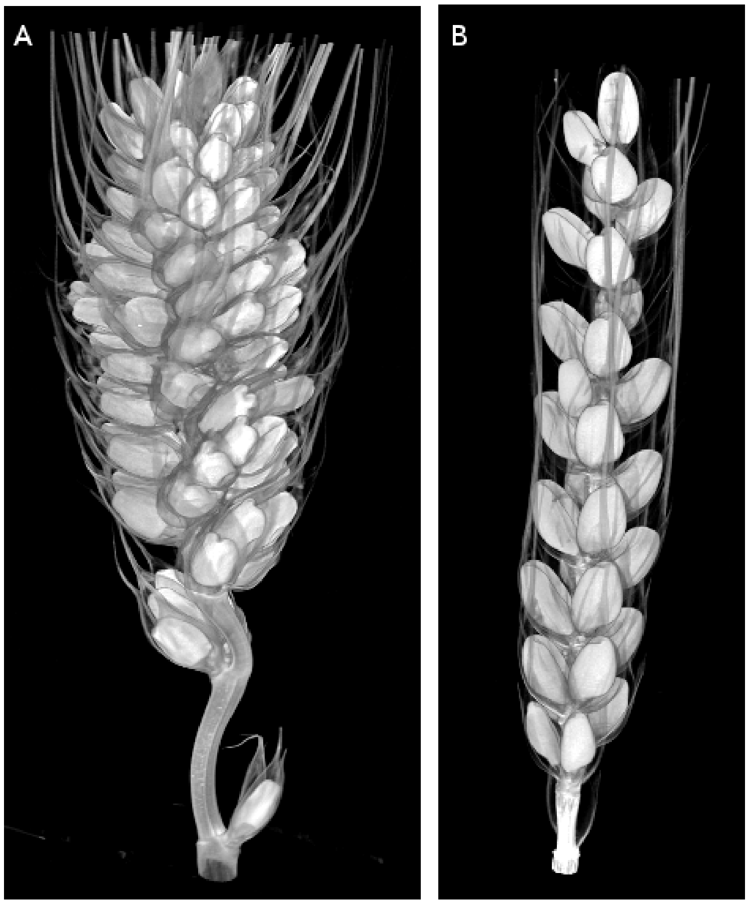


Figure 1.3: Scans of wheat, showing diversity in Population, Compactum (6N) left, Durum right (4N)

Challenges Overview

The challenges which this project tackles come in two flavours: Computational and Biological. As such keen awareness of these is needed to appreciate the novelty of this work.

Biological Challenges

Previous studies have been able to demonstrate that variation in wheat grain morphology can be partially explained, in 2010 Gegas et al. demonstrated this through a 99.4% 2 component PCA [21]. However there is much left to do in terms of formal classifications and descriptions of these differences. This project deals with this problem through computational analysis.

Two effects run parallel in this study, both of these need to be accounted for in the analysis, and questions need asked in a manner to extract each effect independently:

1. The effects of ploidy in wheat.
2. The effects of domestication in wheat.

Hypothesis are required to take into account, both of these effects so as not to misidentify results.

Computational Challenges

Using µ-CT data in plant sciences is becoming more and more common [1, 15, 17, 22] and whilst a lot of studies focus on the traits of grains specifically no formal model has been created, no accepted data format. This is a data engineering problem and the methods described in this project address this.

Further to data organisation, proposals are made for the statistical analysis which should be used. This allows for studies to become more robust and repeatable, thus strengthening the studies overall.

The biological material used in this research is much more diverse a population than has been previously studied with µ-CT image analysis, this requires current computer vision methods to be adapted in order to be accurate.

Deliverables

This project provides three final deliveribles:

1. A flexible software suite written in *Python* that provides a standardised method for analysing and interpreting µ-CT data output.
2. A Graphical User Interface (GUI) which offers a point and click method for data gathering, graphing and manipulating µ-CT data, using the library from deliverable 1 as a backend.
3. Answers to the proposed questions (hypothesis), the *Results* and *Discussion* sections of this report provides this.

Chapter 2

Software Design, Implementation and Testing

This chapter outlines choices and methodologies employed in the software engineering aspect of this project, as well as highlighting the key functional requirements and implementation decisions.

Functional Requirements

Requirements for this project are split between software requirements for both the CT Analysing Library and the CT GUI Application and the research requirements (i.e. the answers to the proposed hypothesis). Here the requirements for the software are discussed:

Requirements for CT Analysing Library

These are the functional requirements for the Python library produced:

0. Provide an OOP means to deal with data
1. Make gathering of data simplified
2. Handle Saving of data in a usable format
3. Easily enable data transformations
4. Perform hypothesis testing
5. Process rejoining of split scans
6. Handle Removing of erroneous data
7. Enable matching data to external information
8. Auto plot data (boxplots, histograms etc.)
9. Allow easy filtering of data

Requirements for CT GUI Application

10. Provide a intuitive user interface for working with CT data
11. Allow a interaction with data without the need for programming
12. Implement the Matplotlib plotting utility
13. Easily join experiment data with CT data
14. Use an MVC model
15. Implement the CT Analysis Library
16. Display data visually
17. Dynamically create graphs
18. Provide hypothesis testing

Software Development Methodology

This project made use of formal design methods and strict organisation whilst being flexible to change. Overall the design took a hybridised form in order to best suit the scientific environment which this domain specific software is built for.

Data analysis drove the direction of the project, as a result an agile methodology was adopted. Weekly sprints were implemented as a list of "todo's", these were written on a Monday morning based off of the previous week's list.

Critical self-evaluation was performed by means of a "one-man SCRUM" meeting, this is a technique which requires self-discipline in order to accurately find faults and areas for improvement [23].

Further to this, regular meetings with research staff, at the National Plant Phenomics Centre, allowed for a developer-client relationship which SCRUM defines as being key. During these meetings details of the research was discussed and ideas given as to how future experiments could proceed. This allowed for critical decisions to be made as to software design and overall structure.

Sprint Timeline

The implementation of this work was done following agile sprint planning, treating each week as a encapsulated working frame, for each of these a detailed organisational programme was created, discussed with supervisors and then used to formulate plans of action for following up on.

A full account of the sprints, which are in part running documentation for this project, are available in appendix:E.

Language Choices

Both the CT Analysing Library and the CT Analysing GUI are implemented using the Python programming language, it has been developed and tested in versions 3.5 and 3.6 (Python 2 is not supported at all by this project).

In scientific programming three of the most commonly used languages are Python, R and MATLAB [24].

These three languages are able to provide all the features which this project requires. However Python was chosen for several reasons.

MATLAB could not be used as a potential language due to it being pay to use software, as this project aims to be accessible, the cost of software would greatly reduce the scope of access.

R is a valid candidate, it provides all of the statistical capabilities required by the project, it also provides packages for creating GUI based applications, it is fast and it is widely used in scientific computing and data science.

The main deciding factor is Python's wealth of resources, adoption rate and the developer of this project being vastly more experienced with Python's ecosystem than R's.

Designing Process

Through meetings and emails, the agile principles of communication over comprehensive documentation was used. Where conversations were decidedly much more beneficial than complex planing prior to developing a product.

Graphical elements, such as the graphing functionality of the CT Analysing Library and the CT GUI Application were sketched using wire-frames whilst in meetings where the potential users (clients) could provide their ideas.

In figure:2.1 an example of the wire-frames created during meetings is show (A), next to it is displayed the final look of the loading window (B).

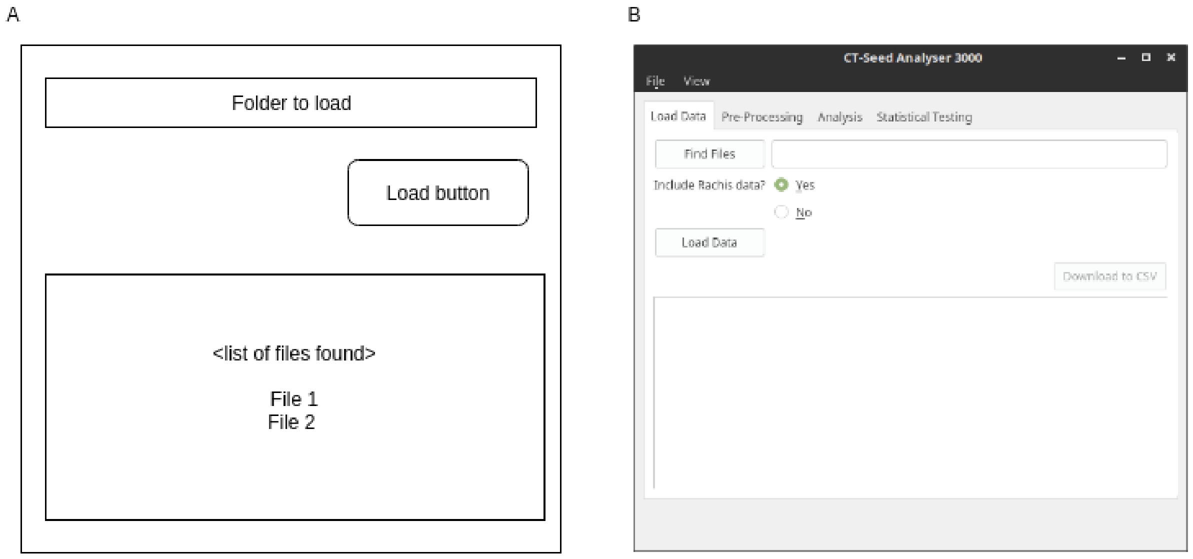


Figure 2.1: Wire-frame of the GUI loading data window

Similarly, figure:2.2 provides the initial wire-frame (A) of how the analysis window could have looked and what kind of GUI elements would be required, again, next to it is the final analysis window (B)

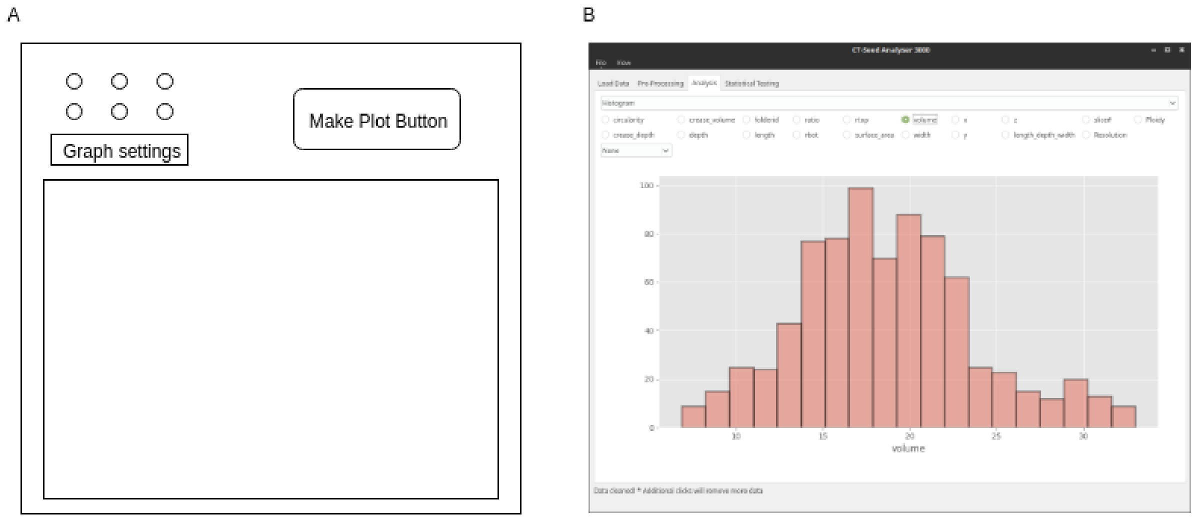


Figure 2.2: Wire-frame of the GUI analysis window

Documentation

Whilst an agile approach was used, some documentation was created for use with the CT Analysing Library.

The provided CT Analysing Library comes with "human-readable" format. Where most documentation generators (Doxygen, Pydocs, Javadocs etc.) implement very well structured and comprehensive documentation, the output is generally not very friendly and easy to read. Particularly for non-career-programmers. A core feature of these provided software implementations are that they are well suited for a biologist, researcher or statistician to use.

This documentation generator was purpose created, implemented in LISP and provided in listing:19.

Beyond this, inline commenting is provided for supplied software. Keeping in line with the agile development ethos the software is self-documented and self-evident. A brief example of this is shown in listing:18

Documentation for the CT GUI application is provided as a visual user guide, and provides sample data for the user to test with.

Software Library Choices

The software libraries used for this project focus around data manipulation, where possible core libraries of the Python language were used and only well supported, established and documented libraries were chosen. Software support is a major requirement for reproducible results.

All software packages used in the Analysing Library are required by the CT Analysing GUI as the Library is a dependency of it. The GUI has a single separate requirement *PyQT5*. Table:A.1 contains a full listing of all software used and required by this project.

Numpy

The Numpy library is one of the most commonly used additions to the Python ecosystem, it is fundamental to many data science projects. Here it is used to handle data lists, arrays and structures. There is no viable alternative to this package and it is required by Scipy and Statsmodels.

Matplotlib/Seaborn

Matplotlib acts as the plotting backend for the project. The Seaborn package acts as a porcelain for matplotlib and makes graph creation and decoration much easier.

Scipy

Data transforms such as Box Cox and PCA are dependant on the functions of the Scipy library. Alternatives are available, however this is the most well established and often used library for these functions.

Pandas

Pandas is used to read the CSV files which the raw data is stored in. This library converts and stores data in dataframes which are used throughout this project to manipulate data.

Xlrd

This extension library is required in order to read Microsoft encoded files. Extra experiment information can be provided with the "xlsx" extension.

Statsmodels

The backbone of the data analysis model is handled by this library, it also provides an implementation of the Markov chain Monte Carlo system used to generate random samples from the model population.

PyQT5

There were many options for creating a user interface in Python, the language provides its own core library via the *TKinter* module. However PyQT is a port of the QT framework, one of the most widely used libraries for GUIs in software development. It is cross platform, robust and has excellent documentation and user-guides.

Version control

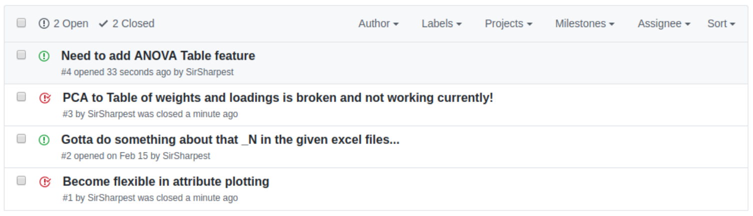
This project has used Git version 2.7.4 throughout. The structure of the project has been as submodules of a larger project.

By using submodules the CT Grain Analysing Library could be kept in sync with the GUI aspect of the project.

Additionally, *setup.py* has been used to provide installation of the library, the code for this can be seen in listing:17. Using *setup.py* provides a quick and easy way for any user to install the software, along with any dependencies.

Issue Tracking

Issues were tracked during the project, both in personal notes and in the Git interface as illustrated in figure:2.3



Need to add ANOVA Table feature
#4 opened 33 seconds ago by SirSharpest

PCA to Table of weights and loadings is broken and not working currently!
#3 by SirSharpest was closed a minute ago

Gotta do something about that _N in the given excel files...
#2 opened on Feb 15 by SirSharpest

Become flexible in attribute plotting
#1 by SirSharpest was closed a minute ago

Figure 2.3: Github Issue Tracking

Implementation Methodology

Strict software engineering principles were applied during creation of this project. The use of standards, design patterns and code-linters have been used throughout to minimise the possibility of errors and to create wholly extendable software. These devices enable understandable and self-documented code allowing future users to quickly start using the provided packages.

Standards

The main standard adhered to for software provided by this project is the PEP8 style guide [25]. The principle behind this coding style, as stated by Guido van Rossum, is "Code is read much more often than it is written". This makes this styling guide perfect for the chosen agile methodology of self-evident documentation in the software.

In addition to PEP8, a Python code linter Flake8 has been used to prevent "code smells", bad formatting, incorrect white space usage etc.

CT Analysing Library Design Pattern

The CT Analysing Library uses a Singleton style design pattern. A single data object is created from a *CTData* class.

A very functional paradigm is used by this library. By applying mapping and filter style functions data elements can be passed to the supporting modules: *data_ transforms.py*; *graphing.py*, *statistical_ tests.py*. These modules enable scientific functions to be applied to the *CTData* object. A UML style class diagram is shown in figure:2.4, here the interactions of the classes can be seen, as well as their internal functions.

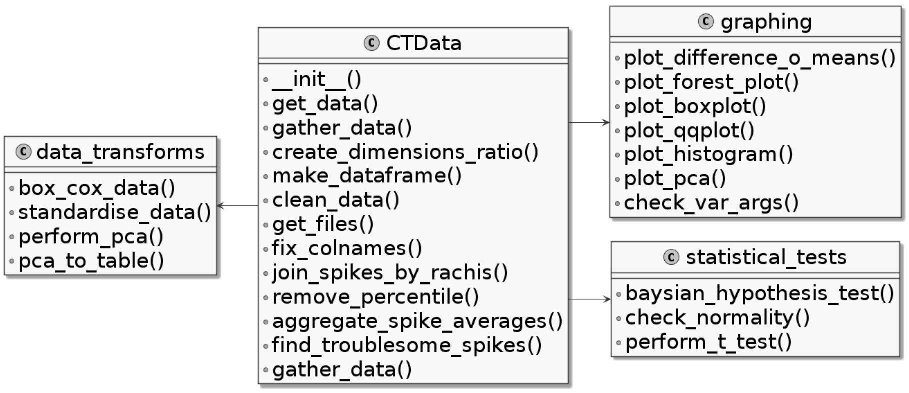


Figure 2.4: CT Analysing Library UML

CT GUI Application Design Pattern

The Model-View-Controller (MVC) design pattern is one of the most commonly structures for creating user interfaces. It allows for the user's view/interface code to be separated from the model, the code which changes the data. The model and the view communicate and update each other via the controller element of the design.

The QT framework provides "connectors" which act as triggers/activations for functions, these are set off by the user providing either keyboard or mouse based input.

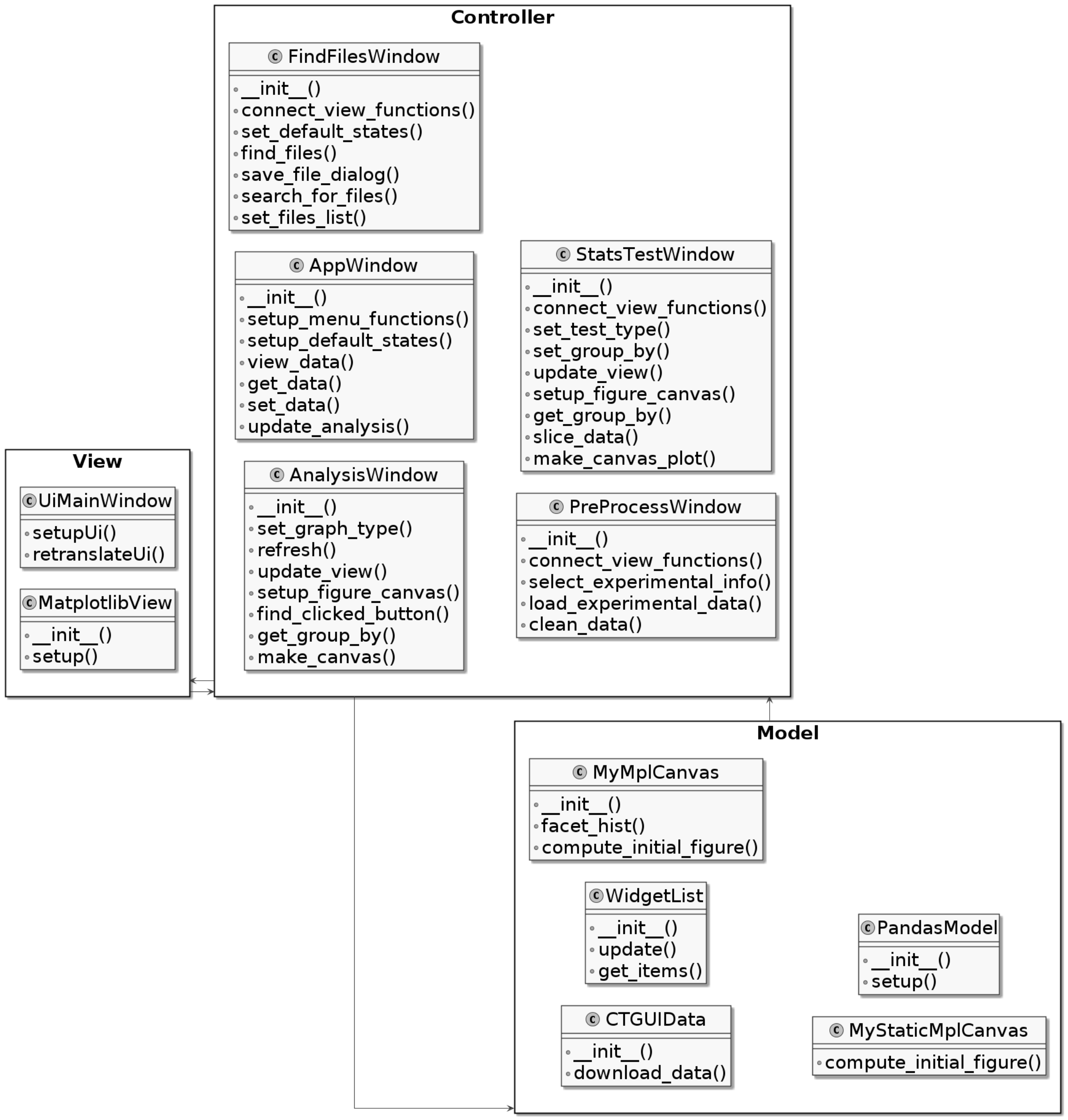


Figure 2.5: CT Analysing GUI UML

Library Versioning

The development of this project was performed using the Python *virtualenv*. This is a virtual environment package which Python offers, it allows for an isolated working copy of the project.

By developing in this manner, libraries were ensured to be using the correct versions required by the software.

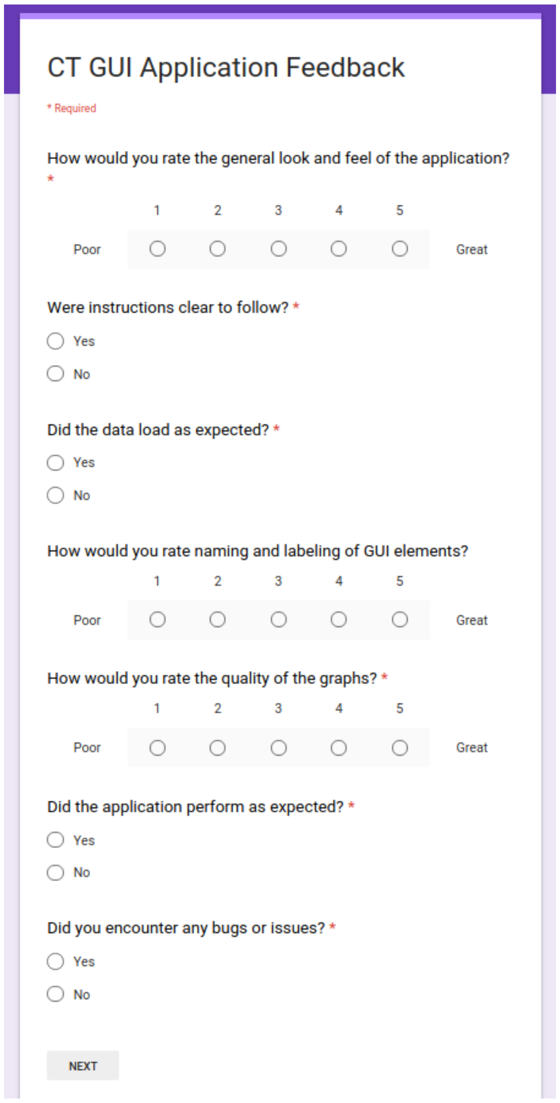
Testing

Testing was performed both in acceptance testing by using user feedback, the functional requirements and the ability to use the software to answer the hypothesis of the research elements of this project. Further to this, unit testing was performed to allow for automated testing as well as test-driven-development of features.

Feedback Forms

Feedback and constructive suggestions were made by researchers at the National Plant Phenomics Centre, these were submitted via the Google forms service.

These provided a method of acceptance testing by those who would be using the software to help with investigating data. Page 1 of the given form is shown in figure:2.6. This form was completed by 3 researchers at the National Plant Phenomics Centre, feedback was very positive overall.



CT GUI Application Feedback

* Required

How would you rate the general look and feel of the application? *

	1	2	3	4	5	
Poor	○	○	○	○	○	Great

Were instructions clear to follow? *

○ Yes
○ No

Did the data load as expected? *

○ Yes
○ No

How would you rate naming and labeling of GUI elements?

	1	2	3	4	5	
Poor	○	○	○	○	○	Great

How would you rate the quality of the graphs? *

	1	2	3	4	5	
Poor	○	○	○	○	○	Great

Did the application perform as expected? *

○ Yes
○ No

Did you encounter any bugs or issues? *

○ Yes
○ No

Figure 2.6: CT Feedback form

Unit Testing CT Analysing Library

The unit tests for the CT Analysing Library were straightforward, using the *PyTest* framework and a subset of data from a data set, these tests assert that features are implemented correctly and that the correct results are given.

Table 2.1: Output of *pytest* Unit Tests and results for CT Analysing Library

I.D.	**Result**	**Test**
0	Passed	CTData.py::test_aggregate_spike_averages
1	Passed	CTData.py::test_clean_data_maximum_removed
2	Passed	CTData.py::test_clean_data_minimum_removed
3	Passed	CTData.py::test_load_additional_data
4	Passed	CTData.py::test_load_additional_data_no_data
5	Passed	CTData.py::test_load_data
6	Passed	CTData.py::test_NoDataFoundException
7	Passed	Data_transforms.py::test_box_cox_data
8	Passed	Data_transforms.py::test_pca_to_table
9	Passed	Data_transforms.py::test_perform_pca
10	Passed	Data_transforms.py::test_standardise_data
11	Passed	Graphing.py::test_plot_boxplot_as_dataframe
12	Passed	Graphing.py::test_plot_boxplot_as_object
13	Passed	Graphing.py::test_plot_difference_of_means
14	Passed	Graphing.py::test_plot_histogram_as_dataframe
15	Passed	Graphing.py::test_plot_histogram_as_object
16	Passed	Graphing.py::test_plot_pca
17	Passed	Graphing.py::test_plot_qqplot
18	Passed	Statistical_tests.py::test_baysian_hypothesis_test
19	Passed	Statistical_tests.py::test_t_test
20	Passed	Statistical_tests.py::test_test_normality

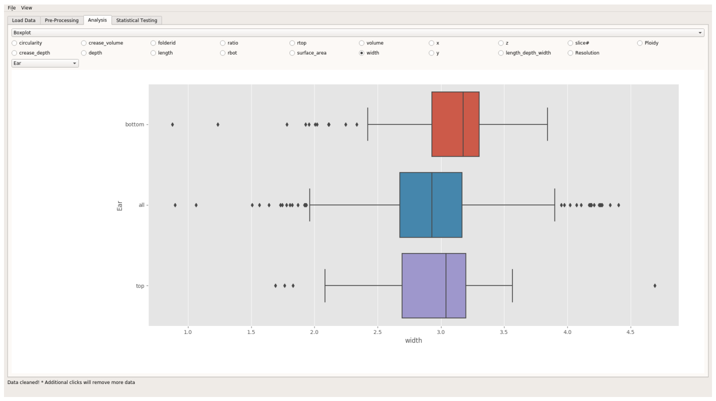
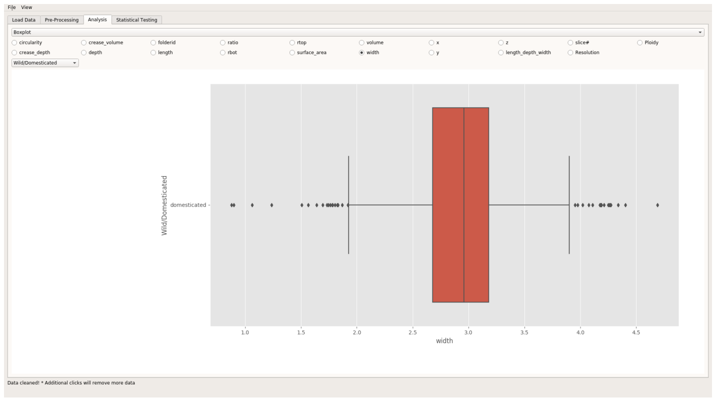
Unit Testing CT GUI Application

The unit testing used for the CT GUI Application was more sophisticated than that of the Library. This testing required visual confirmation that figures and graphs generated were displayed correctly and that they showed what the user would expect, given the data.

To do this a *PyTest* plugin was used, *QtBot* which provides simulated user input. This allows for the GUI to be thoroughly tested, automatically.

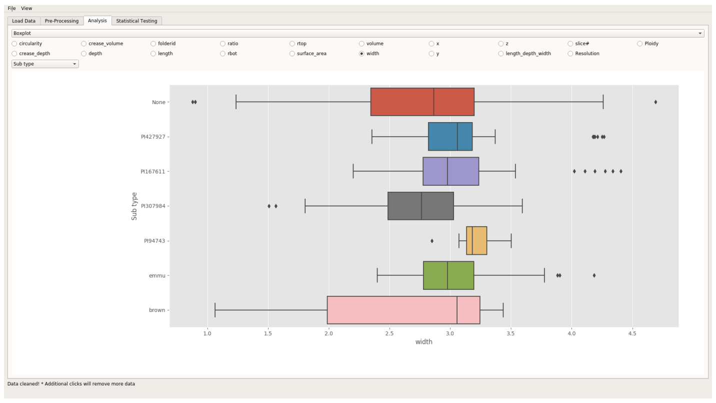
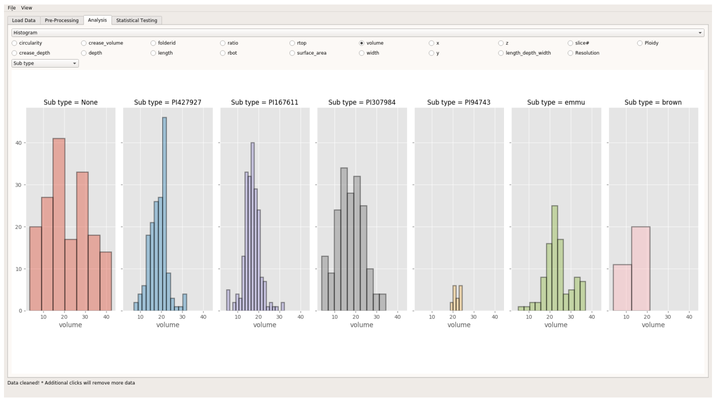
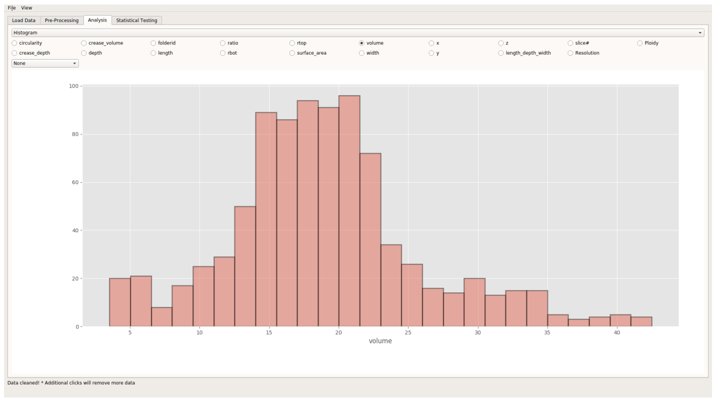
In table:2.2 the results of the automated testing is given along side an image of several of the tests, tests of the same graphs but with different parameters were also generated and manually verified and provided as supplemental data.

Table 2.2: Output of *pytest* Unit Tests and results for CT GUI Application

I.D.	**Result**	**Test**	**Image**
21	Passed	analysis.py:: box_groupby_1_rb_1	 File View; Load Data; Pre-Processing; Analysis; Statistical Testing; Boxplot; circularity; crease_volume; folderid; ratio; rtop; volume; x; z; slice#; Ploidy; crease_depth; depth; length; rbot; surface_area; width; y; length_depth_width; Resolution; Ear; bottom; all; top; 1.0; 1.5; 2.0; 2.5; 3.0; 3.5; 4.0; 4.5; width Boxplot, width selected, grouped by Ear. Data cleaned! * Additional clicks will remove more data
22	Passed	analysis.py:: box_groupby_2_rb_2	 File View; Load Data; Pre-Processing; Analysis; Statistical Testing; Boxplot; circularity; crease_volume; folderid; ratio; rtop; volume; x; z; slice#; Ploidy; crease_depth; depth; length; rbot; surface_area; width; y; length_depth_width; Resolution; Wild/Domesticated; domesticated; 1.0; 1.5; 2.0; 2.5; 3.0; 3.5; 4.0; 4.5; width Boxplot, width selected, grouped by Wild/Domesticated. Data cleaned! * Additional clicks will remove more data

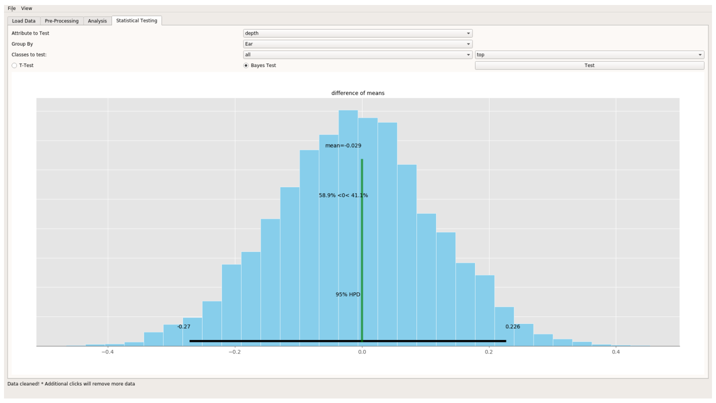
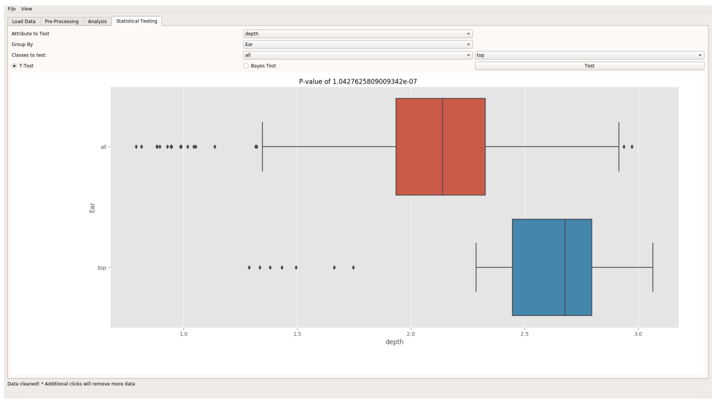
Continued on next page

Continued from previous page

I.D.	**Result**	**Test**	**Image**
23	Passed	analysis.py:: box_rb_1	 Data cleaned! * Additional clicks will remove more data
24	Passed	analysis.py:: hist_groupby_1_rb_1	 Data cleaned! * Additional clicks will remove more data
25	Passed	analysis.py:: hist_rb_1	 Data cleaned! * Additional clicks will remove more data

Continued on next page

Continued from previous page

I.D.	**Result**	**Test**	**Image**
26	Passed	hypothesis_tests.py:: bayesg1_att_1	 Attribute to Test depth Group By Ear Classes to test: all top ○ T-Test ● Bayes Test Test Data cleaned! * Additional clicks will remove more data
27	Passed	hypothesis_tests.py:: tg1_att_1	 Attribute to Test depth Group By Ear Classes to test: all top ● T-Test ○ Bayes Test Test P-value of 1.0427625809009342e-07 Data cleaned! * Additional clicks will remove more data
28	Passed	analysis.py:: box_rb_2	N/A
29	Passed	analysis.py:: hist_groupby_1_rb_2	N/A
30	Passed	analysis.py:: hist_rb_2	N/A
31	Passed	analysis.py:: loads	N/A
32	Passed	hypothesis_tests.py:: bayesg1_att_2	N/A
33	Passed	hypothesis_tests.py:: bayesg2_att_1	N/A
34	Passed	hypothesis_tests.py:: bayesg2_att_2	N/A
35	Passed	hypothesis_tests.py:: tg1_att_2	N/A
36	Passed	hypothesis_tests.py:: tg2_att_1	N/A
37	Passed	hypothesis_tests.py:: tg2_att_2	N/A
38	Passed	hypothesis_tests.py:: loads	N/A
39	Passed	GUI.py:: startup	N/A

Continued on next page

Continued from previous page

I.D.	Result	Test	Image
40	Passed	load_data.py:: load_data_with_rachis	N/A
41	Passed	load_data.py:: load_data_without_rachis	N/A
42	Passed	preprocessing.py:: clean_data_remove_large	N/A
43	Passed	preprocessing.py:: clean_data_remove_none	N/A
44	Passed	preprocessing.py:: clean_data_remove_small	N/A
45	Passed	preprocessing.py:: clean_data_remove _small_and_large	N/A
46	Passed	preprocessing.py:: load_additional_data	N/A
47	Passed	preprocessing.py:: load_additional_data _expected_fail	N/A

Acceptance Testing

These unit tests, alongside user tests have been used to meet the outlined functional requirements; specific tests are shown in table:2.3 to match them to the associated functional requirement.

Table 2.3: Functional requirements and Unit tests

F.R.	U.T.	How is the F.R. met?
0	n/a	An OOP interface is provided by the *ct_ data* object
1	3,4,5	Loading of data is tested multiple times specifically through unit tests
2	4,5	Unit tests carry out saving loaded data into appropriate formats, in particular python *pandas* tables and CSV files, this was also user tested
3	7,8,9,10	All the functions in the *data_ transforms.py* library are unit tested automatically on test data of known output
4	26,27,32-38	Hypothesis testing was tested with known data 8+ times in automated unit testing for both CT GUI Application and the CT Analysis Library group of tests , testers also used these functions
5	6,7	Loading of spikes was tested both for data which was known to exist and could be successfully done, and for data which was know to throw errors
6	1,2,7,42-46	Finding and removing erroneous data was unit tested by having a combination of parameters tested for both the GUI and the Analysis Library
7	3-6,40,41	Matching of pre-existing experiment information to the extracted data was tested for bad inputs through unit testing, users tested this too and reported no unexpected results
8	11-17,28-38	Plotting of data automatically, inferring of axis, titles and measurements was tested, with images automatically recorded by unit tests.

Continued on next page

Continued from previous page

F.R.	**U.T.**	**How is the F.R. met?**
9	0,1,2,3	Unit testing checked if given data could easily be separated and divided into subsections, this was specifically tested and tested by proxy in other tests when hypothesis testing was performed
10	21,26 39,40,42	Multiple tests are taken to check that the GUI loads as expected, planning sessions and user feedback was also used to meet this particular functional requirement
11	40,42	Data is loaded through GUI elements with little prior knowledge of the software in order to have data processed, this was unit tested and user feedback was used to make more accessible
12	21-27	Testing of *Matplotlib* and *Seaborn* interfacing with, and integrating with the CT Analysing GUI.
13	43-47	Loading of experiment data is pivotal for most functionality of the GUI, it is tested multiple times in an isolated formatted, as well as by other future functions which require it
14	n/a	The entire software is wrapped in an MVC model, this cannot be tested but the functional requirement has been met
15	21-47	All of the unit tests for the CT Analysing GUI depend on the CT Analysing Library, it is reasonable that the integration is well tested, by product of all other tests
16	21-27	Several tests, those shown in table:2.2 are specifically recorded visually and presented as images in order to automatically unit test that visual data is intact
17	21-27	By using automatic image capturing of unit testing in progress, visual checking can be carried out and confirmed that data is visually what is expected and to a high standard
18	26,27, 32-28	With multiple known outputs, several different groupings and combinations the hypothesis tests were unit tested and this functional requirement is known to be acceptably implemented

Chapter 3

Methods and Solutions

This chapter will discuss and expand upon the software implementations which were carried out in the course of this project, with particular focus on novel and non-trivial elements of software engineering.

Where appropriate code segments are provided to illustrate how a feature has been implemented in software.

Data Processing Methods

This project presents a start to finish style of data analysis, data are generated by extracting information from raw scans, the resulting data is then post-processed and statistical analysis draws information from this.

Historically, these processes and tasks are huge time sinks in terms of manual work, this project emphasises an increase in automation.

Two separate pipelines are provided to carry out these tasks.

Image-Data Requisition Pipeline

Acquiring data for this study first required Wheat plants to be grown and harvested, the process outlined here begins with scanning samples from these plants in a µ-CT machine.

These scans are processed by MATLAB [26]. This process is defined as a method by Hughes et al. [1], new and novel additions are added in the watershedding and segmentation processes.

The process begins by loading the scans into greyscale images, the pixel values (which translate to density values) are used to calculate a threshold that is used to define and segment material of interest (seeds).

Due to the low resolution of the imaging technique (68.6µ meters per pixel) objects can appear connected which are not, a three dimensional watershedding algorithm is used to correct any objects which appear connected when they should not be.

Finally, each isolated object is exported for measuring (objects are defined by groups of connected pixels), measurements are recorded into tables and saved as CSV files. The data is recorded as pixel values and converted to appropriate real-world measurements.

The entire process is represented as a diagram in figure:3.1

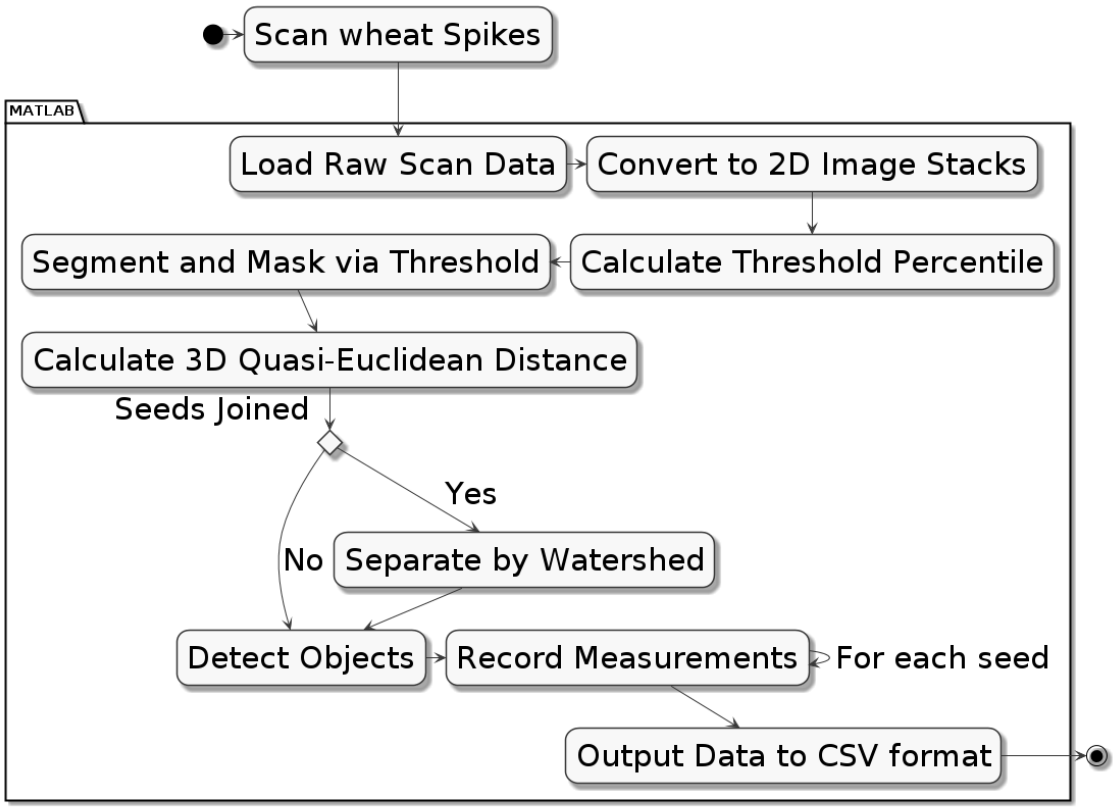

Scan wheat Spikes
MATLAB
Load Raw Scan Data
Convert to 2D Image Stacks
Calculate Threshold Percentile
Segment and Mask via Threshold
Calculate 3D Quasi-Euclidean Distance
Seeds Joined
Yes
No
Separate by Watershed
Detect Objects
Record Measurements
For each seed
Output Data to CSV format

Figure 3.1: Image Processing Pipeline

Data Analysis Pipeline

The data analysis and research in this project is performed in part by automated software, the CT Analysis Library, and by human decision making as to statistical methods which need to be applied.

Here, the pipeline presented in figure:3.2 demonstrates the process used by the CT Analysis Library, and by result the CT Analysing GUI.

The first stage of this process is loading in the data exported by the MATLAB software. These data are loaded on a per scan basis, meaning that the previously mentioned process of spike splitting is initially still an issue, a resulting *CT Data* object is created, this holds the data and carries out the proper treatment for it.

The data needs to be checked for false positives, this is done by first removing outliers which are found by upper and lower percentiles of the data. Additionally constraints are applied to the data based on findings from previous studies [1], this adds robustness.

When the data has been cleaned, additional data is added. The added data provides information about the experiment, plant names; genotypes; top and bottom scans. With this data, information on scans which need to be joined is provided.

Using data on x, y, z coordinates a approximation is made to join spikes at the most likely point at which they were initially cut for imaging. Once this is performed data columns can be aggregated to form averages, medians, standard deviations etc. on a per plant (sample) basis.

Finally, a decision is made by way of human input as to what type of statistics should be used to test hypothesis. This decision is mostly based on the distribution of the data where $X \sim \mathcal{N}(\mu, \sigma^2)$ data is tested with parametric tests (e.g. Student's T-Test) else non-parametric tests are used (e.g. Welch's T-Test).

From statistical testing a final result(s) can be used to answer scientific questions on the data. The entire process is represented in figure:3.2.

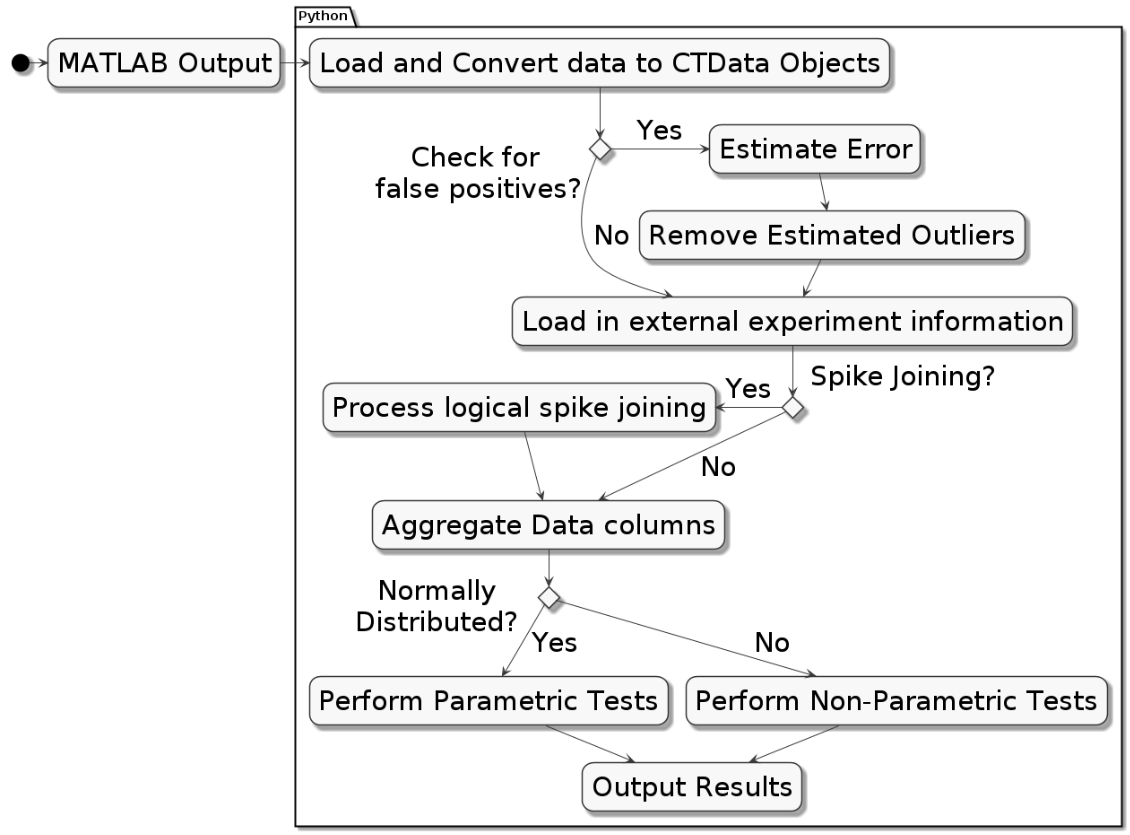

Python
MATLAB Output
Load and Convert data to CTData Objects
Check for false positives?
Yes
Estimate Error
Remove Estimated Outliers
No
Load in external experiment information
Spike Joining?
Yes
Process logical spike joining
No
Aggregate Data columns
Normally Distributed?
Yes
No
Perform Parametric Tests
Perform Non-Parametric Tests
Output Results

Figure 3.2: How data is integrated with the CT Analysing Library

Image Analysis Methods

Initially, this project aimed to use existing software for feature extraction, however the range of data (due to the spectrum of genotypes in the study) meant that software which worked well for certain genotypes of wheat did not work well on others.

The issue became in seeds becoming erroneously joined, research and investigation lead to a new solution being implemented in a forked version of the open source software used for image analysis [1]

New Watershed Algorithm

In order to solve the problem of misidentified and joint seeds, from the primitive collection, a *quasi-euclidean* distance transform was implemented into the analysis pipeline (figure:3.1). This provided much better results than the previous *chessboard* transform which had been successful on more uniform data in previous studies [1].

Quasi-Euclidean algorithm

This algorithm measures the total euclidean distance along a set of horizontal, vertical and diagonal line segments [27].

$$\begin{array}{l} |x_1 - x_2| + (\sqrt{2}-1)|y_1 - y_2|, |x_1 - x_2| > |y_1 - y_2| \\ (\sqrt{2}-1)|x_1 - x_2| + |y_1 - y_2|, \text{otherwise.} \end{array} \tag{3.1}$$

In order to apply this to a 3D space Kleinberg's method is used [28]. This allows for nearest neighbour pixels to be sorted by k dimensional trees and enabling fast distance transforms via Rosenfeld and Pfaltz's *quasi-euclidean* method stated in equation:3.1.

Implementation

The software implementation for this was done in MATLAB [26], this meant that it could instantly and easily be inserted into the existing pipeline in the open source software which was forked and modified to add this feature. In code listing:1 this new algorithm, in full, is shown.

```
function [W] = watershedSplit3D(A)
  % Takes image stack A and splits it into stack W.
  % This is done by using a watershedding
  % algorithm based on quasi-euclidean distance

  % Convert to BW
  bw = logical(A);
  % Create variable for opening and closing
  se = strel('disk', 5);
  % Minimise object missshapen-ness
  bw = imerode(bw, se);
  bw = imdilate(bw, se);
  % Fill in any left over holes
  bw = imfill(bw,4,'holes');
  % Use chessboard for distance calculation for more refined splitting
  chessboard = -bwdist(~bw, 'quasi-euclidean');
  % Modify the intensity of our bwdist to produce chessboard2
  mask = imextendedmin(chessboard, 2);
  chessboard2 = imimposemin(chessboard, mask);
  % Calculate watershed based on the modified chessboard
  Ld2 = watershed(chessboard2);
  % Take original image and add on the lines calculated for splitting
  W = A;
  W(Ld2 == 0) = 0;
end
```

Listing 1: MATLAB Watershedding function

The process creates a black and white copy of the image, performs erosion and dilation to remove holes and erroneous pixels. A distance map is then created, this represents each pixel, which is plant material (grains), as an integer value of distance from the closest non-plant pixel. This map combines with the watershed algorithm to provide an estimate of where objects are incorrectly connected (fused grains), subtracting this from the original 3D image provides cleaning split and segmented grains which can be passed through to the next stage of the pipeline.

Reasons for Improvement

This improved algorithm for measuring distance between points works well with this data because of the variation in seed morphology.

Where previous studies [1] found their methods worked well, they also used elite well-bred wheat varieties. These varieties have a more round more even shape with a low standard deviation in shape, generally.

This study, with such a varied group of genotypes has encountered many seeds which do not have a reasonable degree of regularity in shape. As such, a simple distance based method of splitting from the centre of objects was ill suited. By using a quasi-euclidean distance a more flexible shape is afforded in grains and the resulting segmentation is less prone to over-zealous splitting due to non-uniform shape.

Effect of Enhanced Watershed algorithm in 2D

Figure:3.3 shows in two dimensional flat images how this problem was solved. As well as what the previous method was doing to cause errors in data.

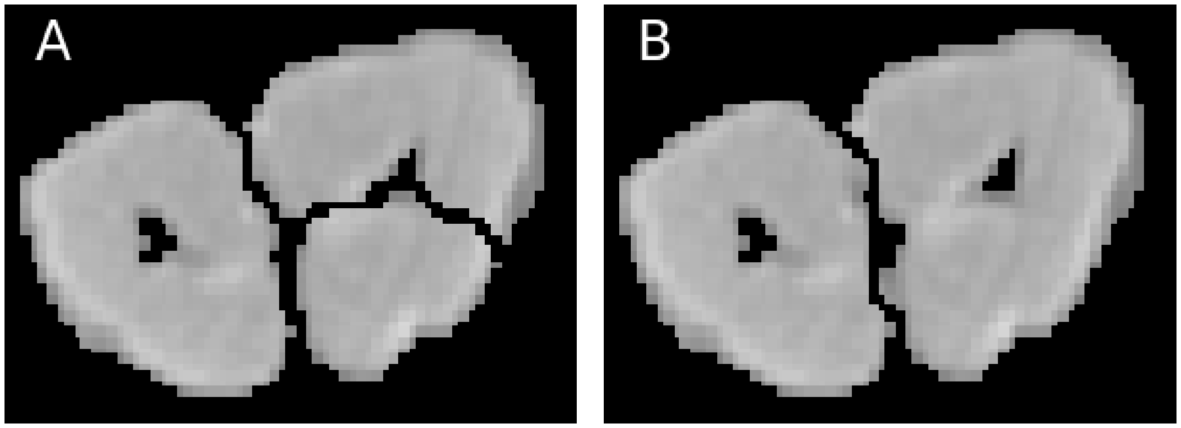


Figure 3.3: *A* showing the chessboard method, *B* improved quasi-euclidean method

Effect of Enhanced Watershed algorithm in 3D

Figure:3.4 shows the effects on grains in three dimensions, where the seeds which where incorrectly split have now been vastly improved, leading to an increase in result accuracy.

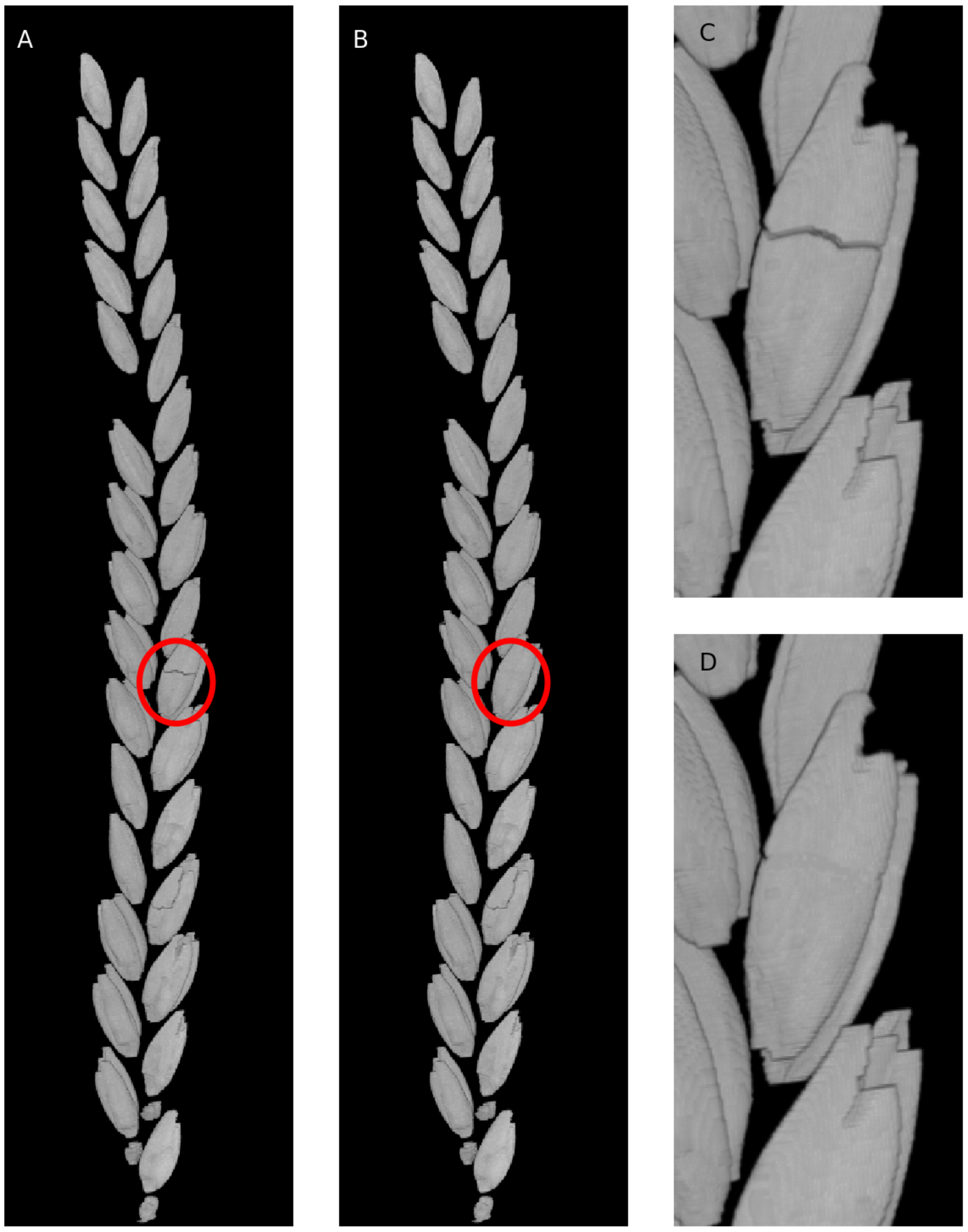


Figure 3.4: *A* showing before the new method, *B* after, *C* zoomed before, *D* zoomed after.

CT Analysing Library Methods

The CT Analysing Library provided by this project, aims to reduce the amount of work and time required to organise, analyse and pull meaning out of µ-CT data. Here several of the more novel features of the library are explained.

CT Data Object

The CT Analysing Library uses a mixture of object orientated (OOP) and functional programming paradigms, the main initialisation function for the object *CTData* enables the storing of data and application of functions upon it, making data organisation much more straightforward.

```python
class CTData():
    def __init__(self, folder, rachis):
        """
        This is the initialise function for the CTData object,
        this will only need called once.

        @param folder the folder to look for data in
        @param rachis a boolean to decide to load in rachis data or not
        """
        try:
            self.make_dataframe(folder, get_rachis=rachis)
            self.clean_data()
            self.create_dimensions_ratio()
            self.df = self.df.reset_index(drop=True)
        except (ValueError, NoDataFoundException):
            raise NoDataFoundException
        self.additional_data = None
```

Listing 2: CT Data Object Initialisation Method

Loading Data

Initial data loading is performed by the *gather_data* function, it takes a folder string and performs a regular expression check for all CSV files. From the files found in this search this function performs separation of rachis information (data about the spike's inner nodes) and the files containing seed/grain data.

These are returned together as a tuple, the data found is stored as part of the *CTData* object for use in the processing pipeline.

```python
def gather_data(self, folder):
    """
    this function gathers together all
    the data of interest

    @param folder is a starting folder
    @returns tuple of (seed files, rachis files)
    """

    # check the end of the folder has a '/'
    if folder[-1] != '/':
        folder = folder + '/'

    search_params = '{0}*/*.csv'
    candidate_files = glob(search_params.format(folder))

    # Check data was found
    if len(candidate_files) == 0:
        raise NoDataFoundException

    # we aren't bothered about the raw files so lets remove them
    candidate_files = [f for f in candidate_files if 'raw' not in f]

    # now let's separate out the rachis
    rachis = [f for f in candidate_files if 'rachis' in f]

    # and just assume the rest is what we want
    candidate_files = [f for f in candidate_files if 'rachis' not in f]

    return (candidate_files, rachis)
```

Listing 3: Loading Data Function

Adding Spike Experiment Information

Using a loosely defined Microsoft excel (.xlsx) file, a file of additional information can be loaded into the *CTData* object. This new information can be any information which is relevant to the individual scans.

This function *get_spike_info* works by finding the matching column to join data on, then using the provided excel file it adds these new columns of information into the *CTData* object.

By using λ functions (named *look_up* and *gather_data*) rapid allocation of information can be done in sections, applying hundreds of data entries to the existing data. This is one of many elements in this project which solves challenges in a very functional programming manner.

```
def get_spike_info(df, excel_file, join_column='Folder#'):
    """
    This function should do something akin to adding additional
    information to the data frame

    @note there is some confusion in the NPPC about whether to use
    folder name or file name as the unique id when this is made into
    end-user software, a toggle should be added to allow this

    @param excel_file a file to attach and read data from
    @param join_column if the column for joining data is required
    """
    # Grab the linking excel file
    info = pd.read_excel(excel_file,
                         index_col='Folder#')
    # These are the features to grab
    features = list(info.columns)
    # Lambda to look up the feature in excel spreadsheet
    def look_up(x, y): return info.loc[x['folderid']][y]
    # Lambda form a series (data row) and apply it to dataframe
    def gather_data(x): return pd.Series([look_up(x, y) for y in features])
    df[features] = df.apply(gather_data, axis=1)
    # Return the copy
    return df
```

Listing 4: Spike Information Joining Algorithm

Spike Rejoining

The *Scanco* µ-CT100 scanner (*Scanco* Medical, Switzerland) allows for a carousel loading of samples, the maximum allowed length of a sample is ~10cm, in this study the majority of wheat spikes exceed this limit. A solution was found in splitting these samples in half and creating two separated µ-CT scans.

This process means that post imaging these sample data need to be reconnected and the z axis of all scans need adjusted.

One of the main issues of this task is that there are multiple ways that rejoining could be implemented, this library provides a function that standardises the process. Making repeatable results much easier to produce.

The implementation for this searches through the provided data frame for spikes which are not marked as being 'all', that is to say complete and non-split. From these spikes left, they are either marked 'top' or 'bottom', using an associated 'Sample name' these can be linked. Once linked the top of the spikes need to have their z attribute increased by the size of the bottom spike.

```
def join_spikes_by_rachis(self):
    """
    Important part of this function is that we accept that the data is what it is
    that is to say: rtop, rbot and Z are all orientated in the proper direction
    It's main purpose is to join split spikes by rachis nodes identified in the
    analysis process

    @param grain_df is the grain dataframe to take on-board
    """
    # So we are only really interested in grains which are not labelled with
    # 'all' in partition, so let's id them to start with
    for sn in self.df[self.df['Ear'] != 'all']['Sample name'].unique():

        # Calculate estimated bottom location of a spike
        bot = self.df.loc[(self.df['Sample name'] == sn)
                         & (self.df['Ear'] == 'bot')]['rbot']

        # Apply bottom measurement where required by incrementing
        # 'z' values to top scans
        self.df.loc[
            (self.df['Sample name'] == sn) &
            (self.df['Ear'] == 'top'), 'z'] =
                self.df.loc[(self.df['Sample name'] == sn) &
                (self.df['Ear'] == 'top'), 'z'] + bot
```

Listing 5: Spike Rejoining Function

Data Aggregating

The analysis method used is focused on extracting data on a per-grain basis, grains have a many to one relationship of spikes.

In order to look at the data on a spike-per-spike basis rather than from seed-to-seed the data needs to aggregated.

The function *aggregate_spike_averages* takes a list of attributes to be aggregated and a manner of grouping the data. Using this data new columns are created based on median; mean; standard deviation; sum. These new columns are stored in the *CTData* object directly.

```python
def aggregate_spike_averages(self, attributes, groupby):
    """
    This will aggregate features (specified by attributes) into their medians
    on a per-spike basis.

    Makes direct changes to the dataframe (self.df)

    @param attributes list of features to average
    @param groupby how the data should be aggregated
    """

    # Functions which are used as an apply to the dataframe
    trans_funcs = {'median': np.median,
                   'mean': np.mean, 'std': np.std, 'sum': np.sum}

    # For each attribute apply
    for att in attributes:

        # For each function apply to each attribute in turn
        for col, func in trans_funcs.items():

            # Add straight to the dataframe using appropriate formatting to keep
            # naming straightforward
            self.df['{0}_{1}'.format(col, att)] = self.df.groupby(groupby)[
                att].transform(func)

    # Also add some information on the counts of seeds per plant
    self.df['grain_count'] = self.df.groupby(
        groupby)[groupby].transform(len)
```

Listing 6: Aggregating Data Function

Cleaning and Removing Erroneous Data

Data are cleaned by calculating percentiles of minimum data, these extreme values are removed. In addition to this, values found in seed/grain literature are used to remove data based on known extremes [1, 21].

```
def clean_data(self, remove_small=False, remove_large=False):
    """
    Following parameters outlined in the
    CT software documentation I remove outliers
    which are known to be errors

    @param remove_small a boolean to remove small grains or not
    @param remove_large a boolean to remove larger grains or not
    """
    # Remove all data which has any null or none types
    self.df = self.df.dropna(axis=1, how='all')

    # Remove obviously wrong and too large grains
    self.df = self.df[self.df['surface_area'] < 100]
    self.df = self.df[self.df['volume'] > 3.50]  # this is given for brachy
    self.df = self.df[self.df['volume'] < 60]

    # If the large parameter is set then remove the upper 5th percentile of volume
    if remove_large:
        self.df = self.df[self.df['volume'] <
                          self.df['volume'].quantile(.95)]
    # If the large parameter is set then remove the lower 5th percentile of volume
    if remove_small:
        self.df = self.df[self.df['volume'] >
                          self.df['volume'].quantile(.05)]
```

Listing 7: Cleaning Data

Data Transformations (PCA)

In this project several data transformation algorithms are implemented uses the *Scipy* library [29].

For principal component analysis (PCA), the CT Analysis Library implements it using two principal components (PCs) only. The decision to limit this library to using only two PCs was made to keep data as easy to interpret as possible. Additionally experimentation with this transform shows a high percentage of coverage and data explanation from only two PCs.

Prior to using the PC function, data are standardised, this is either done through log transforms or by Box Cox power transforms. This provides more normal data and allows for information to be more likely to provide a significant output.

```python
def perform_pca(df, features, groupby, groupby2=None,
                groupby3=None, standardise=False):
    """
    This function will perform a PCA and return the principle components as a
    dataframe.

    @param n_components components to check form
    @param df dataframe of the data to analyse
    @param features features from the dataframe to use
    @param groupby the column in the df to use
    @param standardise=False asks whether to standardise the data prior to PCA
    @returns a dataframe of the data, the pca object & the scaled data for reference
    """
    # Only use 2 pcs
    pca = PCA(components=2)
    # Standardise the data if requested
    data = standarise_data(
        df, features, groupby) if standardise else df.loc[:, features].values
    # Perform pca fitting on the chosen data
    principalComponents = pca.fit_transform(data)
    # form the pca into a dataframe with pca1 and 2 labelled
    principalDf = pd.DataFrame(data=principalComponents, columns=[
                               'principal component 1', 'principal component 2'])
    # Decide how to group the data into the final pca table to return
    # three options are given as multiple analysis may be required
    if groupby2 is None:
        return (pd.concat([principalDf, df[[groupby]]],
                          axis=1), pca, data)
    if groupby3 is None:
        return (pd.concat([principalDf, df[[groupby]],
                           df[[groupby2]]], axis=1), pca, data)
    else:
        return (pd.concat([principalDf, df[[groupby]],
                           df[[groupby2]], df[[groupby3]]], axis=1), pca, data)
```

Listing 8: PCA Data Transform

Bayesian Hypothesis Testing

Null-hypothesis significance testing (NHST) is done typically through T-test like functions, where the means of two groups are tested. As a novel addition, this library provides an improvement upon NHST by using a Bayesian approach proposed by Kruschke in 2012 [30].

Here a full model is produced where data are first normalised through log transforms, model parameters set and then 1000 simulations ran via Markov chain Monte Carlo to produce a larger data sample to perform mean testing on.

The 1000 simulations are ran twice, these form two independent chains, checking of convergence and agreement on these random chains is performed to reduce the chance of error.

This function returns a *trace* object which contains all of the data generated, the data of particular interest is the "difference of means" that this method calculates and the credible interval graphs which it returns the values for, figure:3.5 provides an example of such. The example provided shows the likelihood of two means of an interaction term of *length* $\times$ *depth* $\times$ *width* overlapping.

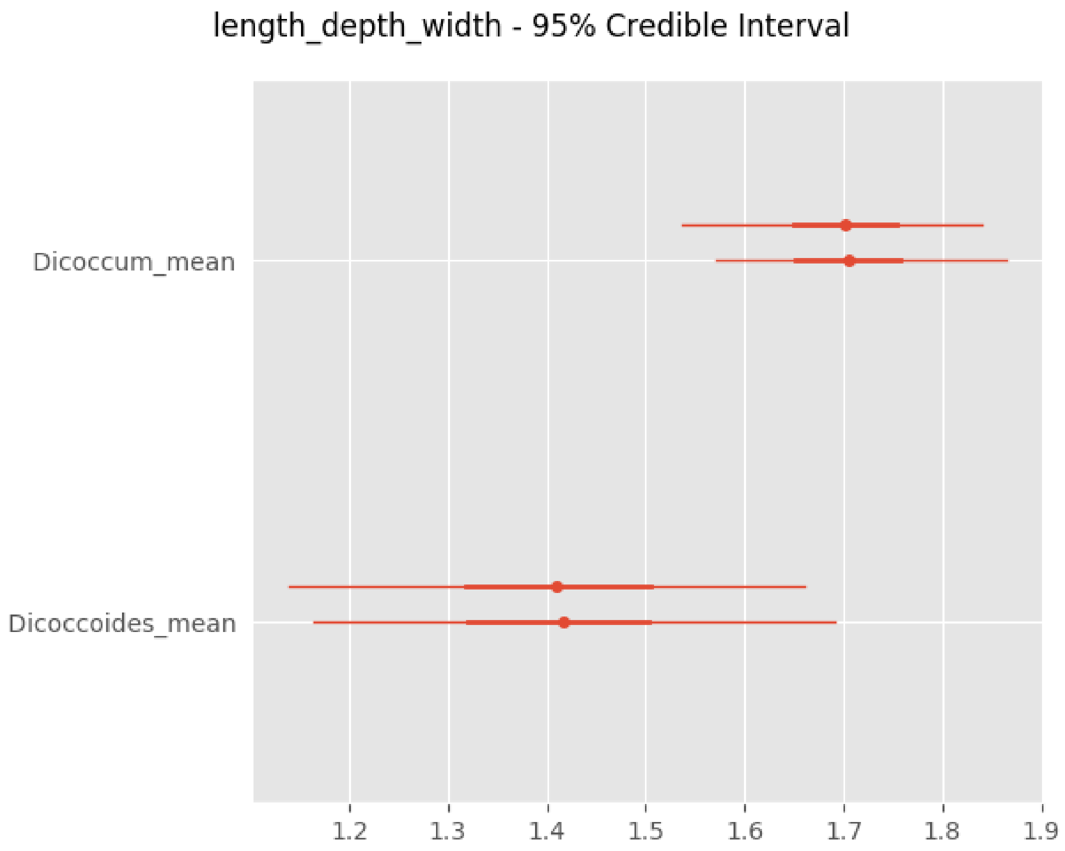


Figure 3.5: An example credible interval produced by this method

The full model is explained later in this chapter in a much more informative manner and fully defined in terms of relevance to results of this project. Listing:9 provides the python model implemented.

```python
def baysian_hypothesis_test(group1, group2, group1_name, group2_name):
    """
    Implements and uses the hypothesis test outlined as a robust replacement
    for the t-test
    @param group1 a numpy array to test
    @param group2 a numpy array to test
    @param group1_name the name of the first group
    @param group2_name the name of the second group
    @returns a summary dataframe
    """
    group1 = np.log10(group1)
    group2 = np.log10(group2)
    y = pd.DataFrame(dict(value=np.r_[group1, group2], group=np.r_[
                [group1_name]*len(group1), [group2_name]*len(group2)]))
    mu_m = y.value.mean()
    mu_s = y.value.std()*2
    with pm.Model() as model:
        group1_mean = pm.Normal('{0}_mean'.format(group1_name), mu_m, sd=mu_s)
        group2_mean = pm.Normal('{0}_mean'.format(group2_name), mu_m, sd=mu_s)
    sig_low = 1
    sig_high = 1000
    with model:
        group1_std = pm.Uniform('{0}_std'.format(
            group1_name), lower=sig_low, upper=sig_high)
        group2_std = pm.Uniform('{0}_std'.format(
            group2_name), lower=sig_low, upper=sig_high)

    with model:
        nu = pm.Exponential('nu_minus_one', 1/29.) + 1
    with model:
        lambda_1 = group1_std**-2
        lambda_2 = group2_std**-2
        group1 = pm.StudentT(group1_name, nu=nu, mu=group1_mean,
                        lam=lambda_1, observed=group1)
        group2 = pm.StudentT(group2_name, nu=nu, mu=group2_mean,
                        lam=lambda_2, observed=group2)
    with model:
        diff_of_means = pm.Deterministic(
            'difference of means', group1_mean - group2_mean)
        diff_of_stds = pm.Deterministic(
            'difference of stds', group1_std - group2_std)
        effect_size = pm.Deterministic('effect size',
                                diff_of_means /np.sqrt((group1_std**2 +
                                group2_std**2) / 2))
    with model:
        trace = pm.sample(2000, cores=2)
        return trace, pm.summary(trace, varnames=['difference of means',
                                        'difference of stds','effect size'])
```

Listing 9: Bayesian Model Function (part 1)

CT GUI Application Methods

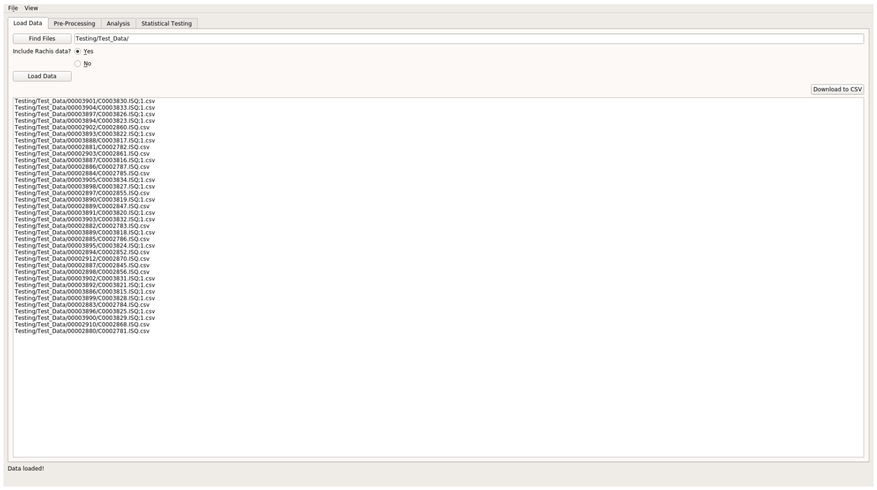
The CT GUI Application enables a researcher to perform analysis with a high degree of automation and ease of use. Here several of the elements of software engineering behind this application and its design are explored.

Windows

The CT GUI Application uses a tabbed interface, each tab carries out a specific task. This follows Gestalt themes of making user interfaces inherently clear with grouping together functionality in specific areas.

The Load Data Window

This window acts as the opening to the software, where a user can select loading of files and data. As well as options such as whether to load spike rachis information.



Testing/Test_Data/

Include Rachis data? [✓ Yes] No

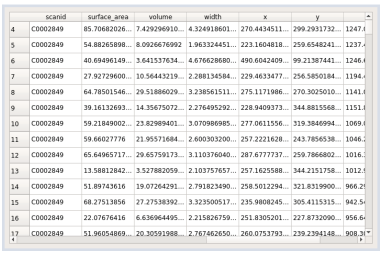
```
Testing/Test_Data/00003901/C0003830.ISQ;1.csv
Testing/Test_Data/00003904/C0003833.ISQ;1.csv
Testing/Test_Data/00003897/C0003826.ISQ;1.csv
Testing/Test_Data/00003894/C0003823.ISQ;1.csv
Testing/Test_Data/00002902/C0002860.ISQ.csv
Testing/Test_Data/00003893/C0003822.ISQ;1.csv
Testing/Test_Data/00003888/C0003817.ISQ;1.csv
Testing/Test_Data/00002881/C0002782.ISQ.csv
Testing/Test_Data/00002903/C0002861.ISQ.csv
Testing/Test_Data/00003887/C0003816.ISQ;1.csv
Testing/Test_Data/00002886/C0002787.ISQ.csv
Testing/Test_Data/00002884/C0002785.ISQ.csv
Testing/Test_Data/00003905/C0003834.ISQ;1.csv
Testing/Test_Data/00003898/C0003827.ISQ;1.csv
Testing/Test_Data/00002897/C0002855.ISQ.csv
Testing/Test_Data/00003890/C0003819.ISQ;1.csv
Testing/Test_Data/00002889/C0002847.ISQ.csv
Testing/Test_Data/00003891/C0003820.ISQ;1.csv
Testing/Test_Data/00003903/C0003832.ISQ;1.csv
Testing/Test_Data/00002882/C0002783.ISQ.csv
Testing/Test_Data/00003889/C0003818.ISQ;1.csv
Testing/Test_Data/00002885/C0002786.ISQ.csv
Testing/Test_Data/00003895/C0003824.ISQ;1.csv
Testing/Test_Data/00002894/C0002852.ISQ.csv
Testing/Test_Data/00002912/C0002870.ISQ.csv
Testing/Test_Data/00002887/C0002845.ISQ.csv
Testing/Test_Data/00002898/C0002856.ISQ.csv
Testing/Test_Data/00003902/C0003831.ISQ;1.csv
Testing/Test_Data/00003892/C0003821.ISQ;1.csv
Testing/Test_Data/00003886/C0003815.ISQ;1.csv
Testing/Test_Data/00003899/C0003828.ISQ;1.csv
Testing/Test_Data/00002883/C0002784.ISQ.csv
Testing/Test_Data/00003896/C0003825.ISQ;1.csv
Testing/Test_Data/00003900/C0003829.ISQ;1.csv
Testing/Test_Data/00002910/C0002868.ISQ.csv
Testing/Test_Data/00002880/C0002781.ISQ.csv
```

Data loaded!

Figure 3.6: The Initial Data Loading Window for the GUI

Table View

In addition to the tabs provide by the application, a table view is accessed via the system menu bar. This allows an alternative view of the data which is not pictorial.

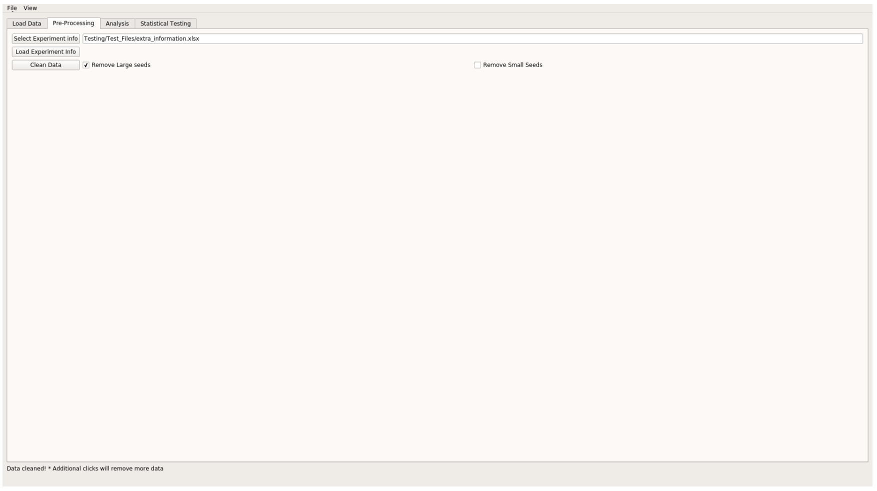


	scanid	surface_area	volume	width	x	y	
4	C0002849	85.70682026...	7.429296910...	4.324918601...	270.4434511...	299.2931732...	1247.
5	C0002849	54.88265898...	8.0926676992	1.963324451...	223.1604818...	259.6548241...	1237.
6	C0002849	40.69496149...	3.641537634...	4.676628680...	490.6042409...	99.21387441...	1246.
7	C0002849	27.92729600...	10.56443219...	2.288134584...	229.4633477...	256.5850184...	1194.
8	C0002849	64.78501546...	29.51886029...	3.238561511...	275.1171986...	270.3025010...	1141.
9	C0002849	39.16132693...	14.35675072...	2.276495292...	228.9409373...	344.8815568...	1151.
10	C0002849	59.21849002...	23.82989401...	3.070986985...	277.0611556...	319.3846994...	1069.
11	C0002849	59.66027776	21.95571684...	2.600303200...	257.2221628...	243.7856538...	1046.
12	C0002849	65.64965717...	29.65759173...	3.110376040...	287.6777737...	259.7866802...	1016.
13	C0002849	13.58812842...	3.527882059...	2.103757657...	257.1625588...	344.2151758...	1012.
14	C0002849	51.89743616	19.07264291...	2.791823490...	258.5012294...	321.8319900...	966.2
15	C0002849	68.27513856	27.27538392...	3.323500517...	235.9808245...	305.4115315...	942.5
16	C0002849	22.07676416	6.636964495...	2.215826759...	251.8305201...	227.8732090...	956.6
17	C0002849	51.96054869...	20.30591988...	2.767462650...	260.0753793...	239.2394148...	908.3

Figure 3.7: The Table view in used in the GUI

The Clean Data Window

The second tab this software introduces is a data cleaning and preparation window. This tab allows additional data to be loaded into an experiment such as scan names, genotype information etc. Options to clean the data by removing outliers is also presented with various options.



Testing/Test_Files/extra_information.xlsx

[✓ Remove Large seeds] Remove Small Seeds

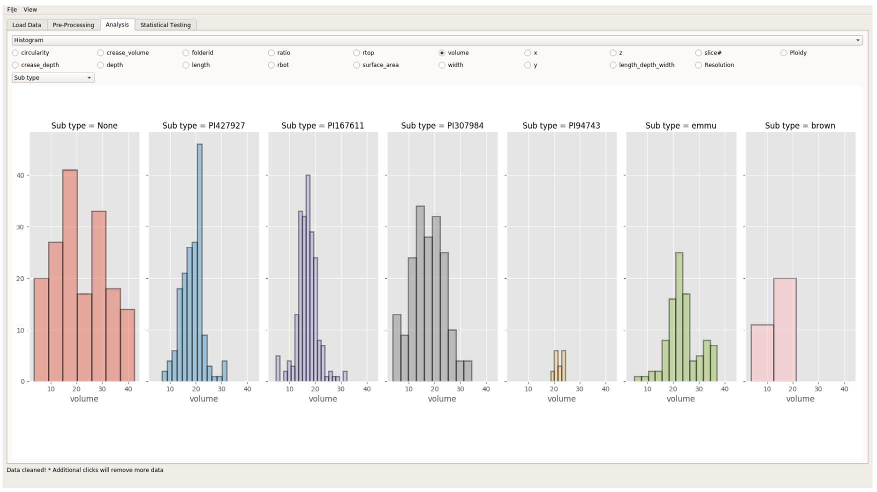
Data cleaned! * Additional clicks will remove more data

Figure 3.8: The Data Cleaning and Loading Window for the GUI

The Analysis Window

The third tab is made available once data has been loaded, access to it prior to loading data will cause the application to provide the user with a warning that it will not function correctly without loading information first.

This window gives a user the ability to rapidly investigate data, it enables the plotting of data via histograms and box plots, both of these are incredibly useful for initial investigation of data. It tells researchers a lot about the shape of their data and allows them to see an estimate of means, averages and spread of data.



Histogram

circularity, crease_volume, folderid, ratio, rtop, [✓ volume], x, z, slice#, Ploidy, crease_depth, depth, length, rbot, surface_area, width, y, length_depth_width, Resolution

Sub type

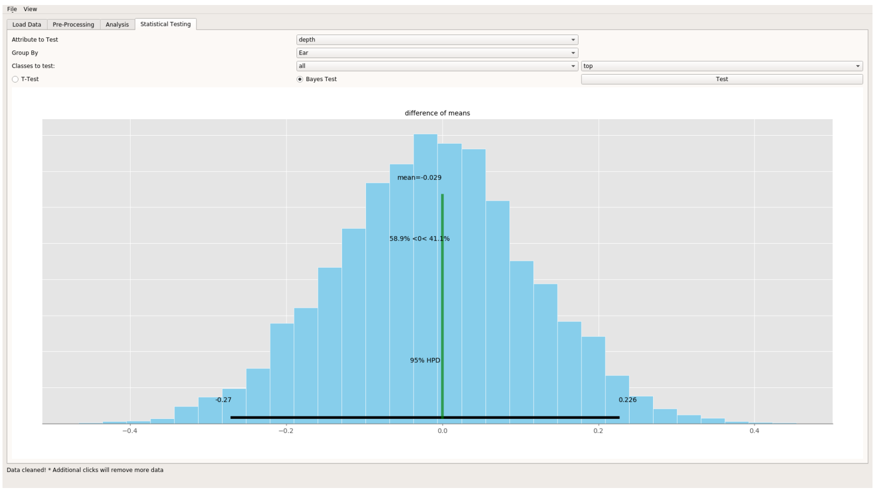
Data cleaned! * Additional clicks will remove more data

Figure 3.9: The Analysis/Investagation Window for the GUI

The Hypothesis Testing Window

The fourth and final window which this application presents is a method for performing hypothesis testing on the loaded data.

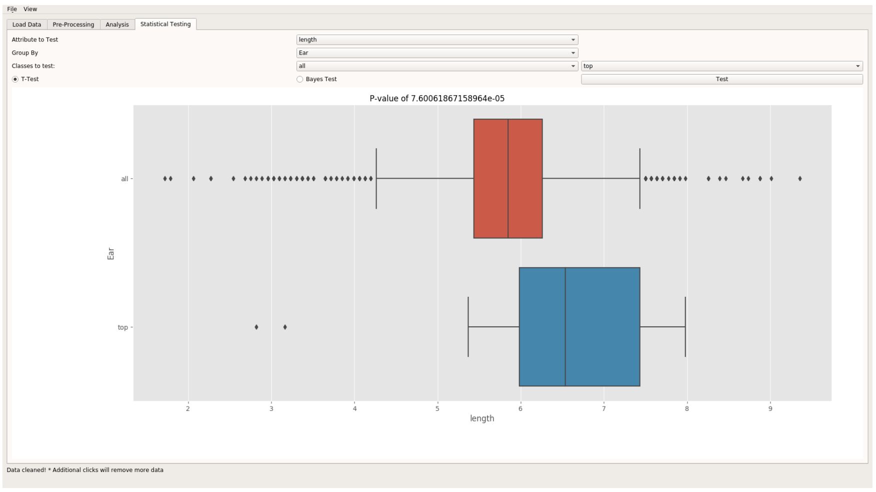
The method for implementing tests is flexible and more could be easily implemented; currently the Student's T-test, Welch's T-test and a Bayesian T-test are implemented to give a range of NHSTs.



Attribute to Test: depth
Group By: Ear
Classes to test: all, top
T-Test [✓ Bayes Test]

Data cleaned! * Additional clicks will remove more data

Figure 3.10: The Hypothesis Testing Window for the GUI with a difference of means plot



Attribute to Test: length
Group By: Ear
Classes to test: all, top
[✓ T-Test] Bayes Test

Data cleaned! * Additional clicks will remove more data

Figure 3.11: The Hypothesis Testing Window for the GUI with a T-test and boxplot

QT GUI Loading

The GUI is started from a minimal main method, as dictated by standard model-view-controller (MVC) convention. This leads to an *AppWindow* object which inherits from the *Qt*'s library *QMainWindow*. The MVC model comes into full effect in the constructor for this class.

This constructor assigns a *PyQt5* GUI file to itself. This allows for a predefined series of objects with rigid layout to be used in the software.

A series of setup functions are called which connect triggers (functions called when GUI elements are interacted with) to the correct controllers.

This main window spawns and hosts the controllers which dictate handling of their respective window (tab).

```
class AppWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
        self.setup_menu_functions()

        # Load in Controllers
        self.find_files_controller = FindFilesWindow(self, self.ui)
        self.analysis_controller = AnalysisWindow(self, self.ui)
        self.ui.master_tab.currentChanged.connect(self.update_analysis_view)
        self.pre_process_controller = PreProcessWindow(self, self.ui)
        self.stats_test_controller = StatsTestWindow(self, self.ui)

        # Set GUI icon for beautification
        self.setWindowIcon(QtGui.QIcon('./images/logo.png'))

        # init some variables
        # Words cannot describe the importance of this object
        self.data = None

        # init states
        self.setup_default_states()
        self.show()
```

Listing 10: The AppWindow Class' initialisation function

Connecting Signals from GUI to Function Calls

In example:11 lines 11,12,13 show how the UI object has individual members, each of these has a "clicked" trigger, which when triggered calls a function pointer which this controller handles.

```
1  class FindFilesWindow():
2      def __init__(self, window, ui):
3          self.ui = ui
4          self.window = window
5          self.connect_view_functions()
6      def connect_view_functions(self):
7          # Load data page
8          self.ui.btn_find_files.clicked.connect(self.search_for_files)
9          self.ui.btn_load_data.clicked.connect(self.find_files)
10         self.ui.btn_to_csv.clicked.connect(self.save_file_dialog)
```

Listing 11: Example of connecting function pointers

Integration with CT Analysis Library

The integration with the CT Analysis Library is frequent and spread throughout this application and functions as a key dependency for data organisation.

One example of this is in the load data function shown in example:12. Here the custom exception classes are imported and used for data already loaded, as well as the loading functions itself.

```
def find_files(self):
    try:
        if self.window.get_data() is not None:
            raise DataAlreadyLoaded
        self.window.set_data(CTGUIData(self.ui.directory.text(),
                                       self.ui.rdb_rachis_yes.isChecked()))
        if self.window.get_data():
            self.ui.btn_to_csv.setEnabled(True)
            self.ui.tab_preprocess.setEnabled(True)
            self.ui.tab_analysis.setEnabled(True)
            self.set_files_list()
            self.ui.lbl_status.setText('Data loaded!')
    except TypeError as e:
        QMessageBox.warning(self.window, "Finding Files Error",
                            "Couldn't find files in given location")
    except DataAlreadyLoaded as e:
        QMessageBox.warning(self.window, "Data Already Loaded",
                            "Sorry! You've already loaded in data")
```

Listing 12: The load data function from the load_data window

Plot Window Rendering

Plots where rendered in this application by creating a custom object which inherited from the *matplotlib* backend *qt5agg*. This object is both a *Qt* widget and a *matplotlib* figure. This class provided a straightforward mechanism that would allow for any style of plot (which *matplotlib* supports) to be created.

This object also acts as a template class for actual implementations, each window type of this application that uses plots (*Analysis Tab*; *Hypothesis Testing Tab*) creates a custom *MyMplCanvas* child class. To specifically handle inputs which are unique to that window.

The unique inheritance of this class allows it to be treated as a *Qt* widget, when it is created by a controller it can be added to the view with no alterations required.

```python
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
...
  class MyMplCanvas(FigureCanvas):
      """Ultimately, this is a QWidget (as well as a FigureCanvasAgg, etc.)."""

      def __init__(self, parent=None, window=None, df=None,
                   column=None, width=5, height=4,
                   dpi=100, plot_type=None, group_by=None, ttest=False):

          if group_by == 'None':
              self.group_by = None
          else:
              self.group_by = group_by
          self.ttest = ttest
          self.window = window
          self.plot_type = plot_type
          self.column = column

          if plot_type == 'histogram' and self.group_by is not None:
              self.fig = self.facet_hist(df, column, group_by)
              FigureCanvas.__init__(self, self.fig)
          else:
              self.fig = Figure(figsize=(width, height), dpi=dpi)
              self.axes = self.fig.add_subplot(111)
              FigureCanvas.__init__(self, self.fig)
              self.compute_initial_figure(self.axes, df)

          self.setParent(parent)
          FigureCanvas.setSizePolicy(self,
                                     QtWidgets.QSizePolicy.Expanding,
                                     QtWidgets.QSizePolicy.Expanding)
          FigureCanvas.updateGeometry(self)
```

Listing 13: The Matplotlib FigureCanvas Polymorphic Class

Creating GUI Elements When Required

One of the key features of this project, and this GUI application specifically is the ability to handle dynamic input. The loading additional experiment information allows for the opportunity for never-before seen data to be presented. This allows users to define custom classification and testing groups. An example from the research element of this project is shown in figure:3.12.

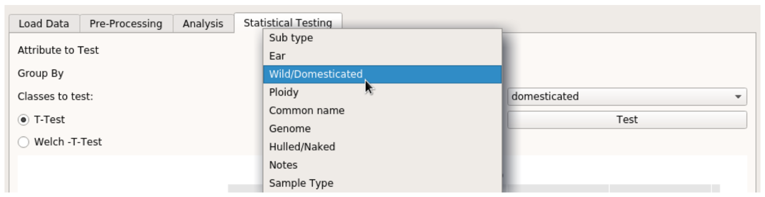


Figure 3.12: Showing how groups are loaded from columns for potential data splitting and testing

Here in example:14 lines 16,17 show that combo buttons are created for the *Hypothesis Testing* window to display to a user. A similar system is added for the *Analysis Window* where decisions are carried out as to what columns could be potentially used for object grouping. This is calculated by dividing the columns contents, if it is numeric and has repeating elements or if the contents are text then they are (generally) applicable for use in grouping grains for testing or plotting.

```
1  class TestWindowView():
2      def __init__(self,df,ui,plot_type):
3          """
4          Given a dataframe and a layout spec
5          this class populates appropriate radio button functionality
6          """
7          self.df = df
8          self.ui = ui
9          self.plot_type = plot_type
10         self.ui.cb_test_grouping.clear()
11         self.ui.cb_test_attribute.clear()
12         # Line up attributes
13         self.ui.cb_test_attribute.addItems(list(
14             filter(lambda x: is_numeric_dtype(df[x]), df.columns)))
15
16         # Find groups
17         self.ui.cb_test_grouping.addItems(
18             self.find_candidates_for_grouping(self.df))
19
20     def find_candidates_for_grouping(self, df):
21         lst = []
22         for c in df.columns:
23             if len(df[c].unique()) < 30:
24                 if not is_numeric_dtype(df[c]):
25                     lst.append(c)
26         return lst
```

Listing 14: The Hypothesis Testing Window class

Dynamically Generating Plots

This application provides the ability to make plots instantly, without requiring input from the user to refresh or reload a window.

To do this, the existing canvas window is repainted based on parameter input. Example:15 shows how this is implemented. A series of conditionals based on UI elements provides data about what data to use and type of plot to use also.

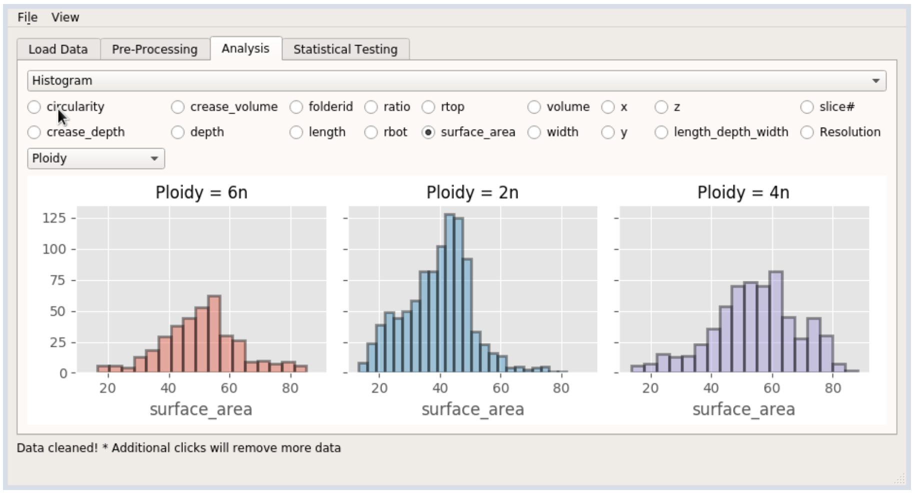
```python
def compute_initial_figure(self, axes, df):
    try:
        if self.plot_type == 'histogram':
            sns.distplot(df[self.column], ax=self.axes,
                            kde=False, hist_kws=dict(edgecolor="k", linewidth=2))
        elif self.plot_type == 'boxplot' or self.plot_type == 'welch':
            sns.boxplot(data=df, x=self.column,
                           y=self.group_by, ax=self.axes)
            if self.ttest:
                u = list(df[self.group_by].unique())
                s1 = df[df[self.group_by] == u[0]][self.column]
                try:
                    s2 = df[df[self.group_by] == u[1]][self.column]
                except IndexError:
                    s2 = s1
                if self.plot_type == 'welch':
                    p = perform_t_test(s1, s2, equal_var=False)
                else:
                    p = perform_t_test(s1, s2)
                self.fig.suptitle('P-value of {0}'.format(p))
            elif self.plot_type == 'bayes':
                u = list(df[self.group_by].unique())
                s1 = np.array(df[df[self.group_by] == u[0]][self.column])
                try:
                    s2 = np.array(df[df[self.group_by] == u[1]][self.column])
                    trace, x = baysian_hypothesis_test(s1, s2, u[0], u[1])
                    plot_difference_of_means(trace, ax=self.axes)
```

Listing 15: Example code of how figures are computed and implemented using *Seaborn* and *Matplotlib*

Creating Facet Plots

To compare data in more than a single dimension the *FacetFrid* function from the *Seaborn* library is used. This provides a simple method of creating multiple plots in a single figure.

An example of this is shown when dividing data by ploidy and comparing distributions via histograms in figure:3.13.



Histogram

● surface_area

Ploidy

Data cleaned! * Additional clicks will remove more data

Figure 3.13: Example of facet plotting by ploidy by grain surface area

The implementation of *facet plots* would be simple, however the ability to create sensible plots that organise into a well spaced grid requires additional input.

Example:16 shows on lines 3,5,7 how decisions are made to best arrange the grid. Decisions are made based on the unique number of columns a grouping variable provides. As well this, additional arguments are used to indicate further aesthetics of the plots.

```python
1  def facet_hist(self, df, column, group_by):
2      col_wrap = len(df[group_by].unique()) // 2
3      if col_wrap < 4:
4          col_wrap = None
5      g = sns.FacetGrid(df, hue=group_by, col=group_by,
6                        col_wrap=col_wrap)
7      g = g.map(sns.distplot, column, kde=False,
8                hist_kws=dict(edgecolor="k", linewidth=2))
9      g.fig.tight_layout()
10     return g.fig
```

Listing 16: Using Facet wrapping to provide

Data Analysis Methods

Bayesian Modelling

Chapter 4

Results

Chapter 5

Discussion

Similar Research

Alternate Solutions

Chapter 6

Critical Evaluation

Organisational Methods

Relevance to Degree

Time Management

Collaborative Work

Other Issues

Appendix A

Software Packages Used

Libraries

Table A.1: Software libraries used

Seaborn	Scipy	Sklearn
MATLAB Image Processing Toolbox	Numpy	Matplotlib
Statsmodels	Pymc3	Xlrd
PyQt5	gcc	Pip

Tools

Table A.2: Software tools used

MATLAB	Python Debugger (PDB)	IPython
Emacs	git	org-mode
Tomviz	ImageJ	PlantUML

Appendix B

Glossary

Table B.1: Dictionary for Terms and acronyms

Term	**Definition**
µ-CT	Micro Computed Tomography
Genotype	A genetically distinct individual or group
Phenotype	A physical/measurable trait
Alleles	A variant of a gene
Genus	Classification ranking, below the *family* grouping
Genome	The complete genetic make up of an organism, which defines its individuality
Morphometric	The shape and form of an organism
GUI	Graphical User Interface
PCA	Principal Component Analysis
Spike	A singular stalk of wheat
Spikelet	A group of seeds all forming from the same node in a spike
MVC	Model View Controller - A design pattern for GUIs
OOP	Object Orientated Programming
$X \sim \mathcal{N}(\mu, \sigma^2)$	Notation for Normal (Gaussian) Distribution

Appendix C

Wheat Varieties

Table C.1: Wheat used in this work and their common names

Species name	**Common name**
Triticum monococcum	Einkorn Domesticate
Triticum boeticum	Einkorn Wild
Triticum durum	Pasta Wheat
Triticum dicoccoides	Emmer Domesticate
Triticum dicoccum	Emmer Wild
Triticum ispahanicum	n/a
Triticum timopheevii	n/a
Triticum spelta	Spelt
Triticum aestivum	Bread Wheat
Triticum compactum	Club Wheat

Appendix D

Code Segments and Examples

Setup.py

```python
from setuptools import setup
setup(name='CT_Analysing_Library',
      version='0.2',
      description='Library used for CT grain analysis at the NPPC',
      url='https://github.com/SirSharpest/CT_Analysing_Library',
      author='Nathan Hughes',
      author_email='nathan1hughes@gmail.com',
      license='MIT',
      packages=['ct_analysing_library'],
      install_requires=['pandas',
                        'numpy',
                        'matplotlib',
                        'seaborn',
                        'scipy',
                        'sklearn',
                        'statsmodels',
                        'pymc3',
                        'xlrd'],
      zip_safe=True)
```

Listing 17: The *setup.py* configuration for the CT Analyser Library

Self-Documenting Code Example

```
def get_spike_info(self, excel_file, join_column='Folder#'):
    """
    This function should do something akin to adding additional
    information to the data frame

    @note there is some confusion in the NPPC about whether to use
    folder name or file name as the unique id when this is made into
    end-user software, a toggle should be added to allow this

    @param excel_file a file to attach and read data from
    @param join_column if the column for joining data is
    different then it should be stated
    """
        # Grab the linking excel file
        info = pd.read_excel(excel_file,
                             index_col='Folder#')

        features = list(info.columns)
        # Lambda to look up the feature in excel spreadsheet
        def look_up(x, y): return info.loc[x['folderid']][y]

        # Lambda form a series (data row) and apply it to dataframe
        def gather_data(x): return pd.Series(
            [look_up(x, y) for y in features])

        self.df[features] = self.df.apply(gather_data, axis=1)
    except KeyError as e:
        print('Error matching data')
        print(e)
        raise NoDataFoundException
    except AttributeError as e:
        print(e)
        raise NoDataFoundException
```

Listing 18: Example of code documentation and readability from *data_transforms.py*

Custom Documentation Generator

```
(defun populate-org-buffer (buffer filename root)
  (goto-char (point-min))
  (let ((to-insert (concat "* " (replace-regexp-in-string root "" filename) "\n") ))
    (while (re-search-forward
            (rx (group (or "def" "class"))
                space
                (group (+ (not (any "()"))))
                (? "(" (* nonl) "):" (+ "\n") (+ space)
                   (= 3 "\"")
                   (group (+? anything))
                   (= 3 "\"")))
            nil 'noerror)
      (setq to-insert
            (concat
             to-insert
             (if (string= "class" (match-string 1))
                 "** "
               "*** ")
             (match-string 2)
             "\n"
             (and (match-string 3)
                  (concat (match-string 3) "\n")))))helm-semantic-or-imenu
    (with-current-buffer buffer
      (insert to-insert))))
(defun org-documentation-from-dir (&optional dir)
  (interactive)
  (let* ((dir  (or dir (read-directory-name "Choose base directory: ")))
         (files (directory-files-recursively dir "\py$"))
         (doc-buf (get-buffer-create "org-docs")))
    (dolist (file files)
      (with-temp-buffer
        (insert-file-contents file)
        (populate-org-buffer doc-buf file dir)))
    (with-current-buffer doc-buf
      (org-mode))))
```

Listing 19: Custom lisp code for generating easy to read documentation

Appendix E

Sprint Timeline

Sprint - Week 0

Initial planning was taken out, discussions with researchers at the National Plant Phenomics Centre (NPPC), to create a general set of targets and research goals.

A website was built in order to host weekly progress reports, this was used to share with supervisors and with staff at the NPPC. It also provided a list of discussion points to go through at weekly meetings.

A bug was identified in image analysis software, this was raised to be fixed a later date.

A literature review was taken out to highlight the novelty of this research, as well as current trends in the field in terms of analysis and known/accepted information.

Sprint - Week 1

Initial running of grain analysis software was performed multiple times, as per instructions in literature [1], multiple parameters for minimum and maximum expected sizes of grains needed to be tested. Data which was produced was very noisy and would require further work.

Spike work was carried out in investigating the potential of using a skeletonising method on the wheat spikes. The hope behind this was to simplify structure in a three dimensional structure, of 1 pixel thick lines. This technique is often used in plant root analysis [31,32]. Experimentation with these methods were technically challenging and a decision was made to revisit if time permitted at the end of the project.

A key function of the CT Analysis Library was created, showing in listing:4. The method enables additional experiment information to be joined with extracted seed data. This provides a way of grouping seeds into more useful groupings than just their scanning data.

An issue was identified in the choices of testing which are typically used in these studies, the use of ANOVA and Student's T-Test, for example, are best used with parametric data, that is to say data where distribution is normally distributed. Further reading into this presented the use of Box Cox transforms as a method to counter these issues.

An issue was raised; a method for visualising outliers in the data could provide greatly beneficial insight into finding errors. If time was available at the end of the project, this would be explored further.

Sprint - Week 2

Spikes of wheat are scanned in two separate imaging cycles sometimes, this is because the tube used by the µ-CT machine are 10cm tall and often a spike will exceed this. In order for full analysis to be carried out these separate scans need to be rejoined. A method for doing this was added to the python Analysis Library.

An initial Model-View-Controller (MVC) model was constructed for how a GUI might take form around. Using this several wire-frames were created

An idea for how data could be cleaned was created by using the information found in previous studies [1]. From this reported data minimum and maximum expected size could be assumed for wheat grains in terms of volume.

Decisions to move the CT Grain Analysing Library towards an object orientated model were made during this sprint, after evaluating the potential of a functionally programmed model or a object one. Handling everything in terms of classes was decidedly easier in terms of understanding how to use the library.

Sprint - Week 3

A lot of progress was made on constructing a GUI here, dynamic plotting was put together as a proof of this concept. Histograms of the data were able to be made by using the mouse to select which attribute to measure.

GUI elements such as navigation, data tabs and file menus were added to make more clear to the user how functionality should work.

A novel and new method for generating documentation was created specifically for documenting this software library. Using a plain text format, doc strings from python code was used to make a single easy to read PDF to be distributed with the software package.

Another piece of functionality was introduced to the GUI that would enable the user to view the data they had loaded as a single 2 dimensional data frame. This is for quickly viewing data as a single source and all together. This had arisen as a functionality desired by researchers in a previous weeks meeting.

Sprint - Week 4

A new method for watershedding was developed and deployed in the data extraction pipeline, this moved towards using euclidean distance transforms. Allowing for the more complex shapes which this data set presented as a problem to be fully and properly segmented.

Further work was done in terms of the GUI to enable matplotlib and seaborn library integration with the dynamic plotting features which were developed previously.

Custom exceptions were written for the CT Analysing Library, these exceptions allow for more detailed feedback to a programmer when they perform actions which might result in erroneous data processing. In particular a "NoDataFoundException" was made for when empty data files where found in searching.

Current versions of doc strings were enhanced to match with newer functionality which had been added in recent weeks, and documentation was regenerated.

Sprint - Week 5

Functions were added to the CT Analysing Library which would allow for aggregation of data columns, this meant that averaging functions could be performed on a per-spike basis. Mean, standard deviation, sum and count functions were applied by default.

Further research was carried out into data transforms which are commonly used in data science; Principal component analysis, linear modelling and multivariate analysis were topics researched as possibilities for inclusion in this library.

Sprint - Week 6

Preparation for a mid-project demo took up the majority of this sprint, ensuring data which had been gathered to date was well understood was vital.

A presentation was created using data extracted via the CT Analysing Library, this was in the mode of figures and graphs which could then be used to better explain the theme of the project.

Sprint - Week 7

An initial implementation of principal component analysis was tested and found to be useful, a considerable amount of time was dedicated to refactoring the code-base to allow for new data transforms to be easily incorporated.

Additions to the graphing section of the library also allowed for splitting data into multiple plots which could then be featured side-by-side, separation of data helped contribute to seeing the differences in the data.

Some initial ideas for unit testing were written up, an investigation into which platform and library were best to use was carried out. This had to be taken into consideration as testing a GUI required some additional software to simulate a user testing widgets and onscreen objects.

Sprint - Week 8

With data extraction parameters finalised, a new watershedding method implemented and a more robust selection process implemented the data was now fit for hypothesis to be tested.

Through meetings with researching staff, the five key hypothesis/null-hypothesis had been formed and the grouping of the data well defined. This allowed for initial Null-Hypothesis-Significance-Testing (NHST) to be carried out. The initial results found were not representative of the data. A requirement for new hypothesis testing was now required.

The *PyTest* library was decided upon, this was used as a unit testing framework for both the Analysis Library and the GUI software. In total around 50 tests were devised.

Sprint - Week 9

More graphing options were added to the GUI, these came from the boxplot functionality which the updated CT Analysing Library made available. A new type of grouping by sub-type was added. This let multiple plots be displayed for the same feature i.e. viewing a histogram of volume but split by genotype. This new feature required a rework of the dynamic plotting functionality. This refactoring took a considerable amount of time and work.

An initial solution to hypothesis testing was added in the form of Welch and Student T-Tests to first the CT Analysing Library and then to the GUI also. The GUI now had an additional tab added specifically for testing groups of data.

Sprint - Week 10

A previous issue was highlighted in meetings of the data being ill-suited to normally accepted statistical testing and NHST in general. Through research a new method was devised that was able to use Bayesian statistics to carry out hypothesis testing [30]. This new method of hypothesis testing was added into the CT Analysing Library, requiring a lot of work in data transforming and preparation in order for correct, robust and repeatable model fitting to be carried out. Previous research questions were repeated, using this method. This time new and significant findings were uncovered.

Sprint - Week 11

A feature freeze was enforced during the 11th week of the project. This was to ensure time was allocated for completing work, finalising unit tests and documentation. The > 50 unit tests were finalised, completed and ensured to be passing. Documentation was completed and exported.

References

[1] N. Hughes, K. Askew, C. P. Scotson, K. Williams, C. Sauze, F. Corke, J. H. Doonan, and C. Nibau, "Non-destructive, high-content analysis of wheat grain traits using X-ray micro computed tomography," *Plant Methods*, vol. 13, 2017.

[2] H. Özkan, A. Brandolini, R. Schäfer-Pregl, and F. Salamini, "AFLP Analysis of a Collection of Tetraploid Wheats Indicates the Origin of Emmer and Hard Wheat Domestication in Southeast Turkey," *Molecular biology and evolution*, vol. 19, no. 10, pp. 1797–1801, oct 2002. [Online]. Available: http://academic.oup.com/mbe/article/19/10/1797/1259152

This paper discusses the origin of wheat domestication and provides evidence for it's origin in society

[3] B. Shiferaw, M. Smale, H.-J. Braun, E. Duveiller, M. Reynolds, and G. Muricho, "Crops that feed the world 10. Past successes and future challenges to the role played by wheat in global food security," *Food Security*, vol. 5, no. 3, pp. 291–317, jun 2013. [Online]. Available: http://link.springer.com/10.1007/s12571-013-0263-y

[4] E. J. Finnegan, B. Ford, X. Wallace, F. Pettolino, P. T. Griffin, R. J. Schmitz, P. Zhang, J. M. Barrero, M. J. Hayden, S. A. Boden, C. A. Cavanagh, S. M. Swain, and B. Trevaskis, "Zebularine treatment is associated with deletion of <i>FT</i> - <i>B1</i> leading to an increase in spikelet number in bread wheat," *Plant, Cell & Environment*, apr 2018. [Online]. Available: http://doi.wiley.com/10.1111/pce.13164

[5] I. Aleksejeva, "EU Experts' Attitude Towards Use of GMO in Food and Feed and Other Industries," *Procedia - Social and Behavioral Sciences*, vol. 110, pp. 494–501, jan 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S187704281305533X

[6] T. Twardowski and A. Małyska, "Uninformed and disinformed society and the GMO market," *Trends in Biotechnology*, vol. 33, no. 1, pp. 1–3, jan 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167779914002327

[7] M. Lynas, "Opinion | With G.M.O. Policies, Europe Turns Against Science - The New York Times." [Online]. Available: https://www.nytimes.com/2015/10/25/opinion/sunday/with-gmo-policies-europe-turns-against-science.html?ref=todayspaper

[8] G. Charmet, "Wheat domestication: Lessons for the future," *Comptes Rendus - Biologies*, vol. 334, no. 3, pp. 212–220, 2011. [Online]. Available: http://dx.doi.org/10.1016/j.crvi.2010.12.013

[9] R. T. Furbank and M. Tester, "Phenomics - technologies to relieve the phenotyping bottleneck," *Trends in Plant Science*, vol. 16, no. 12, pp. 635–644, 2011. [Online]. Available: http://dx.doi.org/10.1016/j.tplants.2011.09.005

[10] B. Naumann, M. Eberius, and K.-J. Appenroth, "Growth rate based doseresponse relationships and EC-values of ten heavy metals using the duckweed growth inhibition test (ISO 20079) with Lemna minor L. clone St," *Journal of Plant Physiology*, vol. 164, no. 12, pp. 1656–1664, dec 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0176161706003154

[11] B. M. Prasanna, J. L. Araus, J. Crossa, J. E. Cairns, N. Palacios, B. Das, and C. Magorokosho, "High-Throughput and Precision Phenotyping for Cereal Breeding Programs," in

Cereal Genomics II. Dordrecht: Springer Netherlands, 2013, pp. 341–374. [Online]. Available: http://link.springer.com/10.1007/978-94-007-6401-9{_}13

[12] J. F. Humplík, D. Lazár, A. Husičková, and L. Spíchal, "Automated phenotyping of plant shoots using imaging methods for analysis of plant stress responses a review," *Plant Methods*, vol. 11, no. 1, p. 29, dec 2015. [Online]. Available: http://www.plantmethods.com/content/11/1/29

[13] J. Roussel, F. Geiger, A. Fischbach, S. Jahnke, and H. Scharr, "3D Surface Reconstruction of Plant Seeds by Volume Carving: Performance and Accuracies," *Frontiers in Plant Science*, vol. 7, no. June, pp. 1–13, 2016. [Online]. Available: http://journal.frontiersin.org/Article/10.3389/fpls.2016.00745/abstract

[14] G. Wang, H. Yu, and B. De Man, "An outlook on x-ray CT research and development," *Medical Physics*, vol. 35, no. 3, pp. 1051–1064, feb 2008. [Online]. Available: http://doi.wiley.com/10.1118/1.2836950

[15] V. M. Jhala and V. S. Thaker, "X-ray computed tomography to study rice (Oryza sativa L.) panicle development," *Journal of Experimental Botany*, vol. 66, no. 21, pp. 6819–6825, 2015.

[16] S. R. Tracy, C. R. Black, J. A. Roberts, C. Sturrock, S. Mairhofer, J. Craigon, and S. J. Mooney, "Quantifying the impact of soil compaction on root system architecture in tomato (Solanum lycopersicum) by X-ray micro-computed tomography." *Annals of botany*, vol. 110, no. 2, pp. 511–519, 2012.

[17] R. Metzner, A. Eggert, D. van Dusschoten, D. Pflugfelder, S. Gerth, U. Schurr, N. Uhlmann, and S. Jahnke, "Direct comparison of MRI and X-ray CT technologies for 3D imaging of root systems in soil: Potential and challenges for root trait quantification," *Plant Methods*, vol. 11, no. 1, pp. 1–11, 2015.

[18] Y. M. Staedler, D. Masson, and J. Schönenberger, "Plant Tissues in 3D via X-Ray Tomography: Simple Contrasting Methods Allow High Resolution Imaging," *PLoS ONE*, vol. 8, no. 9, 2013.

[19] F. J. Leigh, I. Mackay, H. R. Oliveira, N. E. Gosman, R. A. Horsnell, H. Jones, J. White, W. Powell, and T. A. Brown, "Using diversity of the chloroplast genome to examine evolutionary history of wheat species," *Genetic Resources and Crop Evolution*, vol. 60, no. 6, pp. 1831–1842, 2013.

[20] J. Cockram, H. Jones, F. J. Leigh, D. O'Sullivan, W. Powell, D. A. Laurie, and A. J. Greenland, "Control of flowering time in temperate cereals: Genes, domestication, and sustainable productivity," *Journal of Experimental Botany*, vol. 58, no. 6, pp. 1231–1244, 2007.

[21] V. C. Gegas, A. Nazari, S. Griffiths, J. Simmonds, L. Fish, S. Orford, L. Sayers, J. H. Doonan, and J. W. Snape, "A Genetic Framework for Grain Size and Shape Variation in Wheat," *The Plant Cell*, vol. 22, no. 4, pp. 1046–1056, 2010. [Online]. Available: http://www.plantcell.org/lookup/doi/10.1105/tpc.110.074153

[22] S. R. Tracy, J. F. Gómez, C. J. Sturrock, Z. A. Wilson, and A. C. Ferguson, "Non-destructive determination of floral staging in cereals using X-ray micro computed tomography (μCT)," *Plant Methods*, vol. 13, no. 1, pp. 1–12, 2017.

[23] A. Andrews, "Scrum Of One: How to Bring Scrum into your One-Person Operation." [Online]. Available: https://www.raywenderlich.com/162654/scrum-one-bring-scrum-one-person-operation

[24] C. Ozgur, U. Hall, T. Colliau, G. R. J. o. D. . . . , and U. 2017, "MatLab vs. Python vs. R," *Journal of Data Science*, 2016.

[25] G. van Rossum, "PEP 8 – Style Guide for Python Code | Python.org." [Online]. Available: https://www.python.org/dev/peps/pep-0008/

[26] MATHWORKS, "MATLAB - MathWorks - MATLAB & Simulink," 2017. [Online]. Available: https://www.mathworks.com/products/matlab.html

[27] J. L. Pfaltz, "Sequential Operations in Digital Picture Processing," *Journal of the ACM*, vol. 13, no. 4, pp. 471–494, oct 1966. [Online]. Available: http://portal.acm.org/citation.cfm?doid=321356.321357

[28] J. M. Kleinberg, "Two algorithms for nearest-neighbor search in high dimensions," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing - STOC '97*. New York, New York, USA: ACM Press, 1997, pp. 599–608. [Online]. Available: http://dl.acm.org/citation.cfm?id=258533.258653

[29] T. E. Oliphant, "SciPy: Open source scientific tools for Python," *Computing in Science and Engineering*, vol. 9, pp. 10–20, 2007. [Online]. Available: http://www.scipy.org/

[30] J. K. Kruschke, "Bayesian estimation supersedes the t test," *Journal of Experimental Psychology: General Version of May*, vol. 31, 2012. [Online]. Available: http://www.indiana.edu/{~}kruschke/BEST/BEST.pdf

[31] S. Mairhofer, C. J. Sturrock, M. J. Bennett, S. J. Mooney, and T. P. Pridmore, "Extracting multiple interacting root systems using X-ray microcomputed tomography," *Plant Journal*, vol. 84, no. 5, pp. 1034–1043, 2015.

[32] K. R. Daly, S. R. Tracy, N. M. Crout, S. Mairhofer, T. P. Pridmore, S. J. Mooney, and T. Roose, "Quantification of root water uptake in soil using X-ray computed tomography and image-based modelling," *Plant, Cell & Environment*, 2017. [Online]. Available: http://doi.wiley.com/10.1111/pce.12983