PIPELINE AND TOOLS FOR THE ANALYSIS OF MULTIPLEXED ELISA DATA

Jesse A. Asimeng

A thesis presented in partial fulfilment of the requirements for the degree of Master of Science (Molecular Biology) in the Faculty of Medicine and Health Sciences at Stellenbosch University.

Supervisors:

Prof. Gerard Tromp

Dr. Elizna Maasdorp

Prof. Gian van der Spuy

March 2023

DECLARATION

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

March 2023

List of research outputs

1) Poster presentation at the sixty-sixth Annual Academic Day, Faculty of Medicine and Health Sciences, Stellenbosch University, August 2022.
2) Oral presentation at the Division of Molecular Biology and Human Genetics Seminar, Stellenbosch University, May 2022.
3) Poster presentation at the BIO2022: Bioscience, Big Data & the 4th Industrial Revolution Conference, Stellenbosch, April 2022.
4) Oral presentation at the South African Society for Bioinformatics & South African Genetics Society Students' Symposium, Stellenbosch, April 2022.
5) Flash oral presentation at the Division of Molecular Biology and Human Genetics Research Retreat, Stellenbosch University, February 2022.
6) ePoster presentation at the sixty-fifth annual academic day, Faculty of Medicine and Health Sciences, Stellenbosch University, September 2021.

Abstract

A cornerstone of scientific progress is independent data verification. It is, therefore, necessary to develop robust analysis pipelines that can ensure reproducible and verifiable analyses. The pipeline should also record all steps and software that generated the results. The analysis of multiplexed ELISA data (Luminex data) can be challenging due to its complexity and variability. In particular, the data preprocessing stage has many steps and is often ad hoc, leading to inconsistency, non-standard approaches and lack of reproducibility. An existing in-house data preprocessing pipeline, the Luminex Pipeline, addresses some of the aforementioned challenges. However, there remains substantial work to extend its utility, robustness, and overall reproducibility. Thus, in this work, I improved the summary statistic reports by using Rmarkdown and implemented unit testing of pipeline components using the R Testthat package. Unit testing ensured the greater robustness of the code, which was compiled into an R package. The pipeline execution was also automated by using the Nextflow workflow management system. Finally, I deployed the pipeline in a Singularity container for execution on any platform including high-performance computing clusters.

Opsomming

'n Hoeksteen van wetenskaplike vooruitgang is onafhanklike databevestiging. Dit is dus nodig om robuuste ontledingspyplyne te ontwikkel wat reproduseerbare en bevestigbare ontledings kan verseker. Die pyplyn moet ook alle stappe en sagteware wat die resultate gegenereer het, aanteken. Die ontleding van vermenigvuldige ELISA-data (Luminex-data) kan uitdagend wees weens die kompleksiteit en veranderlikheid daarvan. Die data-voorverwerkingstadium het veral baie stappe en is dikwels ad hoc, wat lei tot inkonsekwentheid, benaderings wat nie gestandardiseerd is nie en 'n gebrek aan reproduseerbaarheid. 'n Bestaande interne datavoorverwerkingspyplyn, die Luminex-pyplyn, spreek sommige van die voorgenoemde uitdagings aan. Die uitbreding van die bruikbaarheid, robuustheid en algehele reproduseerbaarheid van die huidige pyplyn vereis nog baie werk. In hierdie werk het ek dus die opsommende statistiese verslae verbeter deur Rmarkdown te gebruik en eenheidstoetsing van pyplynkomponente geïmplementeer deur die gebruik van R Testthat-pakket. Eenheidtoetsing verseker meer robuustheid van die kode, wat nou in 'n R-pakket saamgestel is. Die uitvoering van die pyplyn is ook geoutomatiseer deur die Nextflow-werkvloeibestuurstelsel te gebruik. Laastens het ek die pyplyn in 'n Singularity-houer ontplooi vir uitvoering op enige rekenaar platform, insluitend hoëprestasie-rekenaarklusters

Acknowledgements

I would like to express my sincerest gratitude to my supervisors; Prof. Gerard Tromp, Dr Elizna Maasdorp and Prof. Gian van der Spuy for their expertise and guidance in my learning process, their contributions to this work, and their mentorship.

I would like to also say a special thank you to Prof. Helena Kuivaniemi whose mentorship, support and counsel has been very impactful in this academic journey.

I acknowledge and thank Ncité Lima DaCamara, for her initial work on the Luminex Pipeline, which facilitated a smooth start for this project.

I also acknowledge the German Academic Exchange Services (DAAD) for funding my MSc studies. Additionally, I acknowledge the Centre for High Performance Computing (CHPC), South Africa, for providing computational resources to this research project.

I am grateful to my family and friends especially my fiancée, Dorcas for their love and support during this phase of my education.

Many thanks to all staff and students at the Division of Molecular Biology and Human Genetics, Stellenbosch University, I learnt a lot from all the wonderful presentations. Also, I would like to thank the Division's head, Prof. Gerhard Wazl, for creating such a stimulating scientific environment.

Finally, I would like to acknowledge God for bringing me this far and for His sustenance during my MSc studies.

Table of Contents

List of figures

List of tables

List of abbreviations

.mat – Matlab file

.rds – R data file

AIC – Akaike information criterion

API – Application programming interface

BASH – Bourne again shell

CRAN – Comprehensive R archive network

CSS – Cascading style sheet

DAG – Directed acyclic graph

DOI – Digital object identifier

ELISA – Enzyme-linked immunosorbent assay

FI – Fluorescence intensity

FMHS – Faculty of medicine and health sciences, Stellenbosch University

GUI – Graphical user interface

HDF5 – Hierarchical data format version 5

HPC – High-performance computing

HRP – Horseradish peroxidase

HTML – HyperText markup language

IDE – Integrated development environment

JSON – JavaScript object notation

LED – Light emitting diode

MIT - Massachusetts Institute of Technology

OS – Operating system

PBS – Portable batch system

PC – Personal computer

PDF – Portable document format

POSIX – Portable Operating System Interface

q-q – Quantile - Quantile

RAM – Random access memory

SATBBI – South African Tuberculosis Bioinformatics Initiative

SIF – Singularity image file

SNP – Single nucleotide polymorphism	
SSE – Sum of square errors	
URL – Uniform resource locator	
VM – Virtual machine	
WMS – Workflow management system	

CHAPTER 1: INTRODUCTION

1.1 Executive summary

Reproducibility is a key foundational component of science. The term as defined by (Gundersen, 2021) is the "*ability of independent investigators to draw the same conclusions from an experiment by following the documentation shared by the original investigators*". This ability is important in establishing the veracity of scientific conclusions which directly or indirectly impact our overall well-being, decision-making (informing policies) and our overall understanding of the world (Committee on Reproducibility and Replicability in Science, the National Academies of Science, Engineering and Medicine, 2019a). Despite the extreme importance of this tenet of science, there is a current reproducibility crisis where many scientific results or conclusions are not reproducible. For example, out of the 53 'landmark' cancer studies subjected to reproducibility test, only six studies were found to be reproducible despite the replication studies being closely undertaken with the original authors (Begley & Ellis, 2012). The scientific process and methods therefore need more rigour, transparency (openness) and scrutiny to ensure that scientific results and conclusions are verifiable and can hold up to scrutiny.

The rising concern of irreproducibility has been reported well in several disciplines. In Scientific Computing (the domain of this project), the reproducibility challenge is presented in poor consistency in achieving numerical stability across different computing environments, poor documentation practices and an inadequate number of robust tools to help improve reproducibility in certain specific domains e.g., Luminex data processing.

Multiplexed enzyme-linked immunosorbent assay (ELISA) is a high-throughput alternative to conventional ELISA used in detecting and quantifying biological analytes (eg. proteins) in biomedical diagnostics and research purposes. The Luminex platform developed by the Luminex® corporation (https://www.luminexcorp.com/) is as such a multiplexed ELISA platform. Unlike conventional ELISA, Luminex can simultaneously quantify biological analytes through a specialised technology called the Luminex xMAP technology. The quantification of analytes often results in the generation of large and complex data sets which require multiple analytical and processing steps. The inherently large variance of Luminex data also accounts for the multiple-step processing and analyses — which opens a window for irreproducibility and inconsistency because these analytical steps and processing are usually performed ad hoc.

In this work, I present the LuminexPipeline, an in-house R-based pipeline that helps to improve the reproducibility and consistency of multiplexed ELISA data processing. The LuminexPipeline is an original work done by Ncite DaCamara as part of her PhD dissertation project. My MSc project aims to further extend the utility of her initial work. First, I addressed the risk of irreproducibility and inconsistency during the multiple-step processing with Nextflow (Di Tommaso et al., 2017), an automated workflow management system. This ensured minimal human intervention which is a major source of variation and irreproducibility during Luminex data processing. Second, I containerised the pipeline execution environment with Singularity (Kurtzer, Sochat & Bauer, 2017), which helps to guarantee numerical stability across different computing platforms and aids in cross-platform execution of code. Containerisation also helps to facilitate sharing and publishing of computational environments from which research analyses were done for independent reproduction and verification. Third, I performed unit testing with the R testthat package (Wickham, 2011) to ensure the robust and accurate functioning and maintainability of the pipeline components. Lastly, I extended the utility of the pipeline by the addition of a statistical summary report.

Through the development of the LuminexPipeline, I contributed to strengthening open science by making this tool and all associated development code open source – freely available for use and further development by third-party individuals or organisations. Such open science promotes rigour, verification, and reproducibility.

1.2 Introduction of concepts

1.2.1 Reproducibility

The concept of reproducibility is not new. In the 17th century, Robert Boyle on his controversial invention of the vacuum pump wrote, *"that the person I addressed them to might, without mistake, and with as little trouble as possible, be able to repeat such unusual experiments"* (Berkowitz, 2014; Kim, Poline & Dumas, 2018; Stark, 2018). The definition of the term "reproducibility" may vary across disciplines, but its underlying principle and concept is usually the same. For example, the National Academy of Sciences of the USA defines the term reproducibility as *"obtaining consistent computational results using the same input data, computational steps, methods, code, and conditions of analysis"* (Committee on Reproducibility and Replicability in Science, the National Academies of Science, Engineering and Medicine, 2019b). Another definition of the term is: *"the ability of independent investigators to draw the same conclusions from an experiment by following the documentation shared by the original investigators"* (Gundersen, 2021). Though implemented in different domains, e.g., computational and experimental (laboratory) domains, the concept is the same — results or conclusions from an analysis or study should be able to stand independent verification or confirmation upon the provision of sufficient information. Reproducibility is also associated with other terms like replicability, repeatability, generalisability, and robustness which may have slight variations in definitions. For example, in scientific computing, The Turing Way project (of the Alan Turing Institute; https://the-turing-way.netlify.app/) defines replicability as generating similar results with the same analysis (code) using different data sets. Robustness is defined as answering a research question with the same data set but with a different analytical workflow (e.g., using R instead of Python). Generalisability is defined as answering the same research question with different data sets and different analytical workflows (The Turing Way Community, 2022).

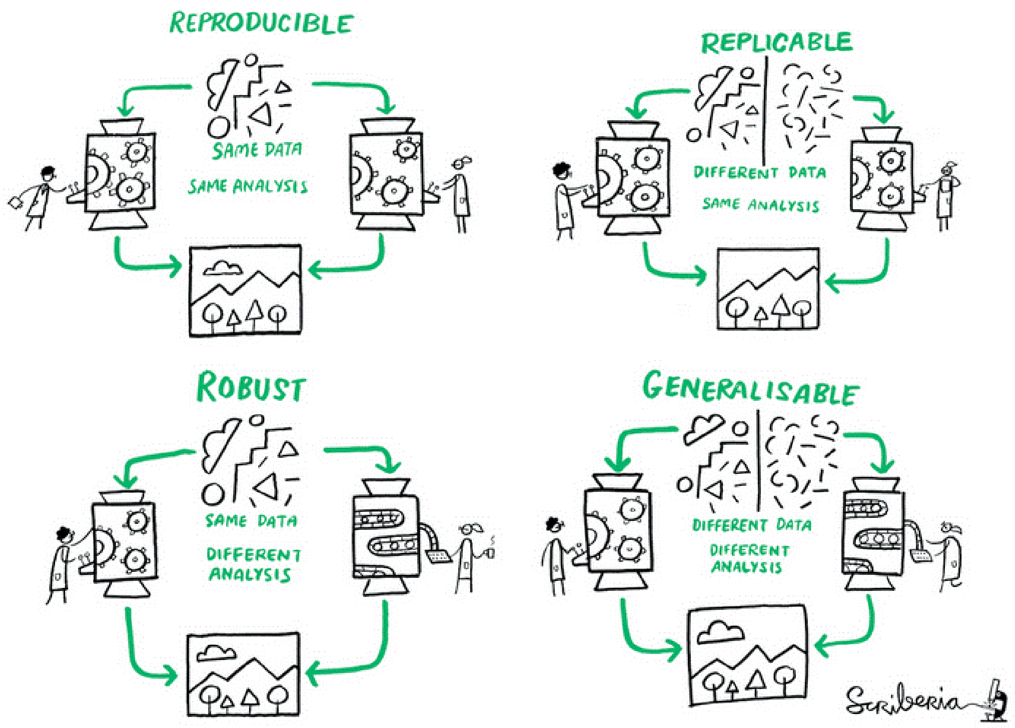


Figure 1. Definition of reproducibility and its associated terms by the Turing Way Community, Alan Turing institute. Image used under permission allowed by the CC-BY 4.0 licence. DOI: https://doi.org/10.5281/zenodo.3332807

In the absence of a standardised definition, the American Society for Cell Biology (ASCB) has tried to define the terms to encapsulate all the various definitions using the following terms: *direct replication* — reproducing previous studies using the same experimental design and conditions in the earlier study; *analytic replication* — reanalysing the same data set to confirm scientific findings; *systemic replication* — confirming scientific findings under different experimental conditions (e.g., in a different model organism); and *conceptual replication* — validating a phenomenon with different methods or experimental conditions (ASCB Task Force, 2014). Throughout this thesis, I use the term reproducibility to broadly encompass the ASCB definitions unless otherwise explicitly defined.

Table 1. summary of the ASCB's definitions for the different types of reproducibility (ASCB Task Force, 2014)

Term	Definition
Direct replication	Reproducing previous experiments with the same study design and conditions
Analytic replication	Re-analysing the same data set to confirm previously reported findings
Systemic replication	Confirming scientific findings in a different experimental condition
Conceptual replication	Verifying a phenomenon using different methods or experimental conditions

The principle of reproducibility is one of the key foundational principles of science (Committee on Reproducibility and Replicability in Science, the National Academies of Science, Engineering and Medicine, 2019a). As humanity and the natural world are directly and indirectly impacted by scientific results, it is expedient that scientific findings and conclusions are able to stand the test of scrutiny and verification. Reproducibility facilitates the process of verification and scrutiny while ensuring transparency and openness of the scientific process. This is an extremely important principle for establishing and building upon scientific knowledge (Arunika, 2016) as well as improving public trust in science. However, there has been an alarming concern recently, regarding the lack of reproducibility in many published studies (Ioannidis, 2005; Begley & Ellis, 2012; Baker, 2016) which may have grave implications for the knowledge generation process in science. It also wastes time and resources and risks losing public confidence in science. This problem, as widely documented in major scientific disciplines has been termed the "reproducibility crisis".

One of the major contributing factors to the current reproducibility crisis is poor documentation practices. Unlike Robert Boyle, many scientists today do not collect or provide adequate information about their research to be independently used to reproduce their work. Such practises may include failure to share code or research data (Huang & Gottardo, 2013), and incomplete or erroneous description of methods, e.g., failure to document what software and which software versions were used for a certain computation. Another contributing factor to the reproducibility crisis, specifically in scientific computing, is poor adherence to current best practices, for example, unit testing of code. Failure to subject code to unit testing or third-party testing may render the analysis or code to be error-prone which may directly impact its ability to be reproduced in a different computing environment (Papin et al., 2020).

Another closely related contributing factor to the reproducibility problem is the poor standardisation of experimental metadata collection. Standardising metadata collection involves the use of standard terms (e.g., ontological terms) and annotation methods to describe scientific methods and experimental conditions which facilitates reproducibility by improving documentation practices and increasing the coverage of individuals who can achieve reproducibility with such documented metadata because of the worldwide adoption of standard usage.

I briefly introduce here, two methods relevant to this project (pipelines and the investigation, study and assay meta data framework), which help to improve reproducibility. I further provide an extensive review on the subject of reproducibility in section 1 of the Literature Review chapter (Chapter 2).

1.2.1.1 Workflow management system (Pipelines)

Many computationally intensive analyses involve a considerable number of processing steps which are often modularised (each step is self-contained). Performing these analyses interactively or ad hoc raises concerns about reproducibility because, the use of these methods (interactive or ad hoc analyses) often results in little or no record of the processing and analytical steps being implemented (Franceschi et al., 2014). Workflow management systems (pipelines) can address this challenge by automating data processing and analytical steps — which is achieved by streamlining or chaining together all analytical processes in an automated manner. Thus, the output of an analytical process or step can be programmed to serve as the input to another analytical process or step without requiring any human intervention or interactivity.

The use of an automated workflow management system in an analysis, therefore, helps to significantly improve reproducibility by eliminating human intervention (a major source of variation contributing to irreproducibility) during data processing and analysis; providing a better and reproducible alternative to ad hoc analytical approaches; keeping provenance information by allowing analytical results to be traced back to the analytical pipeline that generated the results (Committee on Reproducibility and Replicability in Science, the National Academies of Science, Engineering and Medicine, 2019a). It also contributes to improving the efficiency and speed of scientific knowledge generation. For example, an analytical framework (pipeline) can be constructed for a specific analysis in which different data inputs are subjected to the same set of computational instructions — this allows for newly generated data sets to be easily processed and analysed in an expedited manner. Additionally, most workflow management systems natively support the use of containers (one of the current best practices to achieve reproducibility); the scalability of heavy computations on the cloud or high-performance computing cluster and re-entrancy (resumption of computation after an unexpected break), which improves computational efficiency and speed in data processing and analyses. Some examples of the popular workflow management systems used in Bioinformatics are Nextflow (Di Tommaso et al., 2017), Snakemake (Koster & Rahmann, 2012), Galaxy (Afgan et al., 2018), and common workflow language (Amstutz et al., 2016)). In this project, I used the Nextflow workflow management system to automate and improve the reproducibility of multiplex ELISA data processing and analysis which involves multiple steps.

1.2.1.2 The investigation, study, and assay (ISA) metadata framework

A lot of interventions have been established to address the reproducibility crisis, among which is the investigation, study and assay (ISA) metadata framework (Rocca-Serra et al., 2012) for collecting experimental metadata for annotating research data to improve reproducibility in scientific experiments. The ISA framework collects ontological metadata in three tab-delimited or JSON file formats (Johnson et al., 2021). The first file is the investigation file, which is at the top of the ISA framework's hierarchy. It collects contextual metadata of the project. This includes a brief description of the investigation, investigation-related publications, contact details of investigators as well as submission and publication dates.

The study file (second of the hierarchy) collects ontological metadata of the study subjects such as sample type, sample characteristics, factors (an independent variable controlled by the investigator to affect a biological system resulting in a measurable assay; https://isa-specs.readthedocs.io/en/latest/isamodel.html) and the factor type. The type of study design used in the experiment is also recorded in the study file.

The third file i.e., the assay file, records the measurements and types of measurement used in generating experimental results as well as the protocols used to process and analyse the samples. It also captures information about the type of instrument used for the measurement.

All of this rich metadata information is important to attain reproducibility in published research, especially in this new era where research funders are requiring researchers to publicly make research data available to facilitate scientific discovery and re-use of data for new research questions (Committee on Reproducibility and Replicability in Science, the National Academies of Science, Engineering and Medicine, 2019a). The ISA metadata framework is thus important to provide rich and standardised contextual information about a research project to enable independent researchers to comprehend a scientific project, especially a published data set, and to facilitate the reproduction of experimental findings or conclusions (Rocca-Serra et al., 2010; González-Beltrán et al., 2015). The ISA software suite (Rocca-Serra et al., 2010) is one of the extensible utilities that facilitate the collection of ontological metadata.

Table 2. Summary of the ISA hierarchical structure and the type of metadata information collected

ISA file structure	**Content**
Investigation file	Brief description of investigation, publications related to the investigation, contact details of investigators etc
Study file	Description of sample type, sample characteristics etc
Assay file	Description of variable measurements and instruments used for the measurements

1.2.1.3 The FAIR guiding principles

Another intervention to help circumvent the current reproducibility challenges is adherence to the findable, accessible, interoperable and reusable (FAIR) guiding principles (Wilkinson et al., 2016) for sharing research metadata and data. The FAIR guiding principle requires that (meta)data are (i) Findable with a unique identifier (e.g., DOI) and (meta)data are indexed in searchable sources (search engines); (ii) Accessible by their identifiers and are open and freely available for use but, when necessary, the appropriate authentications be applied; (iii) Interoperable using standardised vocabularies and broadly applicable languages; and (iv) Reusable with an accurate description of relevant attributes and provenance information. This principle enhances research metadata and data [(meta)data] to be automatically accessed and readily utilised by machines to facilitate reusability. Thus, this complements other efforts under-way to improve the reusability (reproducibility) of research (meta)data — as they are increasing in numbers due to the increasing demand to make research (meta)data publicly available.,.

1.2.2 ELISA and Multiplexed ELISA

ELISA is a technique used to quantify and identify biological analytes e.g., cytokines, peptides, and hormones based on an antigen-antibody interaction. It was initially described by (Yalow & Berson, 1960) as an antibody-mediated detection technique using a radioactive signal. Because of health concerns with radioactivity, alternative approaches were later sought (Shah & Maghsoudlou, 2016). In 1971, two research groups (Engvall & Perlmann, 1971; Van Weemen & Schuurs, 1971) independently reported a detailed procedure for detecting analytes using an enzyme-labelled antibody, a technique now known as an ELISA. This technique is now a widely used method and the gold standard for quantifying and detecting biological analytes because of its specificity and sensitivity as well as flexibility in its implementation chiefly because it has the capability to bind to a wide variety of organic and inorganic compounds. It also has an excellent specificity for the material (antigen) that binds to an antibody and it has the ability to detect and quantify the strength of antigen-antibody binding (Wild, 2013). Thus, this technique has been successfully applied in clinical diagnostics and biomedical research. There are many variations to the technique but generally, it starts with a coating step where sample antigens or capture antibodies (to bind with sample antigens) are immobilised on a polystyrene microwell plate, either directly (adsorption) or indirectly (Rattle et al., 2013). An antibody linked with an enzyme, usually, horseradish peroxidase (HRP) is then introduced to bind with the sample antigens (Cox et al., 2019). The antigen-antibody interaction is detected by a signalling system e.g., the enzyme-labelled activity in the presence of analytes. The intensity of the signal (e.g., coloured by-product) infers the amount of analyte present in the reaction. The various types of ELISA are categorised based on the method of coating (antigen or antibody immobilisation to plate) and by the method of detection (Figure 2).

1.2.2.1 Direct ELISA

In direct ELISA (Figure 2), the coating step is done directly by immobilising the sample antigen to the microwell plate. An antibody linked with an enzyme binds to the immobilised antigen. This ELISA type is faster because it requires fewer steps which also makes it less prone to errors. On the other hand, direct ELISA may have a relatively high background noise because of its immobilisation step which allows all proteins (including target antigen) in the sample to bind to the plate. Assay sensitivity is also relatively low because of the absence of a secondary antibody to amplify the signal. It is also not flexible because every immobilised protein requires a specific conjugated antibody for binding. This type of ELISA is best suited for studying antigen immune response (Lin, 2015a).

1.2.2.2 Indirect ELISA

Indirect ELISA (Figure 2) has the same coating step as direct ELISA. However, the method of detection involves the binding of an unlabelled primary antibody to the immobilised antigen. An enzyme-labelled secondary antibody, directed at the primary antibody, enables signal amplification since more than one labelled secondary antibody can be directed to the primary antibody to improve overall sensitivity. This type of ELISA offers a lot of flexibility because one labelled secondary antibody can bind to different primary antibodies. A disadvantage is the increased possibility of cross-reactivity between the secondary antibody and the immobilised antigen which could increase the background noise. Additionally, indirect ELISA can be time-consuming because of additional incubation steps, see (Lin, 2015b).

1.2.2.3 Sandwich ELISA

The sandwich ELISA (Figure 2) is the most widely used type of ELISA. In its setup, the coating step is done by immobilising a capture antibody to the ELISA plate. The sample antigen is allowed to bind to the capture antibody. A detection antibody (either labelled or unlabelled) then binds to the sample antigen to form a sandwich. Because both antibodies (capture and detection) bind to different epitopes on the same antigen, the assay is highly specific. This ELISA type is very sensitive and delivers great flexibility since both direct and indirect ELISA can be implemented. A disadvantage, however, is that in the absence of a standardised kit, optimisation to avoid cross-reactivity between capture and detection antibodies can be difficult. Sandwich

ELISA is useful in studying complex samples such as tissue lysates where the target analyte is part of an impure sample.

1.2.2.4 Competitive ELISA

Competitive ELISA (Figure 2) is used to detect and measure the concentrations of minute molecules, e.g., drugs. It is set up with a small concentration of antibodies, sample antigens and inhibitor antigens or tracers (labelled sample antigens). In this assay setup, there is competition between sample antigens and inhibitor antigens to bind with the limited antibodies present in the reaction. The proportion of inhibitor antigens or tracers that bind to the antibodies is indirectly proportional to the concentration of the sample antigens present in the reaction, see (Wild, 2013)

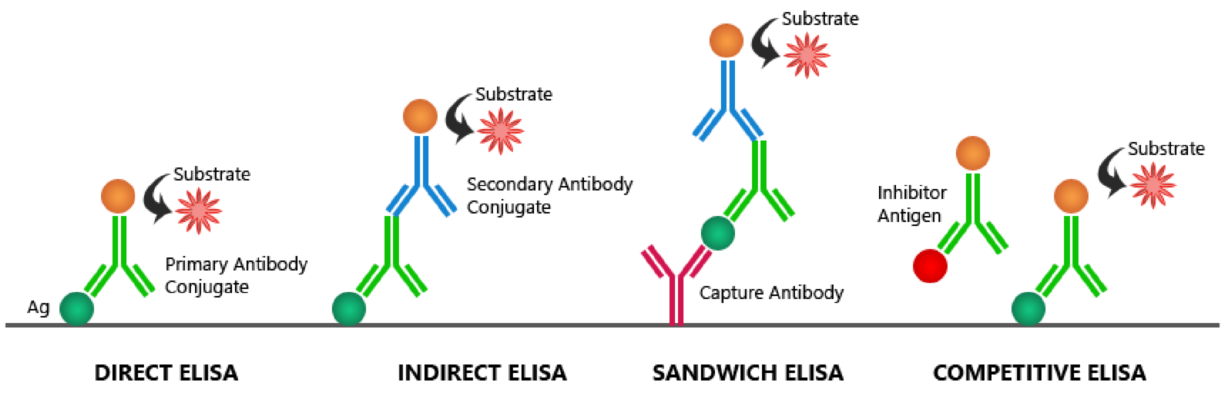


Figure 2. An illustration of the different types of ELISA with highlights on the differences in the types of antigen immobilisation. Image source: Bosterbio

1.2.2.5 Multiplexed ELISA (Luminex xMAP Technology)

A high throughput, cost-effective, sample and labour-efficient alternative to the standard ELISA techniques described above is the multiplexed ELISA powered by the *Luminex® corporation*'s multi-analyte profiling (xMAP) technology — where "x" in xMAP represents the analytes to be investigated (Figure 3). This technique allows several analytes to be concurrently identified and quantified from the same sample by use of magnetic or polystyrene microspheres (beads) coated with carboxyl groups. The carboxyl groups on the beads facilitate the covalent conjugation of analytes to the beads (Carl et al., 2019). The beads are also uniquely dyed with different proportions of two or three fluorescent dyes to form spectral addresses, allowing the distinct identification of analytes (Dunbar, 2006). In a typical multiplexed ELISA (Luminex) assay, the configuration of sandwich ELISA is employed. That is, distinct capture antibodies are conjugated to the beads whereas sample antigens of specific affinity to the capture antibodies bind to the bead-conjugated-capture antibodies. A bound-complex, is formed after a labelled detection antibody specific to the sample antigen binds to the antigen. The detection antibody is usually labelled with phycoerythrin or streptavidin (the reporter molecule) to serve as an enzyme substrate (signalling molecule) that fluoresces during analyte interrogation (Dunbar & Hoffmeyer, 2013; Bio-Rad, n.d.)

Assay interrogation is done in a flow cytometer where analytes are identified and quantified simultaneously. In the flow cell, a red laser or light-emitting diode (LED) excites the fluorescent dye in the beads to uniquely identify the analyte. At the same time, a green laser or LED excites the reporter molecule. The intensity of fluorescence is used to infer the quantity or concentration of analytes since the observed fluorescence intensity is directly proportional to the concentration of the analyte.

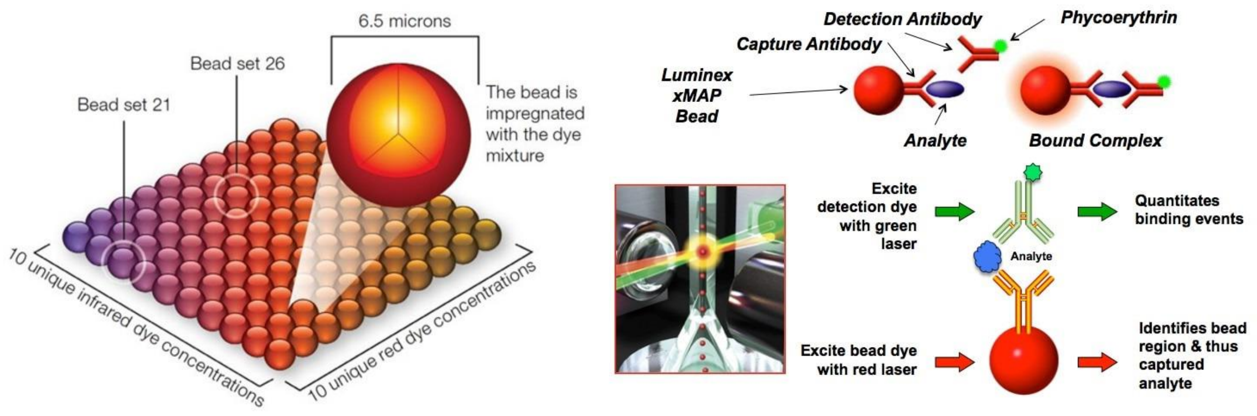

6.5 microns
Bead set 26
Bead set 21
The bead is impregnated with the dye mixture
10 unique infrared dye concentrations
10 unique red dye concentrations
Detection Antibody
Phycoerythrin
Capture Antibody
Luminex xMAP Bead
Analyte
Bound Complex
Excite detection dye with green laser
Quantitates binding events
Analyte
Excite bead dye with red laser
Identifies bead region & thus captured analyte

Figure 3. Distinctly dyed beads are used as spectral addresses to uniquely identify analytes (left). Analyte identification and quantification in a flow cytometer, where red and green lasers are used to identify and quantify analytes (right). Image source: Thermo Fisher Scientific Inc.

Multiplex ELISA has been successfully applied in quantifying multiple biomarkers in clinical drug development studies of multifactorial diseases in which several analytes need to be measured to perform a comprehensive analysis of the biological molecules contributing to the disease pathogenesis (Tighe et al., 2015). It has also been broadly used in pathogen detection and typing; protein-protein interaction studies; gene expression studies; and for genotyping single nucleotide polymorphisms (SNP) (Dunbar, 2006; Dunbar & Li, 2010).

1.3 The Luminex Pipeline

The Luminex Pipeline (Figure 4) is an R-based utility initially developed by Ncité Lima DaCamara as part of her PhD work. This utility is used for processing and analysing multiplexed (Luminex) ELISA assay data in a robust and reproducible manner. The functionalities of the pipeline include importing raw Luminex files (.*txt* files), data cleaning and tidying, recording metadata (e.g., values beyond detectable limits), imputation of missing values, and standardising analyte names with an analyte reference list to ensure consistency. The Luminex Pipeline performs these tasks while recording all the processing and analytical steps for record-keeping and reproducibility purposes.

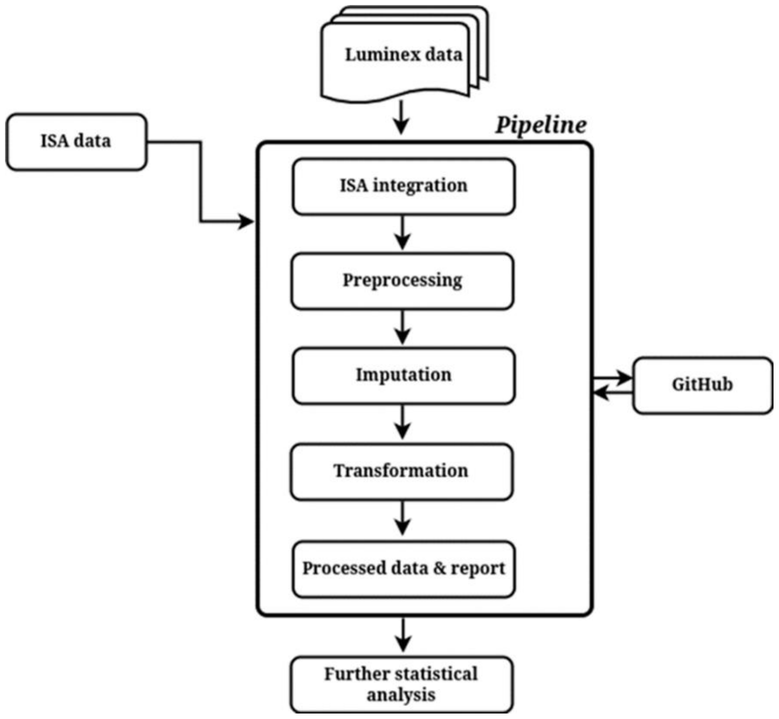


Figure 4. Flow chart of the Luminex Pipeline components

1.4 Problem Statement

Luminex data processing and analyses can be very challenging because of the complexity of the data due to multiplexing. Thus, the data can be highly dimensional and large. Luminex data are also characterised by an inherently high variance, missing values (e.g., concentrations beyond the detectable range of the Luminex instrument), duplicates and non-normality in the distribution of analyte concentrations. These challenges contribute to the complications of data processing and analyses leading to non-standardised and inconsistent data processing and analytical approaches. Such ad hoc approaches limit the reproducibility of the data processing and analytical steps (methods)

The Luminex pipeline was developed to address some of the aforementioned challenges in attaining reproducible methods and generating consistent and reproducible results. However, despite its current robust and reproducible implementation, there is still room for extension of its utility by implementing other robust and reproducible approaches. For example, it would be advisable to automate the pipeline's execution with a workflow management system to circumvent human intervention and other sources of variation during the pipeline execution. Another area in need of extension is the ability to achieve numerical stability (consistent results) across different computing environments — one of the major challenges of analytical reproducibility. A containerised pipeline can help to address this challenge. Additionally, the robustness of the pipeline can further be extended to improve the way it handles errors (generating useful error messages) and to ensure accurate functionality which can be achieved with the help of unit testing. Furthermore, the pipeline components can be extended with a statistical summary report component to improve its utility. Lastly, additional utilities (R Shiny application) can be developed to help facilitate exploratory data analysis.

1.5 Aim

To extend the overall utility, robustness, consistency, and reproducibility of the Luminex Pipeline for analysing multiplex ELISA data.

1.6 Objectives

I. Write new R functions to generate a statistical summary report and compile them together with existing pipeline functions into an R package.
II. Test and document the pipeline components to ensure the general robustness and accurate functionality of the pipeline.
III. Automate pipeline execution with a workflow management system to improve reproducibility and ease of use.
IV. Develop a containerised runtime environment for pipeline execution to improve reproducibility.

V. Develop an R Shiny application for ease of use in exploratory analysis of the pipeline's output data.

VI. Extend pipeline components with a statistical summary report to extend the overall utility of the pipeline.

CHAPTER 2: LITERATURE REVIEW

This chapter is divided into two sections. The first section gives a review of some of the major factors that contribute to the current reproducibility crisis and the efforts underway to alleviate them. The second section gives a review of Luminex data processing and analysis.

2.1 Section 1 – Reproducibility

2.1.1 Introduction

One of the fundamental principles of science is the independent verification and "reproducibility" of scientific methods and results. A broad understanding of the reproducibility term is the ability to repeat published experiments or studies and be able to generate identical findings or draw identical conclusions as the original study. Thus, to achieve reproducibility, there needs to be transparency, openness and rigour in scientific methods and processes. This may be achieved by simply providing all the necessary information needed to reproduce a published work or by using technical approaches like setting stringent significance thresholds for novel discoveries to reproduced (Ioannidis, 2014). Reproducibility is extremely important for ensuring the credibility of scientific results and conclusions and for attaining societal trust in science (Committee on Reproducibility and Replicability in Science, the National Academies of Science, Engineering and Medicine, 2019c). Nevertheless, attaining reproducibility is no guarantee of the accuracy of research findings or conclusions (Ioannidis, 2014). For example, a study which is well documented (to achieve reproducibility) but utilises a wrong experimental design may have its biased or incorrect conclusions easily reproduced by an independent researcher. Thus, the main aim of reproducibility is to foster transparency and openness in the scientific process. Such transparency will allow for the identification and correction of mistakes (like the wrong experimental design example) during independent verification or peer review processes. Reproducibility is therefore important to significantly improve the credibility of research findings and conclusions, and to discourage scientific misconduct.

In addition, despite its overarching importance in science, it is important to note that the concept of reproducibility may be subject to some philosophical and practical limitations in certain contexts, for example, sources of variation in experimental subjects that are difficult to control (e.g., study participant's deliberate failure to disclose medical history) may render an experiment difficult to reproduce (Casadevall & Fang, 2010). Nevertheless, reproducibility remains an indispensable component of doing good science.

2.1.2 Current reproducibility crisis

Reproducibility is increasingly becoming a subject of great concern. For example, a recent survey conducted by the international journal Nature (Baker, 2016) revealed that of the 1,576 respondents surveyed, over 70% reported having failed in trying to reproduce the work of other scientists while a staggering 50% had tried and failed to reproduce their own work. The survey also reported that more than half of the respondents (52%) agreed that there is a significant reproducibility crisis in science. Several other studies have reported challenges in reproducing the work of others (Begley & Ellis, 2012; Arunika, 2016; Kim, Poline & Dumas, 2018; Mullard, 2021) resulting in a heightened uptake of interest in the subject. Here, I review some of the leading contributors to the current reproducibility crisis.

2.1.2.1 Transparency

One of the major barriers to attaining reproducibility in science is poor record-keeping and documentation practices and the unwillingness of researchers to share research data, code, etc. Research findings can easily and independently be reproduced only when sufficient and detailed information about all research procedures is documented and made easily available and accessible. Thus, it represents transparent research. However, this is often not the practice, contributing to the current reproducibility crisis in a major way (Nuzzo, 2015). One of the reasons for poor transparency is the fear of researchers incriminating themselves when they provide enough details for reproducing their work.

2.1.2.2 Cognitive bias

Cognitive bias is one of the major reasons why research findings or conclusions may not be reproduced. Cognitive bias is the tendency of an individual’s subjective, subconscious, or personal beliefs to influence their decision-making and judgement processes (Tversky & Kahneman, 1974; Cooper & Meterko, 2019). Several types of cognitive bias have been identified. *Confirmation bias* is the unconscious tendency to interpret new evidence in a way that supports one's pre-existing beliefs or hypotheses (Munafò et al., 2017). This bias influences data collection and interpretation. *Selection bias* results from poor sampling techniques leading to sample data which are not representative of the population (Tripepi et al., 2010). *Cluster illusion* results in the perception of patterns in data that are non-existent in reality. The *bandwagon effec*t is a type of bias with the inclination to support a viewpoint without giving it enough thought to maintain group cohesion (Landucci & Lamperti, 2021). This type of bias results in the acceptance of ideas based on their popularity but which may not be necessarily accurate. Lastly, *reporting bias* is when research subjects withhold important information from researchers based on their subconscious drive (Munafò et al., 2017). Reporting bias is also true for researchers selectively reporting positive results.

2.1.2.3 Low statistical power

Another major concern for not attaining reproducibility has been attributed to low statistical power (Bishop, 2019). This is because there are very small odds of detecting minimal effects in underpowered studies (small sample size) even when an effect exists. Furthermore, simulation studies, see (Poldrack, 2022) have shown that when effects are detected in underpowered studies, the effect sizes reported are most likely to be grossly overestimated, a phenomenon termed the *winner's curse*. This phenomenon is a major problem for replication studies which usually fail to detect earlier reported effects because the effect size of the original study was overestimated due to low statistical power. In discovery studies, this problem can be a major reason for not attaining reproducible findings in subsequent replication studies.

2.1.2.4 Inappropriate or poor use of statistical tools

P-values have widely been used to test hypotheses and estimate the likelihood of an observed result being attributed to chance. However, p-values on their own, cannot be used to draw scientific conclusions as their use is intended to be in conjunction with background knowledge — the plausibility of the hypothesis (Nuzzo, 2014). In this light, p-values have been used inappropriately by many scientists. According to (Chawla, 2017), p-values should be interpreted as "suggestive evidence" (especially for values between 0.05 and 0.005) and not necessarily as a basis for established knowledge, i.e., p-values are not definitive. Calculations have shown that a statistically significant result with a p-value of 0.01 has an 11% chance of being a false positive result. Also, there is a 29% chance of reporting a false positive result with a p-value of 0.05, see (Nuzzo, 2014). Furthermore, p-values do not indicate the magnitude or size of the effects. In addition to these constraints, p-values are generally over-relied on, misused and misinterpreted (Ioannidis, 2018) contributing to the current reproducibility crisis.

2.1.2.5 P-hacking

Another problem, commonly known as p-value hacking, is the process whereby several statistical analyses are performed to attain significant results. This problem is embedded in the saying of Ronald Coase that "*if you torture the data long enough, it will confess to anything*" – i.e., scientists try every means possible to achieve significant results. P-hacking can be done in any of the following poor research practices (Poldrack, 2022): (i) excluding participants to attain significant p-values; (ii) analysing several variables but reporting only those that gave significant p-values; and (iii) concurrent analysis with data collection but the attainment of a significant p-value prompts the end of data collection. These questionable research practices may significantly hamper the reproducibility of a scientific finding. Additionally, they have been shown to increase the rate of false-positive findings (Simmons, Nelson & Simonsohn, 2011). P-hacking may be one of the outcomes of publication bias where significant results are more likely to be published.

2.1.2.6 HARK-ing

Another reason for the poor reproducibility in science is the common practice of HARK-ing ("hypotheses after results are known"). Harking is defined as *"presenting a post hoc hypothesis (i.e., one based on or informed by one's results) in one's research report as if it were, in fact, an a priori hypotheses"* (Kerr, 1998). Here, researchers make new hypotheses after seeing the trends in the data and change their initial hypotheses. The problem has been described by (Poldrack, 2022) as moving a goalpost wherever the ball goes. The consequence of this is the difficulty of invalidating incorrect ideas since the goalpost can always be adjusted and manoeuvred to match the data.

2.1.2.7 Publication bias and undue preference for novelty

The current scientific culture has been biased against the null hypothesis (Bishop, 2019). It unduly favours statistically significant results or novel findings, rewarding them with a greater chance of publication. On the other hand, negative results (statistically non-significant) or replication studies (non-novel studies) are less likely to be published. This poor culture consequently leads to wasted time and resources and, to a certain degree, hinders scientific advancement because when only significant findings are published, nobody learns about the supposedly "failed" experiments and as such, efforts, time, and resources are wasted repeating such experiments. Also, the preference for novelty discourages replication studies which would enhance the self-correcting nature of science and boosts confidence in scientific knowledge, for example, when a replication study confirms an earlier finding.

2.1.2.8 Inconsistent definition of terms

Another problem is the lack of a consistent definition of the terms for reproducibility (Gundersen, 2021). For example, in biology, reproducibility usually means a different laboratory attaining similar experimental results from scratch, while in computational sciences it often means the provision of sufficient details to repeat computations (Stark, 2018). Reproducibility has other associated terms which may be of varied definitions depending on the discipline. In computational sciences, reproducibility; replicability; robustness and generalisability are all terms associated with "reproducibility" but have different meanings. On top of this, these definitions are not standardised, for example, the definitions of reproducibility and replicability are interchanged by the Association for Computing Machinery (ACM) and *Claerbout and Karrenbach* – who first proposed definitions for the terms (Plesser, 2018) (Table 3). Non-standardised definitions may lead to confusion and may further derail progress to alleviate the current reproducibility challenges.

Table 3. Swapped term definitions by ACM and Claerbout & Karrenbach. Information used under permission allowed by the CC-BY 4.0 license. Source: https://github.com/alan-turing-institute/the-turing-way

Term	**Claerbout & Karrenbach**	**ACM**
Reproducible	"Authors provide all the necessary data and the computer codes to run the analysis again, re-creating the results"	"(Different team, different experimental setup.) The measurement can be obtained with stated precision by a different team, a different measuring system, in a different location on multiple trials. For computational experiments, this means that an independent group can obtain the same result using artifacts which they develop completely independently"
Replicable	"A study that arrives at the same scientific findings as another study, collecting new data (possibly with different methods) and completing new	"Different team, same experimental setup.) The measurement can be obtained with stated precision by a different team using the same measurement procedure, the same measuring system, under the same operating conditions, in the same or a different

analyses"	location on multiple trials. For computational experiments, this means that an independent group can obtain the same result using the author's artifacts"

2.1.3 Reproducibility in scientific computing

The challenge of reproducibility in scientific computing usually has to do with the unavailability of code. In a survey involving 400 Artificial Intelligence papers, only 6% of the papers made the code used in the paper available (Hutson, 2018). However, the challenge of reproducibility in computational science goes beyond just making available the code and data used in a computation. In their case study, (Kim, Poline & Dumas, 2018) outlined the challenges they encountered in trying to reproduce a published bioinformatics paper with available code and data. The authors defined reproducibility and its associated terminology used in their work (Figure 1) as:

Reproducible – generating identical results with the same code and underlying data.

Robust – using the same underlying data but with a different code to arrive at an identical result, e.g., using a Python script instead of an original R script from which analysis was carried out.

Replicable – different data but the same code to attain similar results.

Generalisable – use of different codes with different underlying data.

The authors reported difficulties with hardware compatibility in their attempt to reproduce the results using the same underlying data and code as in the original work. The original authors had used a MATLAB library in their code which was dependent on the architecture of the operating system (OS). Running the analysis on a different OS, the new authors reported their hurdles in trying to recompile the library to be able to reproduce the original work. Additionally, the new authors experimented with the robustness of the code (different code, same data) by recoding the MATLAB code into a Python package — which, in contrast to MATLAB, is a free and open-source alternative. They reported challenges in attaining robustness. These challenges included their inability to change the original file format from a MATLAB *(.mat)* file format to a hierarchical data format version 5 (HDF5) format in Python. Furthermore, they reported errors and challenges due to varying code parameters and arguments. For example, the default arguments of the functions used for clustering in the original MATLAB code differed significantly from those of Python. This highlights the importance of having a thorough understanding of the behaviour of implemented functions or code in safeguarding the validity of results when recoding in a different programming environment or version. Although most of the hurdles encountered in their attempt to attain robustness can be easily overcome by using virtual container environments and by strict adherence to recommended guidelines, the authors acknowledged that developing container environments may be a difficult technical task for some researchers and may require additional training. Furthermore, strict adherence to recommended guidelines may take some time to be widely incorporated into the routine work of researchers.

One essential element necessary for achieving reproducible methods and results in scientific computing is the development of robust tools. A "robust" tool in this context is defined as *"software that works for people other than the original author and on machines other than its creator's"* (Taschuk & Wilson, 2017). In other words, a utility that can be easily installed on different computers and whose integration with other tools is possible. Robust tools are important to improve reproducible research and accelerate scientific research. In this regard, (Taschuk & Wilson, 2017) have proposed ten rules or best practices for developing robust research tools such as software to improve computational reproducibility. These rules are summarised below.

2.1.3.1 Code documentation

Good documentation practice is needed to help new users easily navigate and use the program. Thus, the utility should be properly documented. Documentation is usually done in a README file. The authors recommended some necessary minimal guidelines for good documentation practices, which include explaining the purpose of the program, listing all dependencies, providing installation instructions, describing input and output files, demonstrating usage with few examples and providing licensing information or information on how users can credit the work.

2.1.3.2 Version control

Software development versioning helps to keep track of all developmental changes and facilitates bug-fixes in isolation from the main development repository. This allows for working features (fixed bugs) to be merged with the main repository when needed. Versioning also allows the developer to revert to a previous version of the program in the development phase and enhances collaboration.

2.1.3.3 Seamless control of operations

The program should make exploratory analysis easier by incorporating parameters that directly influence the results in the code. If a parameter is required, a reasonable default value should be provided. Furthermore, the program should be able to check, upon start-up, the correct input values or files and should be able to generate useful error messages when incorrect inputs are encountered. This makes the program easier to use.

2.1.3.4 Include versions for every release

As the program evolves with time, each release (i.e., update of the program) should be allocated a version stamp in incremental order. This identifier makes it easier for future retrieval of a specific release. The most common versioning semantic for open-source software is the "MAJOR.MINOR[.PATCH]" semantic, e.g., version 0.5.3. The "major" version number is updated as significant changes are made to the program while incrementing the "minor" version number is reflective of minor changes to improve the program, e.g., adding new features. Finally, the version of the program should be easily accessible, e.g., with the –version argument on the command line.

2.1.3.5 Reusing other programs

Sometimes, it is necessary and useful to not "reinvent the wheel" by reusing external code, functions, or programs in one's software or program. However, the downside to this may be the introduction of complex dependencies which, often are hard to deal with. The authors propose that the reuse of supplementary programs should be based on a true need before they are incorporated into one's program. Additionally, the developer should ensure that the auxiliary program is robust.

2.1.3.6 Build-tools and package managers for installation

The program should rely on build-tools (e.g., Make, Maven etc) or package managers (e.g., pip for Python) for installation since these sets of utilities can automatically determine and install the program's dependencies. The developer, therefore, must document a machine-readable file of all the program's dependencies.

2.1.3.7 No special or root privileges for installation

Since most scientific software are of themselves not malicious, installing them should not require any special privileges (except for unusual circumstances). Furthermore, as much as possible, the installation of the program should be tested before deployment. This can be achieved by utilising virtualisation containers (e.g., Singularity) or simply asking colleagues to install the program on their computers.

2.1.3.8 Avoid hard-coding file paths

Hard-coding file paths or parameters into a program makes it difficult to execute the program on a different computing environment since hard-coded paths may not exist in the new computing environment. The program should, therefore, allow users to specify the input and output file locations as parameters or arguments to the program.

2.1.3.9 Include test data

The program should include small test datasets with which users can run or explore the program's functionality after installation. The test data can also be used to demonstrate the program's usage, for example, in the documentation. Aside from the inclusion of test data, the program should be subjected to unit testing.

2.1.3.10 Identical inputs generate identical outputs

A particular version of the program should produce the same results given a specific set of parameters and data. To further improve reproducibility, the program should print to standard output or a log file, the software version and the parameters used in an execution.

2.1.4 Overcoming the reproducibility challenges

Several measures are being taken by all stakeholders of science i.e., funders, publishers, research institutions and scientists to help address the reproducibility crisis. Below, I highlight some of the efforts being undertaken and recommendations to help improve reproducibility in science.

2.1.4.1 Improve sharing, record-keeping and documentation practices

Researchers should document and report all undertaken procedures with accompanying metadata including instruments used, measurements taken, and variables measured of the research project. That is, for example, giving a clear description of all analytical procedures and reasons for including or ignoring certain data; reporting on statistical power; and reporting on how uncertainties were dealt with (Committee on Reproducibility and Replicability in Science, the National Academies of Science, Engineering and Medicine, 2019a).

As a major component for achieving reproducibility, good documentation or record-keeping practice is also important to facilitate the self-correcting nature of science. For example, a recent study reported that eight out of the ten pioneer genome sequences of the Orangutan species initially published were mistakenly assigned to the wrong species (Kreier, 2022). The absence of good record-keeping and sharing practices may have left this mistake undiscovered, consequently impacting subsequent studies that would have relied on this “inaccurate” data. This highlights the importance of adopting reproducible methods such as good documentation, record keeping, and sharing to achieve rigour and verification of scientific results, as well as to facilitate the self-corrective nature of science.

2.1.4.2 Training and Education

Researchers need to be trained on the need for reproducible research and should be trained in utilising current best practices and tools available for improving reproducibility in their research. This measure has gained some traction as several institutions are incorporating reproducibility concepts in their curriculum (Committee on Reproducibility and Replicability in Science, the National Academies of Science, Engineering and Medicine, 2019a). For example, postgraduate training at the South African Tuberculosis Bioinformatics Initiative (SATBBI), Stellenbosch University emphasises the development of skills for attaining computational reproducibility as demonstrated by this reproducibility-themed MSc thesis.

2.1.4.3 Pre-registration

One of the undertaken measures to help to prevent questionable research practices and thus improve reproducibility is pre-registration — the provision of detailed experimental design and analytical steps of a research project before embarking on the research project (data collection or analyses). With this approach, researchers first publish their protocols before starting on the study and are then guaranteed a second publication of their final results regardless of the outcome of results (whether positive or negative), on the condition that they adhere to all the pre-registered protocols. Pre-registering a research project can therefore help to minimise some questionable research practices (e.g., p-hacking, harking) and publication bias (Bishop, 2019).

2.4.1.4 Different analytical approaches

One of the analytical approaches to help reveal and minimise certain cognitive biases during analytical procedures is crowd-sourced analysis — several teams answering the same research questions by analysing the same data. For example, this approach was applied in a study where 29 different teams were asked to analyse the same data sets to answer the research question whether dark-skin-toned footballers are more likely to receive a red card in a football match. The results were that 69% of the teams reported significant effects while 31% found no statistically significant effects (Silberzahn et al., 2018). This sharp variation in the results may be a result of several biases that go into data analysis. Unfortunately, in most cases, only one team performs the analysis which increases the probability of having biased results in the publication. Although crowdsourced analysis may not be feasible to implement for every study, it can help to reveal biases in analyses and can also help to establish consensus in complex data analyses.

Another method to help circumvent bias and achieve analytical reproducibility is performing blind data analysis. Here, a researcher or analyst is presented with deliberately altered data sets (e.g., swapping experimental groups). The analyst is blinded to all the alterations made to the data and proceeds with the analysis. The unaltered data is subsequently run through the initial analysis. This approach helps to achieve objective analytical procedures since analysts are less likely to halt the analysis upon the arrival of results that favour their predefined or subconscious thoughts (Nuzzo, 2015).

2.1.4.5 Proper use of statistical tools

The current misuse of statistical tools, especially, p-values for hypothesis testing has prompted the American Statistical Association (ASA) to provide clear guidelines for the interpretation and use of p-values. These guidelines include the definition and the extent of use of p-values: i) P-values can reveal the degree to which the data contradict a certain statistical model; ii) p-values do not measure effect sizes or how relevant a result is; iii) scientific and business conclusions should not solely rely on whether p-values pass or fail to reach a specified threshold; and iv) p-values do not measure the probability that data were randomly generated by chance alone nor do they measure the probability of a hypothesis being true, see (Committee on Reproducibility and Replicability in Science, the National Academies of Science, Engineering and Medicine, 2019a).

Some proposals have suggested lowering the significance threshold in biomedical science from 0.05 to 0.005 (Chawla, 2017; Ioannidis, 2018). This proposal aims to minimise the rate of false positive results in published papers. Another aim is that the stringent threshold may lead researchers to better design their experiments to achieve statistically significant results (Ioannidis, 2018). Although this method has its limitations such as the likelihood of increasing false negative results (failure to detect an effect when there is indeed one), it has been applied with success with an even more stringent threshold ($5x\ 10^{-8}$) in genome-wide association studies (Chawla, 2017). Other proposals have suggested entirely abandoning p-values and instead reporting the magnitude of effects and confidence intervals, see (Ioannidis, 2018), whereas some proposals have suggested the use of more sophisticated approaches like Bayesian statistics (Goodman, 2001).

2.1.4.6 Improve transparency

Many research funders and publishers are beginning to demand that research data and analytical code be made publicly available to foster transparency, openness and verifiable research findings. These are important initiatives to help attain reproducibility in science and allow re-analysis of existing data with the same and novel statistical tools.

2.2 Section 2 – Immunoassay calibration and analytical tools

2.2.1 Immunoassay setup

Immunoassays are used by investigators to measure the concentration of analytes in a sample. A typical immunoassay will have standard samples with known concentrations, test samples with unknown concentrations and blank samples (usually a buffer solution to measure background noise) as part of the set up. The standards are serial dilutions of varying concentrations of analytes from which a standard curve for estimating the concentration of test samples is constructed. To ensure accurate measurement of analyte absorbance, the blanks are used to determine the baseline analyte absorbance. Theoretically, the blanks will have zero absorbance, but it is usually not the case in practice because of background noise or contamination (Sheehan, He & Smith, 2013). Correction for background noise is usually done by subtracting the baseline absorbance from the detected absorbance in the sample.

2.2.2 Dose-response curves

The relationship between the known standard concentrations (dose) and the observed fluorescence intensity (response) is used to construct a standard or calibration curve, also known as the dose-response curve to calculate or extrapolate estimates of unknown concentrations in test specimens. Ideally, the standard curve should resemble or be close to the *true curve* which is the curve that accurately reflects the dose (concentration) – response (fluorescence intensity) relationship without any errors. In other words, the true curve is the resultant curve after measuring the response of an infinite number of concentrations with an infinite number of replicates (Dunn & Wild, 2013). This is, however, not attainable in practice where sample replicates and concentrations are limited, and the standard curve is drawn from a given number of responses. A mathematical function, known as the curve model, is used to approximate the true curve from the assay data by fitting, i.e., adjusting the curve model's parameters to obtain the optimal curve closest to the true curve. During curve fitting (calibration), several fitting errors can be encountered which may have a direct impact on the quality of the true curve approximation, known as the quality of fit (Gottschalk & Dunn, 2005a). It is, thus, important to identify and address the sources of fitting errors when calibrating the standard curve. Two main sources of curve fitting errors have been identified: "*pure error*" and "*lack-of-fit error*". The pure error arises from the inherent random variation of the data. Increasing the number of standard replicates used to derive the standard curve can be used to address the challenge of pure error. The lack-of-fit error is derived from using a curve model that does not reflect the shape of the data or does not correctly approximate the true curve, for example, using a straight line as the curve model to fit assay data that have a sigmoidal shape (the typical shape of most immunoassays). In this type of error, increasing the number of replicates will not reduce the lack-of-fit error (Gottschalk & Dunn, 2005a).

2.2.3 The curve function or model

According to (Gottschalk & Dunn, 2005a), an ideal curve model must have three qualities. First, the curve model should be able to correctly describe the observed dose-response relationship from the assay data, i.e., correctly approximate the true curve. Second, the curve model must be able to average out random noise and variation to generate estimated concentrations with the least minimal influence of pure errors. Last, the curve model must not only accurately estimate a concentration at fitted data points (standard data points) but should also be able to accurately predict estimated concentrations between the fitted data points (test sample data points). It is, therefore, imperative to choose the right curve model to improve the coverage and the accuracy of the estimated concentrations.

Many curve fitting functions have been used with varying degrees of success in attaining the three qualities of a good curve model described above. For example, the linear approach (straight-line) model cannot approximate the true curve of immunoassays which is typical of a sigmoidal shape (lack-of-fit error challenge). The logit-log curve model can fit the sigmoidal-shaped immunoassay data, but its logit transformation intensifies the noise and variability in the standards. Additionally, the logit-log model is better suited for symmetric data and does not properly approximate the true curve of asymmetric data, see (Gottschalk & Dunn, 2005b). Another curve model that has been used to fit immunoassay data is the mass action model, but this model has the challenge of being unable to reduce noise in the data because of its many parameters, see (Raab, 1983; Gottschalk & Dunn, 2005a). Cubic splines (Guardabasso, Rodbard & Munson, 1987) have also been used to fit immunoassay data. The problem, however, with this curve fitting model is that its fitting passes through each data point failing to average out the variability and noise in the data. One of the widely used curve models is the four-parameter logistic (4pl) function, this model has been shown to perform very well with symmetric data but quite poorly with asymmetric data (Cumberland et al., 2015). The five-parameter logistic (5pl) curve model is an extension of the 4pl model with a fifth parameter to accommodate for asymmetry. This enables the 5pl model to be better suited to fit asymmetrical data. The model has also a similar performance to the 4pl model when used on symmetrical data (Cumberland et al., 2015). The 5pl curve model is given by the equation:

$$y = d + \frac{(a+b)}{[1+(x/c)^b]^g}$$

Where a and d are the upper and lower asymptotic ends; b is a parameter for the slope of the curve (rate of change of response with increasing dose); c is a parameter for the point of curve inflection, and g is a parameter controlling for asymmetry. When g is 1, the 5pl curve model is equivalent to the 4pl (Dunn & Wild, 2013).

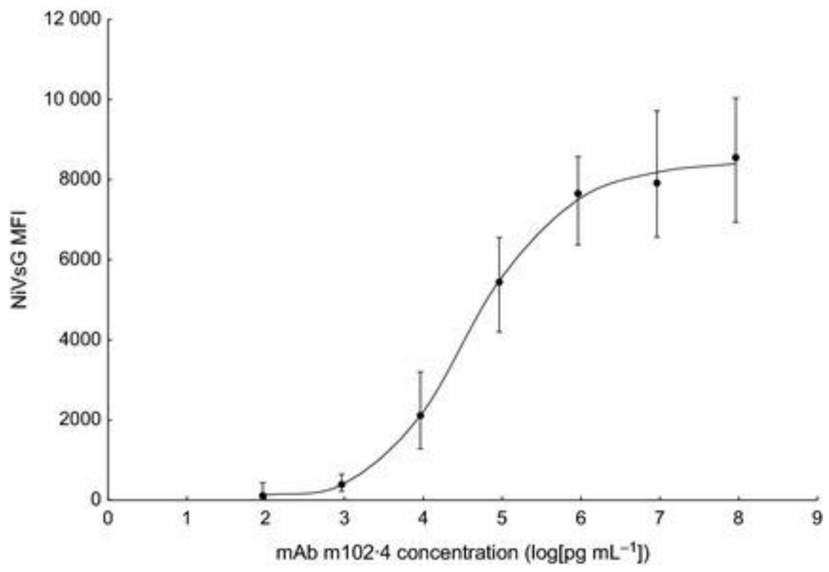


Figure 5. Illustration of a typical standard curve of a Luminex assay. Notably, the unequal variance in the response (y-axis) as the dose (x-axis) increases. This assay appears to have an increasing variation in response (y-axis) with increasing doses (x-axis). It is important to note that some assays may have non-uniform or random variations in the response (heteroscedasticity). Image used under permission allowed by the CC BY 3.0 license. Source: (Baker et al., 2014)

2.2.3.1 Fitting the curve

Upon deciding on the appropriate curve model, the next step is curve fitting. Curve fitting is done by adjusting the free parameters of the curve model until the best fitting curve out of many different possible curves is achieved (the curve closest to the true curve). In other words, the best-fitted curve is the "maximum likelihood estimate of the true curve" and it is selected by calculating the curve with the least "weighted sum of squared errors (SSE)" (Gottschalk & Dunn, 2005a). The least SSE is one of the main methods in statistics used for assessing modelling errors. Errors, also known as residuals, are the differences between the observed responses and the predicted responses of the curve at each dose or concentration (Figure 6). Computing the least SSE is done by squaring the errors or residuals (to avoid negative and positive values from cancelling each other out) and then summing up the squared errors or residuals. This computation is done for all the possible curves that can be derived from the curve model. By the statistical regression principle, the curve whose parameters yielded the least sum of squared errors from the many derived curves is the curve that best models the data or is closest to the true curve.

2.2.3.2 Weighting

The use of only the unweighted SSE in approximating the "maximum likelihood estimate of the true curve" is often inadequate and can impact the accuracy of the concentration estimates. This is because the calculation of the SSE does not account for the heteroscedasticity (non-uniform response variance also known as random errors) in the data (Figure 5). It is common for the variance of the responses at the high and low ends of the curve to differ by three to four folds (Gottschalk & Dunn, 2005a). For example, as shown in Figure 6, fitting the standard curve for data without weighting the SSE will bias the curve towards the upper responses. That is, although the computed or predicted responses at the two (highlighted) doses in the illustration are both 5% lower than the observed response, the margin between their residual squares is too wide (250,000 against 25) which will result in the lower responses having very little contributing effects on the fitted curve.

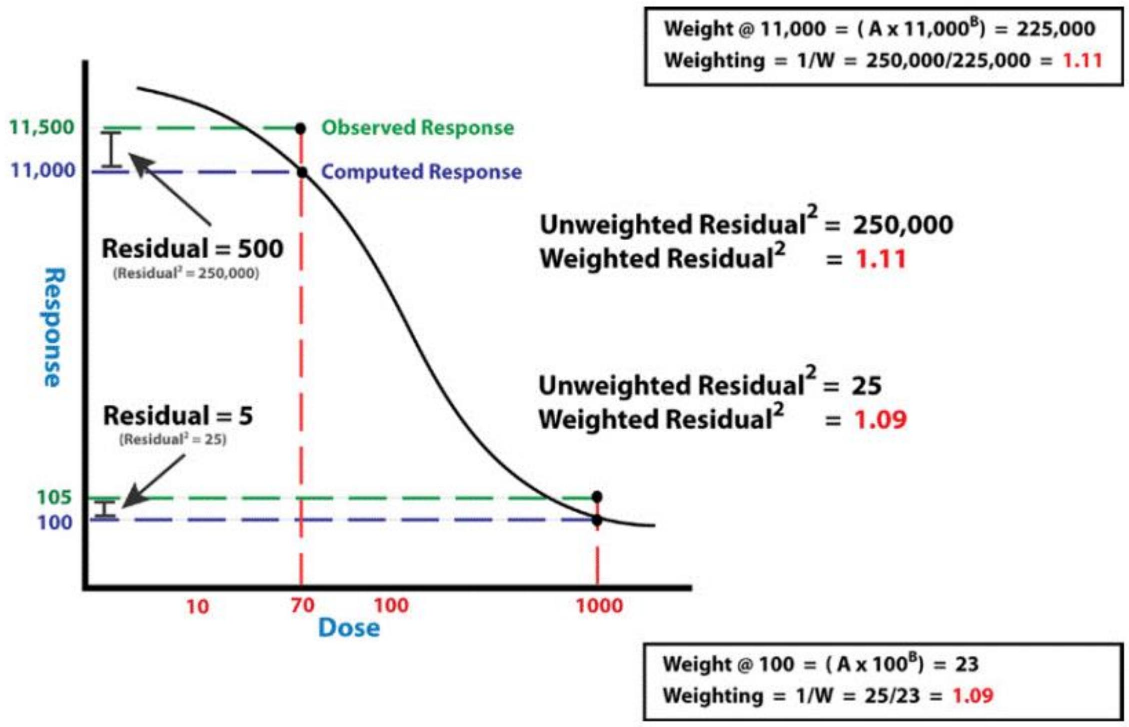

Weight @ 11,000 = (A x 11,000^B) = 225,000
Weighting = 1/W = 250,000/225,000 = 1.11
11,500
11,000
Observed Response
Computed Response
Residual = 500
(Residual² = 250,000)
Response
Unweighted Residual² = 250,000
Weighted Residual² = 1.11
Unweighted Residual² = 25
Weighted Residual² = 1.09
Residual = 5
(Residual² = 25)
105
100
10
70
100
1000
Dose
Weight @ 100 = (A x 100^B) = 23
Weighting = 1/W = 25/23 = 1.09

Figure 6. Equal percentage difference between the observed and computed response at doses 70 and 1000 but a very wide difference in the residual squares. The image is not drawn to scale. Reproduced with permission from (Dunn & Wild, 2013)

The US Food and Drugs Administration and the European pharmacopoeia, thus, require all immunoassays to be weighted, due to the significant impact weighting has on the accuracy of results (Brendan Bioanalytics, n.d.).

The response variance or random errors can be a result of the signal noise or errors in the instrument's detector or the non-linear kinetics of antibody-antigen binding across different concentrations or doses (Dunn & Wild, 2013).

Given the heteroscedastic nature of the responses in Luminex assays, (Dunn & Wild, 2013) argue that applying a transformation, e.g., Log of response, ${}^{1}/_{response}$, ${}^{1}/_{response^2}$ cannot make the variance in the response constant, i.e., homoscedastic, thereby requiring the SSE to be weighted with the inverse variance of the response at the specific concentrations – according to regression theory, see (Gottschalk & Dunn, 2005a). A power function of the responses is classically used to estimate the response variance of standards by the equation:

$$Variance = A(Response)^B$$

Where A is a function of the response magnitude and its average noise level and B ranges from 1.2 to 2 (Gottschalk & Dunn, 2005a).

The equation for calculating the *weighted* SSE of the standard curve is given:

SSE $=\sum_{i=1}^{N} w_i[y_i - \hat{y}_i]^2$

Where $w_i = \frac{1}{response\ variance\ of\ data\ point}$, y_i is the observed response of the standard and $\hat{y}_i$ is the predicted response of the curve model. By weighting the SSE, the curve with the least weighted sum of square errors will best represent the true curve or standard curve.

2.2.4 Luminex data processing utilities

Several tools have been developed to analyse multiplexed ELISA data. In this section, I review two of the utilities used in calibrating and analysing Luminex data.

2.2.4.1 drLumi R package

drLumi (Sanz et al., 2017) is an R (R Core Team, 2021) based package with a general public licence (GPL >=2) used for management, calibration and quality assessment of Luminex bead assay data. The utility can also be generalised to accommodate other multiplexed ELISA assays. drLumi was adapted from other R-based non-linear curve fitting packages eg. ncal (Fong et al., 2013) and it provides the utility for enhancing the precision and accuracy of assay data analysis.

The package achieves this by: (i) computation of quality control metrics; (ii) an automatic extraction of assay data generated from the Luminex xPONENT software; (iii) provision of different methods for handling background noise; and (iv) use of different limits of quantitation (LOQ) techniques. The drLumi package can automatically import Luminex xPONENT software generated Luminex data. However, assay data generated from other software may need some manual data preprocessing to be imported by the package. This, however, may be a limitation since data preprocessing may require some level of programming expertise on the part of the investigator. The package also provides the utility for computing metrics for quality control assessments, e.g., quality of fit of standard curve. This is important for the accurate estimation of unknown sample concentrations since, the use of an inappropriate curve model or a poorly fitted model can be one of the main sources of errors in immunoassay data analysis (Dudley et al., 1985). Additionally, the package makes provision for the use of different methods for handling background noise. Ideally, the response for a solution (e.g., buffer solution) without an analyte should be zero or close to zero, however, this is usually not the case for most immunoassays because of background noise. Classically, background noise is subtracted from the standard response before fitting the standard curve. However, (Sanz et al., 2017) argue that in some situations, this approach may result in a poorly fitted curve. They, therefore, provide, in the drLumi package, different approaches to account for background noise to optimise standard curve calibration. Furthermore, the package can utilise different techniques to estimate the limits of quantitation (LOQ). LOQ are the boundaries, i.e., the minimum and maximum concentration thresholds on the standard curve from which unknown concentrations can be accurately estimated or interpolated. Since estimation or interpolation is reliably done from around the linear section of the standard curve, this functionality is essential in assessing the accuracy of estimated concentrations (Figure 7).

After assay data has been imported, drLumi can fit the standard curve using three different curve models — the 5-parameter logistic curve model (5pl), the 4pl curve model and the exponential growth model to estimate unknown concentrations. The drLumi utility fits these curve models on a log base 10 transformed data (both dose and response) to minimise the heteroscedasticity and large variability in Luminex data. However, according to (Dunn & Wild, 2013), log transformation alone may not be an effective method for stabilising the variance, i.e., making the variance uniform or close to uniform. Thus, they recommend using the weighted response variance at each concentration to fit an unbiased standard curve. Based on this argument, the sole use of log transformation by the drLumi package may inefficiently address the inherent heteroscedasticity in the assay data and may result in bias in its curve fitting as well its estimation of unknown concentrations.

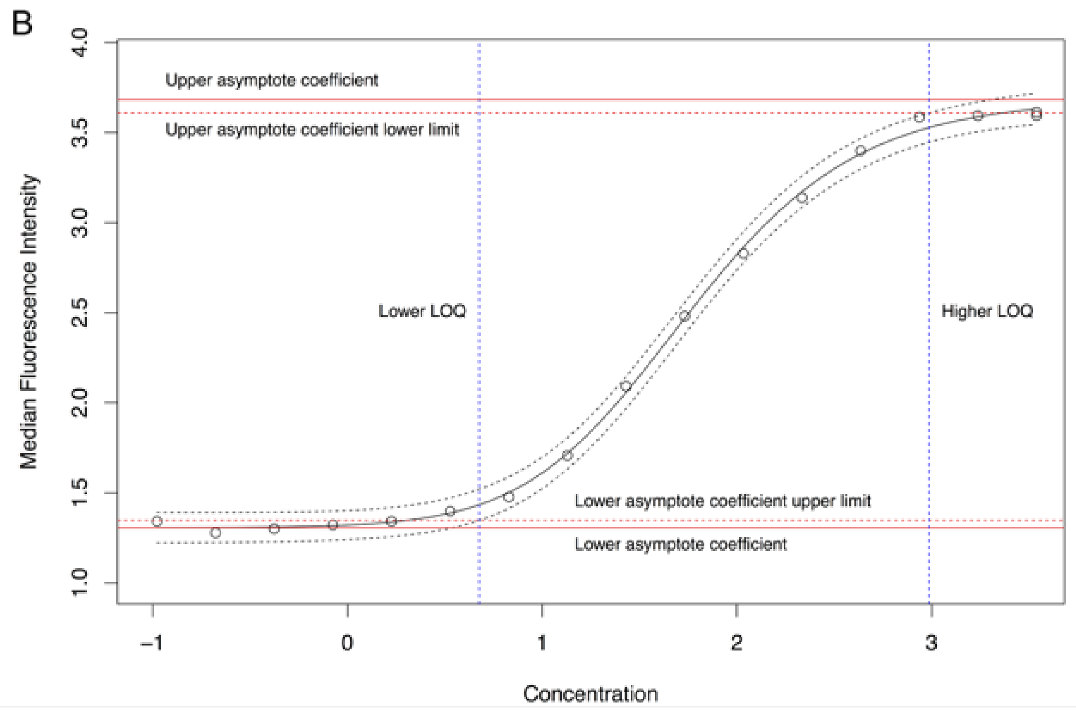


Figure 7. Illustration of limits of quantitation using the interval method with the drLumi package. The limits of quantitation for this method are marked by the blue vertical lines. Image used under permission allowed by the CC BY 4.0 license. Source: (Sanz et al., 2017)

The drLumi implements four different ways of handling background noise: the *subtract*, *ignore*, *include* and *constraint* methods. The *subtract* method is what is traditionally used but it is sometimes inappropriate because usually the background noise approximates the lower asymptote of the standard curve. This means that subtracting the background noise from the observed response will translate to removing the lower asymptote of the curve which will result in an improperly fitted curve. The *ignore* method fits the standard curve without considering the background noise. The *include* method uses the geometric means of the blank controls (representative of the background noise) together with the standards to derive the standard curve. This method can be problematic by generating a heavily biased curve when blank controls are contaminated. However, it can be useful when there is accurate assaying of blanks with enough standard dilutions. The final alternative is the *constraint* method which constrains the lower asymptote of the curve model to the background noise.

Quality assessment or quality control methods implemented by the drLumi R package involve computing the goodness of fit test (Neil test p-value) and as well reporting the Akaike information criterion (AIC) for the fitted model. Additionally, the package provides the utility to plot the standard curve with confidence intervals – a good standard curve will have a narrow confidence interval with data points lying directly on the curve or close to the fitted standard curve while unreliable curves may have large confidence intervals and most data points further away from the predicted or fitted curve. Additionally, drLumi provides the utility to flag out data points as outliers after the user classifies them as such. This utility allows flagged data points or outliers to be excluded during curve fitting. The drLumi utility also provides functions to visualise the quality of fit of the curve model with a plot of residual errors and a quantile-quantile plot.

Additional functionality of the drLumi R package is its ability to estimate the limits of quantitation from which unknown sample concentrations can be reliably interpolated. The package provides three different methods, each dependent on a specific characteristic of the curve. These methods are the coefficients of variation method, derivative method and interval method. The interval method limits quantitation thresholds to the lower and upper values that are statistically significantly different from the two asymptotes. The derivative method restricts the boundary of quantitation of concentrations to the approximately linear portion of the standard curve. Finally, the coefficient of variation method uses a predefined cut-off value, determined by the user, usually set at 20%, to estimate the LOQ from the variability of the fitted concentrations. The authors of the package suggest that the optimal use of these methods is dependent on the specific assay and analyte being investigated. However, they propose that the ideal method will permit quantitating the most samples while keeping the background noise of the plate below the LOQ (Sanz et al., 2017).

Overall, the drLumi package is a great tool for performing quality assessment and calibration of standard curves with varying options to optimise and improve calibration performance. The open-source license of the package makes it freely available for use and modification of code as well as contributes to open science. However, its flexibility (varying parameters) may render it somewhat suited for expert use only but not for investigators with non-linear modelling experience — which may demand a steep learning curve for non-statisticians. Additionally, the package may demand users to be well-versed in R programming, especially for users using calibration software other than the Luminex xPONENT software, which has support for automatic data import.

Nota Bene: As of April 2020, drLumi has been archived on the comprehensive R archive network (CRAN) site (*https://cran.r-project.org/web/packages/drLumi/index.html*) due to lack of continued development and support. Installing the package may therefore be challenging since it is not available for automatic installation in R.

2.2.4.2 LabKey server tool for Luminex

The LabKey server (Nelson et al., 2011) is a web-based utility for general research data management in Java for which an extended support for Luminex assay analysis was developed (Eckels et al., 2013). The server stores research data in the PostgreSQL relational database. Developed to facilitate large collaborative studies, the LabKey server provides a unified system or platform for managing research data. At its centre, is a central data repository allowing clinicians, statisticians, lab researchers, administrators etc. involved in a study, to interact, query or update the research data. The unified system also allows users or collaborators to track specimen records and request specimen information such as clinical data on the web-based utility. Being one of the few open-source utilities with a graphical user interface (GUI) for managing research data, its GUI allows users to flag or add scientifically-relevant attributes to data columns — a useful utility for quality control purposes. The GUI also helps to visualise missing values and relationships between tables, e.g., clinical data and metadata, and to filter metadata. Additionally, the LabKey server integrates clinical data with complex experimental data and specimen data through SQL queries and its GUI.

Being a multi-purpose utility, the LabKey server has customisable assay templates for Luminex assays, microarray, enzyme-linked immunosorbent spot (ELISpot) assay, and ELISA assay data management and analysis (Nelson et al., 2011).

Luminex assay analysis on the LabKey server is done by importing raw Luminex assay data and integrating the data with associated metadata. Only the Bio-Plex Manager software output data and Excel data formats are supported for import. The assay analysis on the LabKey server tool for Luminex can be done automatically with an extensible and customisable R script that computes dose-response curves from standard dilutions, estimates unknown sample concentrations, detects outliers, computes quality control metrics for curve fitting procedures and flags questionable experimental data for exclusion in further analysis (Eckels et al., 2013). During curve fitting, the LabKey server tool for Luminex provides an option to subtract or ignore background noise, as opposed to the drLumi utility which makes provision for four options to manage background noise. The LabKey server tool for Luminex also provides an option to log transform fluorescence intensities (FI) before calibrating the curve to reduce the noise and variability in the data. Two curve models, the 5pl and 4pl curve models are utilised to generate dose-response curves, from which unknown sample concentrations are estimated or interpolated. Unlike the drLumi package, the LabKey server tool for Luminex can apply weights ($weight = \frac{1}{FI^{1.8}}$) to the squared errors to lower the heteroscedasticity of the response signals, improving the overall curve quality and the estimated concentrations by accounting for the unequal variance of the response signal.

Overall, the integration of metadata in Luminex assay analysis can significantly improve reproducibility when the processed data are shared. The customisable R script implemented in the analysis and the web API improves the flexibility of the utility. Additionally, the open-source licence (Apache 2.0) of the source code promotes open science. Also, the GUI implementation of the LabKey server tool for Luminex is an important utility for ease of use, especially for researchers with little programming experience. Finally, the centralised data repository of the LabKey server, on which the Luminex tool runs, can facilitate secure data sharing between collaborators.

Some of the limitations of the LabKey server for Luminex assay analysis include the limited compatibility for different assay file formats — the utility only recognises data outputs from the Bio-Plex Manager software and the Excel data format. Also, extensive expertise in R may be required to customise assay data analysis. Finally, the utility does not support the design of Luminex plate layout, thus, it cannot be used to run a Luminex instrument (Eckels et al., 2013).

CHAPTER 3: METHODS

3.1 The R LuminexPipeline package

An R package is the encapsulation of bundles of code, tests, documentation and data which functions as a shareable unit (Wickham, 2015). Thus, one of the most efficient ways of sharing R code and, to some extent data is by bundling them into an R package — ready to be shared and distributed (Merow, 2019). Most R packages are hosted on the comprehensive R archive network (CRAN) (https://cloud.r-project.org/) server for distribution. Outside the context of code sharing, R packages can also be a great way to organise and structure R code.

Package development is an essential toolkit that can help to improve analytical reproducibility. This is because R packages utilise *functions* as a core component to automate repeated tasks which can be useful in reproducing analytical results. That is, given the same R version and function with the same arguments, analytical results, including graphs and tables etc., can be easily reproduced provided that the computational environment does not change.

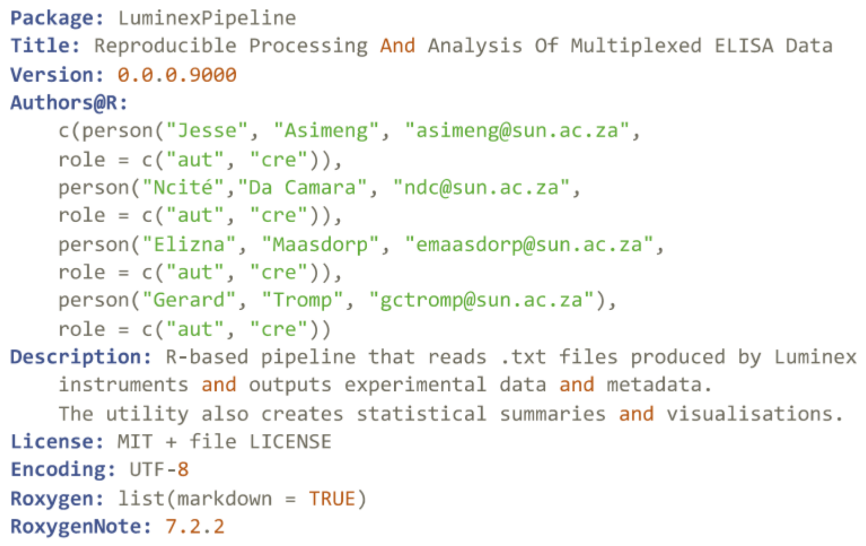
All pipeline components were written as R functions and bundled together into an R package (LuminexPipeline package) to achieve the aforementioned benefits including efficient code distribution as well as good code organisation. The R devtools v2.4.5 package (Wickham, Hester, et al., 2022) was used to compile together all the functions of the pipeline components into an R package. I utilised the best package development practices outlined in the *R packages* book (Wickham, 2015) in the package development procedures. I intend to submit this package to the comprehensive R archive network (CRAN) for broader use by the scientific community.

Here, I describe briefly, a workflow used to develop the LuminexPipeline package.

1. I started by loading the package development utility, devtools (Wickham, Hester, et al., 2022) in the R v4.2.1 environment (R Core Team, 2021) on a Linux server. I then set up the rudiments of the new package with the `usethis::create_package()` function from the usethis v2.1.6 package (Wickham, Bryan, et al., 2022), one of the meta packages of the devtools package. This function automatically sets up the minimum required files and file system to develop an R package i.e., an *R/* directory (to house all R code and functions), a man directory (which houses the package manual and all documentation), a DESCRIPTION

and a NAMESPACE file which collects information of package metadata and dependencies respectively (Merow, 2019).

2. I edited the DESCRIPTION file with the necessary metadata for the package, i.e., the package name, title, version, authors and description fields etc, (Figure 8). I used the license helper `usethis::use_mit_license()` function to automatically edit the License field of the DESCRIPTION file with an MIT licence. This function also makes a copy of the full license in a markdown (*.md*) file. MIT license is an open-source license that permits users to freely use and modify the code in the package. Full permissions allowed by the MIT license are shown in Figure 15.



```
Package: LuminexPipeline
Title: Reproducible Processing And Analysis Of Multiplexed ELISA Data
Version: 0.0.0.9000
Authors@R:
    c(person("Jesse", "Asimeng", "asimeng@sun.ac.za",
    role = c("aut", "cre")),
    person("Ncité","Da Camara", "ndc@sun.ac.za",
    role = c("aut", "cre")),
    person("Elizna", "Maasdorp", "emaasdorp@sun.ac.za",
    role = c("aut", "cre")),
    person("Gerard", "Tromp", "gctromp@sun.ac.za"),
    role = c("aut", "cre"))
Description: R-based pipeline that reads .txt files produced by Luminex
    instruments and outputs experimental data and metadata.
    The utility also creates statistical summaries and visualisations.
License: MIT + file LICENSE
Encoding: UTF-8
Roxygen: list(markdown = TRUE)
RoxygenNote: 7.2.2
```

Figure 8. Package metadata displayed in a DESCRIPTION file

3. The NAMESPACE file was automatically updated with the `usethis::use_package()` helper function to import the namespace of all dependent packages I used in the LuminexPipeline package.
4. I wrote and organised R functions with the `usethis::use_r()` helper function. This function automatically creates an Rscript named with its input argument and stores the script in the *R/* directory of the package.
5. I intermittently used the `devtools::load_all()` function to update the package when new changes were made. The function simulates the building, installation and attachment of the development package for experimentation. For example, the `devtools::load_all()` function makes available a newly created function for interactive use or experimenting without defining that function in the global environment.

6. I intermittently used the `devtools::check()` function to ensure the complete functionality of the package as well as detect possible package development problems for rectification. The `devtools::check()` function gives a large output of information with a summary of the number of errors, warnings and notes. Intermittent checks help to easily identify problems early and debug them as this may otherwise lead to difficulty in debugging incremental problems. The ideal aim of executing the `devtools::check()` function was to have zero errors and warnings. The `R CMD check` command on the command line is an alternative command to the `devtools::check()` function.
7. A directory named *inst/ext* was created to host sample data sets which were used as examples in the package documentation as well as for package testing. This directory was also used to host the R markdown (Allaire et al., 2022) file that will be used to generate the statistical summary report.
8. Once the package was complete and fully functional, I installed the package with the `devtools::install()` function and generated the source package from the *build* menu in Rstudio. The source package is a compressed *.tar.gz* file that can be easily distributed and installed with the R utils `utils::install.packages()` function or with `R CMD INSTALL` command on the Linux command line.

```
library(devtools)

#setup the package
usethis::create_package("path/to/new/package/directory")

#helper function to write R functions/code
usethis::use_r("name_of_function")

#load changes for interactive experimenting
devtools::load_all()

#update NAMESPACE FILE imported function/dependencies
usethis::use_package("name_of_package")

#assess package is in good functioning order
devtools::check()

#automatically edit license in description file
usethis::use_gpl3_license()

#install source package
utils::install.packages("path_to_source_file", repos=NULL, type="source")
```

Figure 9. An example of a simplified workflow that was used to develop the R LuminexPipeline package.

3.1.1 Package documentation

I documented the LuminexPipeline package with the roxygen2 v7.2.1 package (Wickham, Danenberg, et al., 2022), one of the meta packages for devtools (Wickham, Hester, et al., 2022). I wrote Roxygen notes above every exported function. Exported functions are functions that are made available to the end user. Documenting the exported functions thus assists users in easily navigating and using the package. Roxygen notes are documentation notes that start with a "`#'`". They are written above functions that need documentation such as exported functions. My use of Roxygen notes involved inserting an Roxygen skeleton from the *Code* menu in Rstudio. The Roxygen skeleton provides fields, e.g., function description, parameters, and details from which I populated the documentation. The helper `devtools::document()` function was used to document Roxygen notes. It triggers roxygen2 (Wickham, Danenberg, et al., 2022) to automatically convert the Roxygen notes into a "*.Rd"* documentation file in the *man/* directory of the package.

3.1.2 Unit testing

Unit testing is one of the essential elements in package development to achieve robust and accurate code significantly impacting the efficiency, robustness, reproducibility and maintainability of the program or utility. The purpose of unit testing is to ensure that each unit of a program, the smallest testable component of the program executes exactly as purposed. In my use case, I refer to a unit as an R code or function written for use in the LuminexPipeline package. Usually, when functions are written, they are tested interactively in the console to ascertain the accuracy of their functionality. However, the problem is that these tests are not documented or automated. Thus, if the code or function needs to be refactored, it becomes difficult to determine whether the new changes alter the behaviour or functionality of the code or function (Wickham, 2015). This is where unit testing becomes useful in ensuring the accurate and reliable functionality of the code. Other benefits of unit testing include having fewer bugs and a robust code. This is due to the nature of unit testing, which requires some level of hostility. For example, exposing the functions and code to unexpected conditions (e.g., inputs) intended to break the function or code. The pre-exposure to possible errors helps to identify bugs early and to improve code and robustness. Additionally, unit testing forces the researcher and programmer to write modular code since it is much easier to write tests for modules or units of the program (Wickham, 2015).

I performed unit testing in the LuminexPipeline package by utilising the R testthat v3.1.4 package (Wickham, 2011), one of the meta-packages of the devtools package (Wickham, Hester, et al., 2022).

I initialised the system for unit testing by using the helper `usethis::use_testthat()` function to automatically set up a *tests/testthat/* directory where all test files live in the package. Test files are files in which automated tests are written. They are, by convention, required to be prefixed with a "test" name. I used the helper `usethis::use_test()` function to create new test files. This function automatically prefixes the names of test files to have the "test" prefix.

After setting up the appropriate file system for testing, the `testthat::testhat()` function was used to test for the accurate functionality and behaviour of package functions. The `testthat::testthat()` function has a structure starting with an input text description of what is being tested and a code section for the test code (Figure 10). Within the code section, several expectation functions (from the testthat package) were used to test for the expected behaviour of functions. Among the expectation functions I used were: `testthat::expect_error()`, to test for instances where error messages were expected; `testthat::expect_equal()`, where functions were expected to generate an output or behaviour equal to a specific value or behaviour; `testthat::expect_match()`, where specified outputs of functions were expected to match certain characters and `testthat::expect_warning()`, for instances where functions were expected to throw a warning message.

```
#the testthat function structure

test_that("description of test", {

  code section with expectations of behaviour.

})

#example of a unit test

test_that("multiplication works", {

  expect_equal(2 * 2, 4)

})

#test a single test file
devtools::test_file("path/to/test/file")

#test the whole package
devtools::test()
```

Figure 10. An example of a code snippet for the usage of the R testthat function. The function has two major sections, the test description section and the code section where code expectations are documented. A demonstration of how to execute a test file is illustrated here.

Tests were run with the `devtools::test_file()` function for a single test file (the argument to the function is the test file's path) and `devtools::test()` to test all the test files of the package.

I tested for the following functionalities:

- The behaviour with unexpected input, i.e., expectation of warning and error messages
- The presence of output files written to specified directories.
- The dimensions of output data frame or tibbles.
- The expected column names in the output data frame or tibble.
- The class of output data.
- The class of output variables.
- The case of variable names (UPPER, lower etc).
- The presence or absence of white spaces in column names.
- The presence of string characters in numeric-type variables and vice versa.

3.1.2.1 Package test files

I included raw Luminex text files as test data in the LuminexPipeline package as part of the recommended best practices (Taschuk & Wilson, 2017) to aid in package documentation and unit testing. The test data were de-identified sample Luminex data generated from the Stellenbosch University Immunology Research Group (SUN-IRG). I trimmed the original test data, extracting only a few rows while preserving the original structure of the data, to reduce the size of the package. The trimmed test data sets were placed in a directory named *tests/test_data* in the LuminexPipeline package.

3.1.3 Quality control

I borrowed and incorporated curve fitting functions from the drLumi R package (Sanz et al., 2017) into the LuminexPipeline package. The major difference between these two packages is that while drLumi's major utility is for quality control i.e., curve fitting and estimation of unknown analyte concentrations from Luminex data, the main purpose of the LuminexPipeline package is to standardise Luminex data processing steps after curve calibration. For example, data cleaning, recording metadata information and standardising analyte names.

The standard curve fitting utility incorporated into the LuminexPipeline is not included in the pipeline's automation with Nextflow but serves as a quality control utility for certain use cases where Luminex data come without the estimated concentration values or when there is a need for validation of the estimated concentrations. The Luminex pipeline therefore utilises the full functionality of the drLumi R package to generate standard curves and estimate unknown concentrations from the median fluorescent intensity (MFI) using the ignore, include, subtract and the constraint methods of treating background noise employed by the drLumi package. It is important to note that the use of a different background treatment method other than the one employed by the Luminex instrument may result in different concentrations estimates compared to the estimated concentrations generated by the Luminex instrument.

Code incorporation was done with the original drLumi functions but not as a package dependency because, during the development phase of the LuminexPipeline package (September 2022), the drLumi package was not available for automatic installation on the comprehensive R archive network (CRAN) but was only available as an archived tarball. Thus, incorporating the drLumi package as a dependency would have caused complications in the subsequent installation of the LuminexPipeline package.

3.2 Containerisation

One of the major challenges in attaining reproducibility, especially in scientific computing is the challenge of inconsistent software versions, dependencies, as well as computational environments, which contribute to numerical instability and inconsistent results. To successfully reproduce published analytical results given the code and data, one will need to replicate the computational environment in which the original analysis was run. Replicating this environment will involve the use of the same OS, software and the exact software versions used in the original analysis. However, this process can be very challenging. For example, dealing with complicated software dependencies during installation and, to some extent, incompatible hardware environments (Kim, Poline & Dumas, 2018) as well as challenges with accessing the exact software versions. Different software versions can generate significantly different results (The Turing Way Community, 2022). Encountering these challenges may be a big barrier for scientists, especially, those with minimal computational background, to attain analytical reproducibility.

A containerised environment can efficiently address these challenges. A container is an isolated runtime environment independent of its host environment. One can think of it as an OS running on top of another OS. For example, running Ubuntu 18.04 on a host computer that runs on Ubuntu 22.04. A container, therefore, is a fully functional virtual environment with an independent OS that can encapsulate all software (the exact version), software dependencies, code, data etc., used in an analysis. This virtual environment is portable and can be shipped or shared with other researchers to facilitate analytical reproducibility by allowing analyses to be run in the same computational environment as the original, even after several years. The portability attribute of containers also circumvents the challenges encountered in managing software dependencies and versions during installation. Unlike other virtual environments like virtual machines (VM) whose OS kernel is independent of the host OS, containers do not have a full copy of their OS kernels. They utilise the host OS kernel and are, thus, "lightweight" (Mitra-Behura, Fiolka & Daetwyler, 2022). Additionally, their design strips off the layer of the GUI to conserve computing resources.

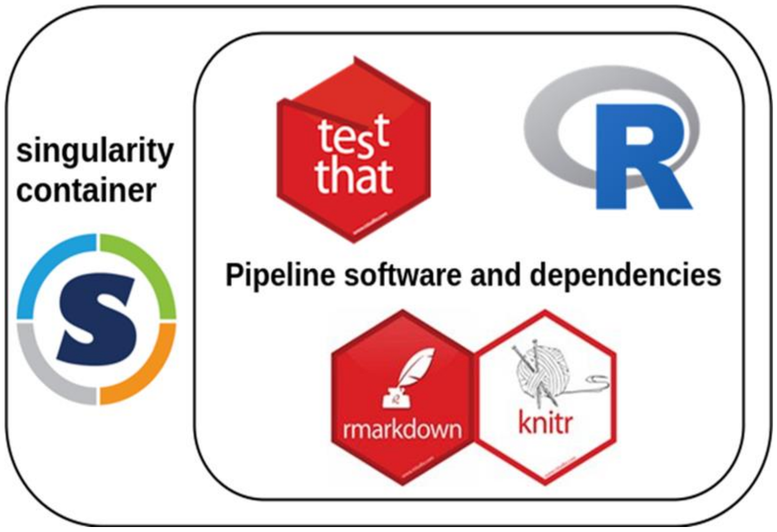
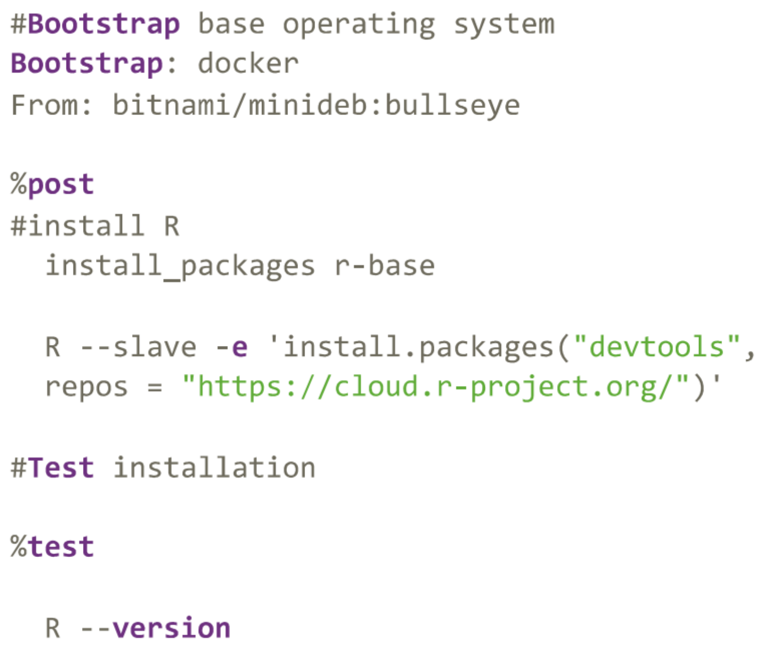


Figure 11. An illustration of an isolated containerised environment encapsulating installed software (the inner circle) independent of its host environment (the outer circle)

Singularity containers can be developed in two ways. 1) Write a set of commands in a definition file, the "recipe file" (Figure 12). This definition file has all the commands to set up the container environment, install software or even copy files to the container. This is a highly reproducible approach since all the ingredients for building the container are documented in the definition or recipe file. However, this is a non-interactive process, and therefore, may render its development a to be a bit limiting. 2) The sandboxing method on the other hand is an interactive process where a base container (usually with only the OS) is built as a writeable directory on the host computer. The developer can then interactively modify the base container (install software, add files etc.) and distribute it as a sandbox (writeable directory) or convert it into an unwritable image — singularity image file (.*sif*). The advantage of sandboxing is the convenience of modifying the container as well as its interactivity; however, reproducing a container built as a sandbox may be a challenge if modifications done to it are not appropriately recorded.



```
#Bootstrap base operating system
Bootstrap: docker
From: bitnami/minideb:bullseye

%post
#install R
  install_packages r-base

  R --slave -e 'install.packages("devtools",
  repos = "https://cloud.r-project.org/")'

#Test installation

%test

  R --version
```

Figure 12. Sample definition file containing the commands to build a container. The code is annotated to describe each code section. One only needs the "Bootstrap" and "From" sections to build a base container. Definition files have the extension (.def). See the appendix for the definition file used to build the LuminexPipeline container.

3.2.1 Building the container

To improve reproducibility in the Luminex pipeline's execution, I developed a Singularity (Kurtzer, Sochat & Bauer, 2017) container that encapsulates all the pipeline's software and dependencies. Below, I describe the steps used to build this container.

1. I developed container 1 – a sandbox with Minideb (James Westby et al., 2022) as the base OS from a definition file (Figure 12). Minideb is a minimalist OS based on the Debian OS with wide compatibility with most Linux programs.

singularity build --fakeroot --sandbox *name_of_sandbox path_to_definition_file*
A snippet of the command we ran to build a Singularity sandbox.

2. I then installed R (R Core Team, 2021) and all LuminexPipeline-dependent R packages in container 1.
3. I ran the `ldd` (list dynamic dependencies) command on the R executable files in */usr/lib/R/bin/exec/R* and kept a record of all the shared library dependencies, shared object files and program files from the `ldd` command output.
4. I recursively followed each `ldd` output (i.e., shared object files and library dependencies) with another `ldd` command to determine all the dependencies and dynamic links needed to run R, while keeping the records of each `ldd` output.
5. I built container 2 – a sandbox with Minideb as the base OS from a definition file (figure 12)
6. I then copied all the program files (installed packages), binary files and shared object files (libraries) recorded in steps 3 & 4 (see Appendix for the full list) from container 1 to container 2. The copied files are the minimally required files necessary to run R (R Core Team, 2021).
7. I ran the `ldconfig` command to cache and link the shared libraries.

The above-described steps are an example of a manual multi-stage build process, utilised to build lightweight containers. This eliminates unnecessary software and dependencies, e.g., those automatically downloaded during installation to facilitate the installation process, but which are not required to run the installed program. Thus, the multi-stage build process typically uses two steps: an initial or development stage where software is installed and built, and the second, or deployment stage, where only the minimally required components necessary for the software's full operation are copied from the initial or development stage. This process is used to build containers with significantly smaller sizes (Huls, 2022).

3.3 Pipelines and workflow management systems

Workflow management systems are used to manage, scale and dispatch multiple-step computational analyses which are typical of Bioinformatics and other scientific computing disciplines. The multiple-step computational analysis is usually referred to as a pipeline or a workflow. In Bioinformatics, these steps usually include trimming the data, data cleaning, alignment to reference genomes, recalibrating quality scores, quantification etc. Manual execution of each of the computational steps in a reliable and reproducible manner can, however, be frustrating for the analyst or researcher, especially when the analytical steps are required to be executed in a specific order (Jackson, Kavoussanakis & Wallace, 2021). To worsen the frustrations, each analytical step may often require a specific software utility for its execution. Workflow management systems (WMS), therefore, join together the execution of multiple-step analyses in an automated and reproducible manner while facilitating a seamless execution. WMSs also capture provenance information such that certain analytical results can be easily attributed to a specific pipeline (Committee on Reproducibility and Replicability in Science, the National Academies of Science, Engineering and Medicine, 2019a). Additionally, many WMSs facilitate robust pipeline execution with support for re-entry or resumption of execution from where an error stopped execution, and can selectively execute only parts of a pipeline, contrary to classical shell scripts which do not support re-entry unless explicitly defined by the authors (Jackson, Kavoussanakis & Wallace, 2021). Furthermore, most WMSs can automatically scale computational pipelines from a single core PC to a large computational resource structure like the cloud or compute cluster environments such as high performance computers (HPC) without altering the code (Koster & Rahmann, 2012; Di Tommaso et al., 2017). This is made possible by the support for job schedulers, e.g., PBSpro, and Slurm, in many WMSs. WMSs can further enhance computational reproducibility through the support for containerised computations on HPC clusters, which allows for easy management of numerical instability (variation in computational results, across HPC environments), one of the main sources of computational irreproducibility (Di Tommaso et al., 2017). Many WMSs have been developed to facilitate reproducibility in scientific computing, especially in Bioinformatics. These include the common workflow language (CWL) (Amstutz et al., 2016), Biopipe (Hoon et al., 2003), Nextflow (Di Tommaso et al., 2017), Galaxy (Hérisson et al., 2022), and Snakemake (Koster & Rahmann, 2012).

3.3.1 Choosing a workflow management system

Different WMSs have been developed based on different philosophies, functionalities, and implementations. I aimed to select a WMS framework which can dispatch production-ready workflows via both parallel and serial processing, accommodates a wide variety of software and complex data flow dependencies (the output of an upstream process as input for a downstream process), allows for a wide range of input data types and allows for fixed and customisable parameters as recommended by (Leipzig, 2017). Additionally, I considered other factors such as scalability, ease of use, installation, documentation, technical support, development, implementation, stability, future support and popularity in the Bioinformatics community. I surveyed and reviewed 16 WMS which are: Nextflow, Big data scripting, Snakemake, Python corral, Bioqueue, Ruffus, Common workflow language, Script of scripts, Makeflow, Canonical workflow framework for research, Pegasus, Kepler, Airflow, VisTrails, Workflow description language, Bpipe and Pipeline pilot. I selected two, Snakemake and Nextflow for prototyping after reviewing all 16 WMSs against the selection criteria. Prototyping involved using simple R scripts and dummy data to assess the WMS functionality as similarly implemented by (Jackson, Kavoussanakis & Wallace, 2021). Prototyping also enabled us to thoroughly test the two selected WMSs against some of the selection criteria that is scalability, ease of installation, ease of use, and ease of development.

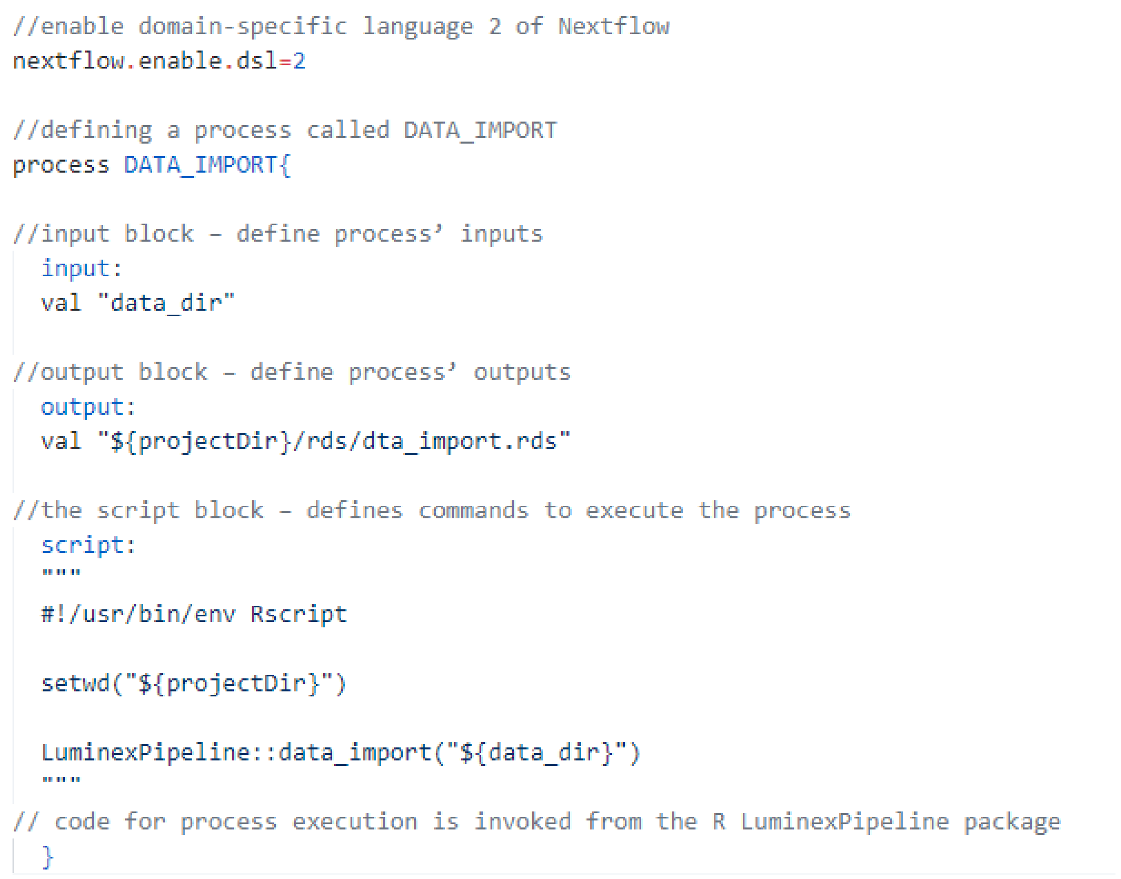
Table 4: Summary of steps and procedures implemented for selecting a workflow management system. Adapted from (Jackson, Kavoussanakis & Wallace, 2021)

Steps	**Tasks**
Identify potential WMS and narrow down candidates	An online survey of available WMS that are likely tailored to our needs. That is the ease of use, HPC support, containerisation support, etc.
	Review of documentation of the surveyed WMS to further assess the eligibility of our set criteria.
	Weekly meetings to discuss, assess and shortlist potential candidates.
Assess candidates using prototypes	Prototype shortlisted candidates with sample R scripts and dummy data to assess the functionality.
	Further, assess each candidate's suitability for the SATBBI group

3.4 Luminex Pipeline with Nextflow

I developed a customised Nextflow script to automate the Luminex pipeline execution. Unlike many Bioinformatics workflows e.g., the nfcore pipelines (https://nf-co.re/pipelines), which utilise different software whose invocation is through the command line e.g., FASTQC (Andrews, Krueger & Segonds-Pichon, 2020), or TrimGalore (Krueger et al., 2021), the Luminex pipeline was implemented using only the R statistical software (R Core Team, 2021) utilising several different R-based packages for data wrangling (eg. dplyr), visualisation (eg. ggplot2), and reporting (e.g., Rmarkdown).

Each modularised step (e.g., imputation, cleaning, reporting) of the Luminex pipeline was written as a Nextflow process. A Nextflow process is *"the basic processing primitive to execute a user script"* (Seqera Labs, 2022), that is, a module of the user's script. A Nextflow process (Figure 13) usually has an input block (defines input files for the process), an output block (defines the output of the process) and a script or shell block (defines the command or script executed to process the input into the output). The Input and output files in the implementation were defined using a customisable configuration file (Figure 14) in line with current best practices (Taschuk & Wilson, 2017). The script block for each process was written to invoke R through the command line. Thus, all the commands for executing each Nextflow process were invoked from the LuminexPipeline package. To streamline the processes in an automated manner, the output of each Nextflow process was given as the input to the next Nextflow process (See the Appendix for the full Nextflow script).

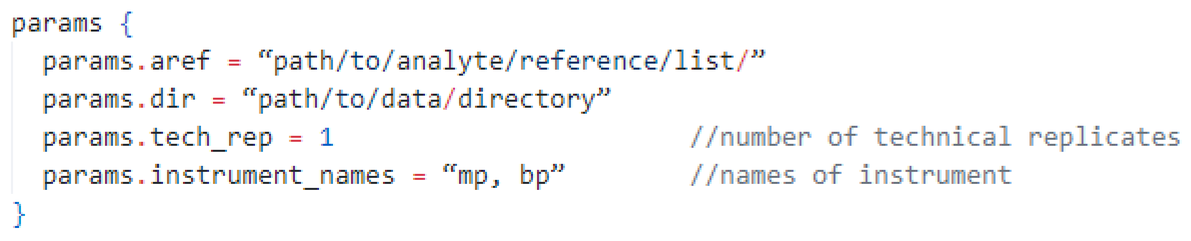


```
//enable domain-specific language 2 of Nextflow
nextflow.enable.dsl=2

//defining a process called DATA_IMPORT
process DATA_IMPORT{

//input block - define process' inputs
  input:
  val "data_dir"

//output block - define process' outputs
  output:
  val "${projectDir}/rds/dta_import.rds"

//the script block - defines commands to execute the process
  script:
  """
  #!/usr/bin/env Rscript

  setwd("${projectDir}")

  LuminexPipeline::data_import("${data_dir}")
  """
// code for process execution is invoked from the R LuminexPipeline package
  }
```

Figure 13. Illustration of the input, output, and script code blocks in an excerpt Nextflow process used in the Luminex pipeline

3.4.1 Configuration and parameterisation

The ability to generalise a pipeline to accommodate other data sets is an important component for achieving reproducibility with pipelines. Parameterisation allows the main pipeline script to be unaltered while accommodating different datasets. It also allows one to specify the compute resources to execute a job (e.g., number of CPUs, threads), the type of job scheduler to use for HPC computations, as well as the specific containers to run the job. In Nextflow, a configuration file named `nextflow.config`, in the working directory is used to specify the parameters for the pipeline execution, e.g., inputs, outputs, executors, and containers. I, therefore, parameterised the input and output files, e.g., technical replicates, and analyte reference of the Luminex pipeline (Nextflow script) to facilitate ease of use and generalisability.



```
params {
  params.aref = “path/to/analyte/reference/list/”
  params.dir = “path/to/data/directory”
  params.tech_rep = 1                          //number of technical replicates
  params.instrument_names = “mp, bp”           //names of instrument
}
```

Figure 14. A sample of a simple configuration file content. A user of the pipeline only needs to change the parameter values to process and analyse different data sets and does not need to alter the pipeline script.

3.5 Statistical summary report

3.5.1 Distribution of analytes

I created an R function that returns a summary table of the minimum value, maximum value, mean, quartiles (1st, 2nd and 3rd), and missingness for each analyte concentration using the R dplyr v1.0.10 `groupby()` function. Minimum and maximum values were computed with the `min()` and `max()` base R functions. I used the base R stats v4.2 `quantile()` function to calculate the quartiles of analyte concentrations.

I also computed measures of central tendency with the arithmetic mean, which is the average of all the data points. This is a good representative value of all the data points in the absence of extreme values (outliers). However, because the mean is sensitive to outliers, the median (the 2nd quartile) is a good alternative for a representative value of the data points. I calculated the means using the base R `mean()` function.

I provided a graphical alternative for visualising the summary of the data using boxplots and histograms. Visualisation helps with providing a generalised idea of the distribution of the data. Boxplots visualise the distribution by indicating the positions of the minimum value (below which data points are classified as extreme values), the 1st quartile, the 2nd quartile (median), the 3rd quartile and the maximum value (above which data points are classified as outliers). A histogram visualises the distribution by creating bins (range of numbers) and representing the frequency of the bins with a bar. A histogram can give a good hint of the distribution of the data. I used the ggplot2 package for visualising the distributions.

3.5.2 Outliers

An outlier is defined as an observation in a sample that differs significantly from other observations in the sample (Grubbs, 1969). Thus, for unidimensional data, they are the observed values at the extreme tails of the distribution of the underlying data (Salgado et al., 2016). An outlier may be observed due to one of two reasons. First, it could be a manifestation of the inherent random variation in the sample — in which case, the outlier observations should be maintained and treated or processed equally as the other sample observations. Second, it could be a manifestation of errors in the data collection process, e.g., instrument calibration errors, pipetting errors, or data entry errors. This situation warrants further investigation to determine the reason for the gross deviation. Based on the outcome of the investigation and the experimental context, outlier observations may be corrected and retained or may be removed from the data. Generally, the processing of outliers and the extent to which they are incorporated in an analysis should be reported (Grubbs, 1969). There are many methods for detecting outlying observations.

In my statistical summary report, I employed three methods for reporting outliers; The z – scores, modified z-score and Tukey’s method. The z-score test transforms the data points as a scale of reference to the number of standard deviations away from the mean. The z-score test assumes the data are normally distributed. I computed the z-scored test using the equation:

$$z_i = \frac{x_i - \bar{x}}{s}$$

Where $\bar{x}$ is the arithmetic mean and s is the standard deviation of the data.

I considered z_i values >= 3 as outliers. Based on the assumption of a normal distribution, 3 standard deviations away from the mean (0) of a standard normal curve will be at the extreme tails (outliers), beyond 99.7% of the data.

Since the arithmetic mean used in the calculation of z-scores is sensitive to extreme values, the modified z-score method addresses this challenge by using the median and the median absolute deviation (MAD). This approach calculates the variation in terms of the MAD away from the median. The modified z-score also assumes that the underlying data are normally distributed (Salgado et al., 2016). I computed the modified z-score test using the equation:

$$M_i = \frac{0.6745(x_i - \tilde{x})}{MAD}$$

where $\tilde{x}$ is the median and $MAD = median\{|x_i - \tilde{x}|\}$. We considered $|M_i|$ values >= 3.5 as outliers.

Tukey's method classifies outliers using a specific distance below the first quartile, Q1 (first 25% of the ordered data points) and above the third quartile, Q3 (last 25% of the ordered data points, also known as the 75th percentile). The interquartile range (IQR) is the distance between Q1 and Q3. *Inner fences* are data points lying 1.5*IQR (1.5 multiplied by IQR) below Q1 and above Q3. *Outer fences* are demarcated with data points lying 3*IQR below Q1 and above Q3. Data points between the inner and *outer fences* are classified as *possible* outliers whereas those above the outer fences are classified as *probable* outliers (Salgado et al., 2016).

In the statistical summary report, I used the probable outliers of Tukey's method to identify outliers. Thus, I classified data points below or above 3*IQR as outliers. I extracted the number of outliers detected for each outlier detection method in a table.

These methods of outlier detection are meant to inform the analyst of the possible outliers in the data.

3.5.3 Correlation

Correlation measures the association between variables. It answers the question of whether increasing one variable increases, decreases or does not have any relationship with another variable. The coefficients of correlation are between -1 and 1. Coefficients close to -1 indicate a strong negative correlation (opposite association), coefficients close to 0 are indicative of a weak association while correlation coefficients close to 1 indicate a strong positive association. Correlation can be measured in three different methods: the Pearson, Spearman, and Kendall correlation. Pearson correlation is a parametric method which is dependent on the distribution of the underlying data. The Spearman and Kendall methods are rank-based methods independent of the distribution of the underlying data.

To compute the associations between analytes in the statistical summary report, I used the Spearman correlation method which is a generally more robust method for computing the correlation of variables whose underlying distributions are unknown. I used functions in the Hmisc v4.7 package in R to compute a correlation matrix (correlation between all analytes). I visualised the associations between all analytes in a correlogram using the `corrplot()` function from the corrplot v0.92 package in R.

3.5.4 Test of non-normality

Many statistical tests such as the t-test assume a normal distribution of the residuals of the underlying data. Residuals, also known as errors, are the differences between the observed and predicted values in a regression analysis. More simply, residuals are the differences between the data points and the mean of univariable data. Thus, to ensure that certain statistical assumptions are not violated, it is useful to assess the normality of the residuals of the underlying data before the main analysis. I, therefore, utilised three methods of assessment of normality in the statistical summary report. These methods are numerical tests (kurtosis, skewness, the Shapiro-Francia test) and a graphical test (quantile-quantile plot).

Kurtosis is a metric for measuring how extreme the tails of a distribution are relative to a normal distribution. Datasets with long or heavy tails are typically known of being highly kurtotic or having many outliers, whereas the opposite is true for light-tailed distributions. A standard normal distribution has a kurtosis of 3 (NIST/SEMATECH, 2012). This value is used as a reference for comparing the tails of distributions relative to a standard normal distribution. I computed the excess kurtosis using the `Kurt()` function from the DescTools v0.99.46 package in R.

Skewness measures the symmetry of distribution relative to a normal distribution. A symmetrical distribution has the left and the right sides of the distribution as divided by the centre point (the mean value for standard normal distribution) looking the same. A standard normal distribution has a skewness of zero. Negative values of skewness indicate a left-skewed distribution, i.e., heavy tails on the left of the distribution whereas positive skewness indicates skewness to the right, i.e., heavy tails on the right of the distribution. I computed skewness using the `Skew()` function from the DescTools v0.99.46 package in R.

Shapiro-Francia test is one of the numerical tests for assessment of non-normality, the test statistic, *W'*, is the *"square of the Pearson correlation coefficient between the ordered sample values and the expected standard normal order statistics ('normal scores')*" (Royston, 1993). Small *W'* values indicate non-normality. An associated p-value tests the null hypothesis that the data are normally distributed. I computed the Shapiro-Francia test for the statistical summary report using the `DescTools::ShapiroFranciaTest()` function in R.

Quantile-Quantile (q-q) plot is a graphical approach (a scatter plot) for assessing the normality of the residuals of a distribution. A q-q plot orders the observed data in ascending order and plots its quantiles against the ordered quantiles computed from a theoretical normal distribution (Clay Ford, 2015). I generated q-q plots for statistical summaries using the `ggplot()` and `stat_qq()` functions from the ggplot2 v3.6.6 R package.

3.5.5 Comparison between groups

I performed hypothesis tests to determine whether analyte concentrations are likely from the same distributions between certain groups e.g., Treatment vs Control. I used the Wilcoxon rank sum test, also known as the Mann-Whitney test employing the `wilcox.test()` function from the base R stats package for this test. This is a non-parametric alternative to the two-sample t-test. This test assigns ranks to the ordered (smallest to the largest) observations. That is, the smallest observation or value is assigned the least rank while the largest observation or value is assigned the highest rank. Tied observations are assigned the average of the ranks (Wild & Seber, 1999). The Wilcoxon test is based on the ranks of all combined sample observations. The test statistic is the sum of ranks of the ordered observation in one sample of the data.

3.5.6 Shiny app

Shiny (Chang et al., 2022) is an R package and framework for creating interactive web applications and dashboards in R without any prior knowledge of JavaScript, CSS or HTML (Wickham, 2021). R shiny enables interactivity of the analyses by utilising reactive programming which automatically updates code outputs when its dependencies (inputs) are altered. The interactive feature of Shiny applications can be useful for reporting results and allows a user to easily sift through a large volume of report information to a specific portion of interest. This interactivity can facilitate and fast-track exploratory data analysis. To utilise the interactivity of R Shiny, I developed a Shiny application to report the statistical summaries and facilitate exploratory data analysis. I accomplished this by using the R Shiny V1.7.2 package (Chang et al., 2022) in R.

3.6 Datasets

In addition to unit testing, the pipeline was tested with real datasets. These datasets were collected from ongoing research by the SUN-IRG. The datasets included clinical data and participant data with a unique ID serving as a primary key to link the two data sets. The Luminex data had 45 distinct analytes with concentration values and fluorescent intensity values, dilution values, plate numbers etc. The participant data had information on sex, and group (control or treatment).

3.7 Materials and tools

3.7.1 Hardware and Operating system

Code was written with a Dell desktop computer with computing resources of intel core i7 (octa-core processor), 16GB RAM, and 1TB storage. The operating system used was Kubuntu V22.04, a flavour of the Ubuntu OS.

R package development was done on a Linux server running Ubuntu 22.04 using Rstudio server V22.02.3-492.

3.7.2 List of tools and packages

The following tools and software packages were used for the project:

Draw (https://draw.io/)

Nextflow V22.10.1 (https://www.nextflow.io/)

R packages (corrplot, DescTools, dplyr, english, ggplot2, gsubfn, Hmisc, lubridate, magrittr, minpack.lm, msm, plyr, purrr, readr, reshape, stringr, tibble, tidyr, drLumi)

R V4.2 (https://cran.r-project.org/)

Rstudio IDE V22.07.2-576

Rstudio server V22.02.3-492

Singularity V3.5.3 (https://docs.sylabs.io/guides/3.5/user-guide/introduction.html)

Visual Studio Code IDE V1.7.2

Zotero (https://www.zotero.org/)

CHAPTER 4: RESULTS

4.1 Improvements to the R LuminexPipeline package

In my MSc research project, I focused on making important improvements to the R LuminexPipeline package.

The LuminexPipeline package can be installed by either of the following two methods.

The first option is to use the R environment and the `install_github()` function from the devtools R package (Wickham, Hester, et al., 2022). The parameter or argument to this function is the GitHub package package repository, `“Asimeng/LuminexPipeline”`. Using this method requires the user to have the devtools R package installed. The following code snippet can be used to install the package.

```
devtools::install_github(“Asimeng/LuminexPipeline”)
```

The second option to install the package is to do it via the Linux command line by using the already-built source file (compressed source code). This file is available for download at https://github.com/Asimeng/LuminexPipeline/blob/main/inst/package_source_code/LuminexPipeline.tar.gz. Once downloaded, package installation can be initiated by executing the following command on the command line.

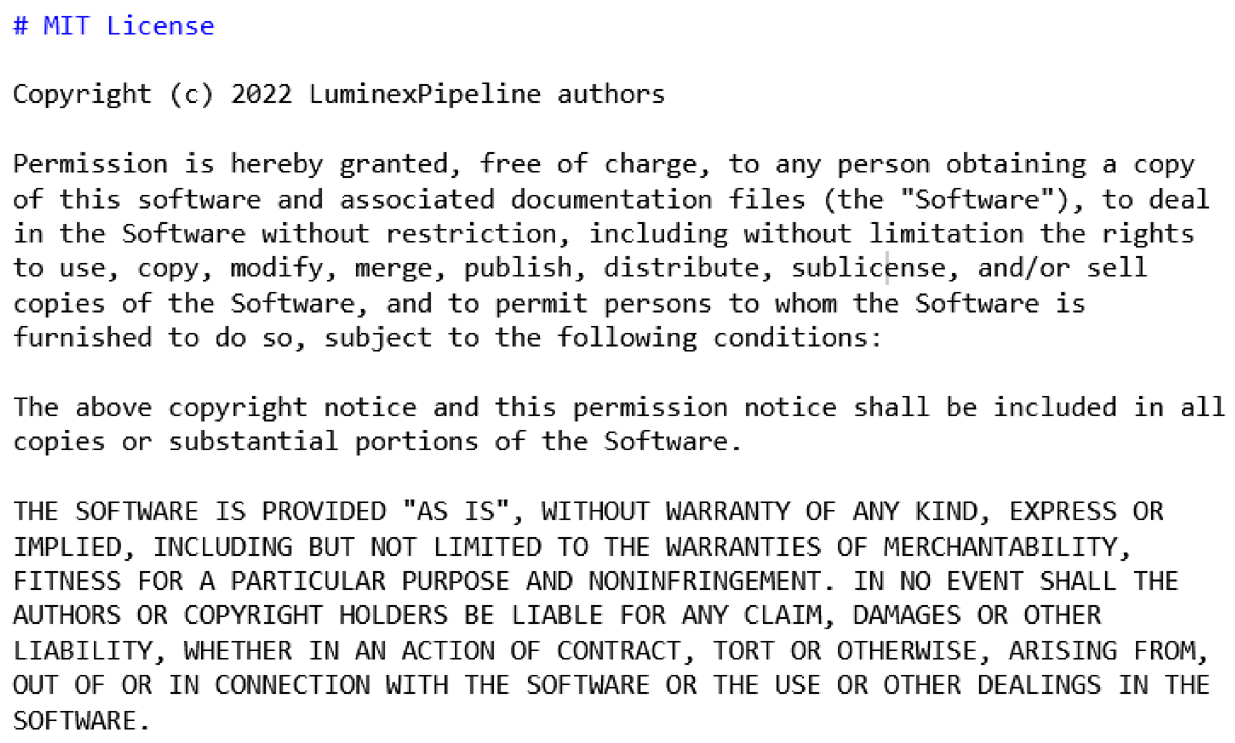
```
R CMD INSTALL LuminexPipeline.tar.gz
```

Note: This code snippet assumes that the binary package file is in the current working directory. One will need to provide the full path to the file if working from a different directory.

Also, a different installation alternative with the built source file can be done in R with the following code snippet.

```
install.packages(“path_to_source_file”,repos=NULL,type=“source”)
```

The LuminexPipeline R package is publicly available at https://github.com/Asimeng/LuminexPipeline and its terms of use are licenced with the MIT license, which is one of the major open-source licenses.

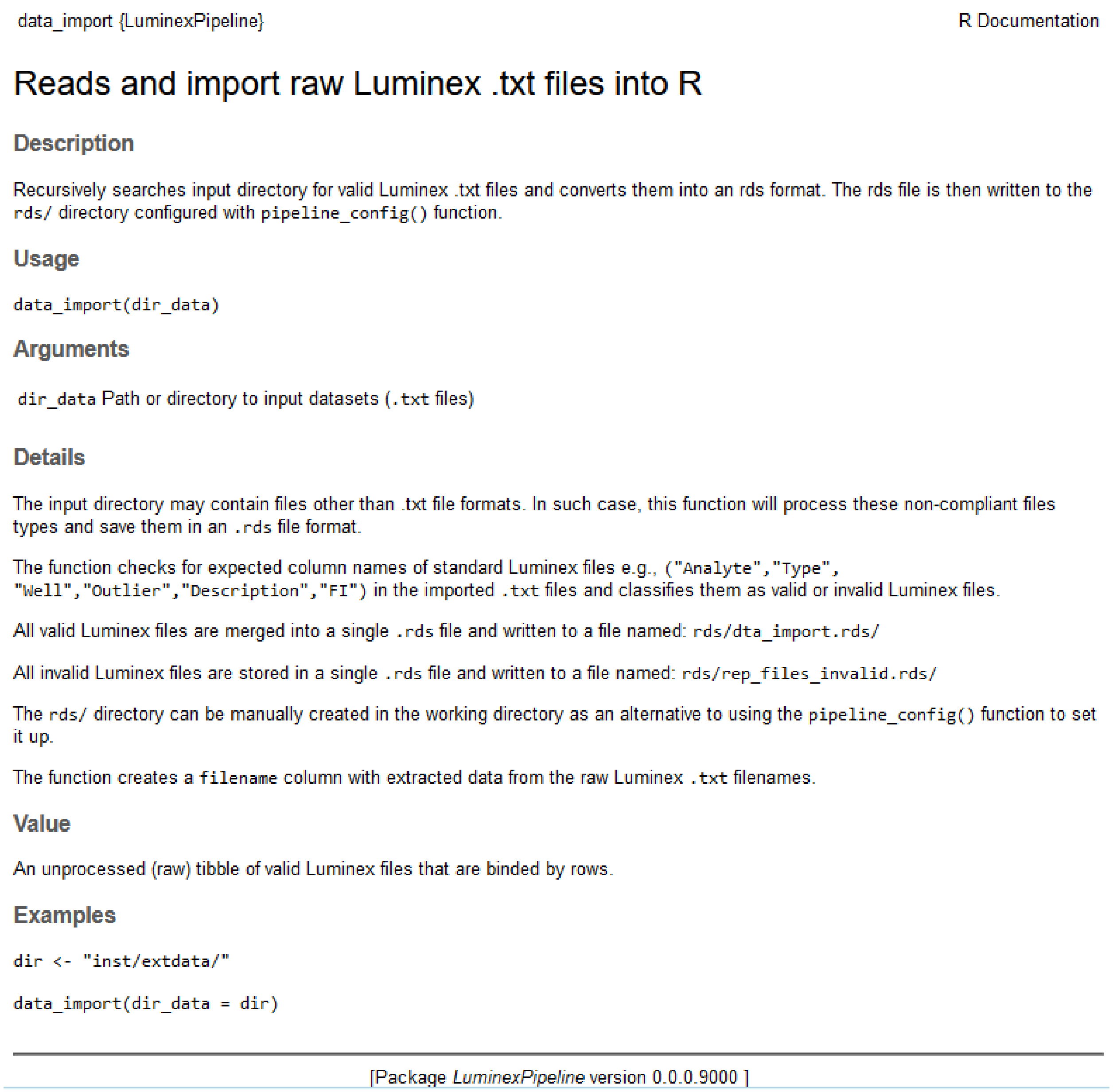


```
# MIT License

Copyright (c) 2022 LuminexPipeline authors

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

Figure 15. The general terms of use and permissions granted by the MIT license.

4.1.1 Package Documentation

The major package functions are documented with descriptions, usage, and examples to facilitate ease of use (Figure 16). This is in line with current best practices to contribute to achieving reproducibility.



data_import {LuminexPipeline}   R Documentation

Reads and import raw Luminex .txt files into R

Description

Recursively searches input directory for valid Luminex .txt files and converts them into an rds format. The rds file is then written to the `rds/` directory configured with `pipeline_config()` function.

Usage

```
data_import(dir_data)
```

Arguments

`dir_data` Path or directory to input datasets (`.txt` files)

Details

The input directory may contain files other than .txt file formats. In such case, this function will process these non-compliant files types and save them in an `.rds` file format.

The function checks for expected column names of standard Luminex files e.g., `("Analyte","Type", "Well","Outlier","Description","FI")` in the imported `.txt` files and classifies them as valid or invalid Luminex files.

All valid Luminex files are merged into a single `.rds` file and written to a file named: `rds/dta_import.rds/`

All invalid Luminex files are stored in a single `.rds` file and written to a file named: `rds/rep_files_invalid.rds/`

The `rds/` directory can be manually created in the working directory as an alternative to using the `pipeline_config()` function to set it up.

The function creates a `filename` column with extracted data from the raw Luminex `.txt` filenames.

Value

An unprocessed (raw) tibble of valid Luminex files that are binded by rows.

Examples

```
dir <- "inst/extdata/"

data_import(dir_data = dir)
```

[Package *LuminexPipeline* version 0.0.0.9000]

Figure 16. An example of documentation for one of the functions in the LuminexPipeline package

4.1.2 Unit testing

As part of best package development practices, the implementation of unit testing helped me to improve the LuminexPipeline package's robustness. Thus, for example, in the instance of bad input, the package will not break with a cryptic error message but will instead generate a useful error message to help with debugging. Unit testing also helped me improve the package's numerical accuracy by testing its functions to return the expected output and results.

Through unit testing, I improved the robustness of our functions by refactoring the code as well as introducing several if statements to robustly handle unexpected inputs –generation of useful error or warning messages.

4.1.3 Quality control

I used functions available in the drLumi R package (Sanz et al., 2017), to generate standard curves from standard dilution concentrations (Figure 17) to estimate the unknown concentrations. The LuminexPipeline package now has the full curve calibration utility of the drLumi package and can therefore provide the goodness of fit metrics and visualisation for quality assessment.

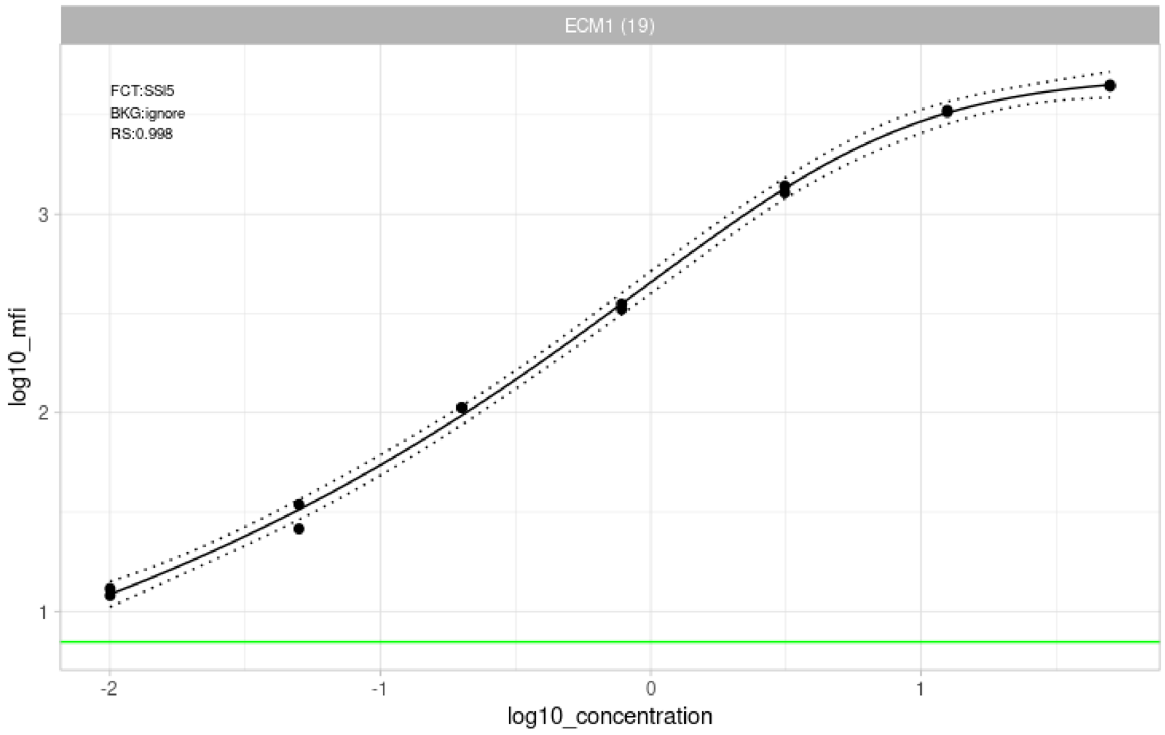


Figure 17. An example of a standard curve generated from the LuminexPipeline package using functions borrowed from the drLumi package. This curve attained convergence with the 5-parameter logistic curve function and it is plotted ignoring background noise.

4.2 Containerisation

The use of a containerised runtime environment in the pipeline execution significantly improves analytical reproducibility in Luminex data processing by addressing the challenge of numerical instability arising from software versioning and different computational environments. Furthermore, the implementation of containerised analyses fosters reproducibility through the recording of provenance information and the portability and ease of sharing of the containerised environment to have results easily reproduced by an independent group or researcher. The containerised runtime environment runs on the bitnami/minideb (https://github.com/bitnami/minideb) OS image, which is a lightweight version of Debian OS. The deployed environment with all the necessary software needed to run the Luminex pipeline is available as a Singularity image file (*.sif*) — an easy-to-distribute file. To foster transparency, I provide the definition file, so called recipe file, to reproduce the container at https://github.com/Asimeng/LuminexPipeline_container.

4.3 Pipelines and workflow management systems (WMS)

The selection of a pipeline framework or WMS was to have one consistent framework to be used for the implementation of current and ongoing pipeline developments as well as for future pipeline development tasks undertaken at the SATBBI. Because of this, I selected a WMS framework that is highly generalisable to different programming languages and data file types. Also, I selected the WMS framework that is best meets our selection criteria, which were easy to install, use, develop and implement as well as requires minimal programming expertise.

Based on the review of existing WMSs, I selected two WMS, namely Nextflow and Snakemake, for side-by-side comparison (Table 5). I chose the Nextflow WMS (Di Tommaso et al., 2017) to be used as the LuminexPipeline framework. Below, I summarise the findings in prototyping and reviewing the two WMS.

Table 5: Summary of comparison of shortlisted WMS — Nextflow & Snakemake

Evaluation criteria	**Nextflow**	**Snakemake**
Log files	Yes	No
Dry-run	No	Yes
Citations (wide use)	980*	1861*
License	Apache 2.0 (open source)	MIT (open source)
Re-entry	Yes	Yes
Support for containers	Yes	Yes
Support for HPC and Cloud	Yes	Yes
Implementation	Dataflow programming	GNU Make build system
Workflow parallelisation	Yes	Yes

*Citations assessed using Google Scholar (https://scholar.google.com/) on September 2022

4.3.1 Implementation & ease of use

I found Nextflow to be fairly straightforward with its execution of steps or processes as compared to Snakemake. Nextflow implements a data flow programming paradigm (from top to bottom) where executions automatically start with the availability of input data through input data channels (Di Tommaso et al., 2017). I favoured this approach to the GNU Make style implementation of Snakemake which pre-computes all workflow dependencies starting from the expected output file(s) through to the input file(s) in a directed acyclic graph DAG. In other words, Snakemake executes its tasks by first mapping out all dependencies or rules that lead to the generation of the specified output file — from the bottom up to the input data. If a dependent file is absent, Snakemake searches for defined tasks or rules that creates the file. Knowledge of this implementation is needed to guide the scripting of snakefiles (a Snakemake script). Since I had no prior GNU Make knowledge, I found this a bit confusing initially during prototyping in Snakemake, unlike Nextflow which was more intuitive and straightforward. Furthermore, Nextflow's process execution is done in an isolated sub-directory which collects metadata of the execution process as log files — which is useful for debugging. This feature is absent in Snakemake. Both systems support workflow parallelisation. Snakemake has a "dry-run" feature that displays the commands that would be executed and can verify that input files are present without actually executing them (Jackson, Kavoussanakis & Wallace, 2021). This feature is absent in Nextflow. Both systems allow re-entrancy (resumption of execution after the unexpected exit of the job). Both WMSs also allow external scripts, e.g., Python, R, to run.

4.3.2 Installation

The major dependency required for installing Nextflow is Java ≥ 11 and BASH ≥ 3.2. Once these requirements are met, Nextflow can easily be installed and run on any "POSIX compliant" system (e.g., Solaris, Linux) as well as on Windows via the Windows Subsystem for Linux (WSL). Snakemake, on the other hand, requires Python to run and its installation can seamlessly be done with the Conda or Mamba package managers. However, these package managers can be challenging to deal with, especially for novice users. I found both WMS very easy to install. However, I preferred Nextflow because its installation is not natively bound or attached to the Conda or Mamba package managers, which sometimes can be challenging to deal with.

4.3.3 Wide use

I found both WMSs to be very widely used in the Bioinformatics community. Although Snakemake has more citations, probably because it is older, Nextflow is equally widely used and used by major research institutions. For example, the Fred Hutchinson Cancer Center, a world-leading biomedical research centre has adopted Nextflow as its workflow management, a strong indication of its high regard in the Bioinformatics community.

4.3.4 Scalability

Both systems can scale well, running workflows on a variety of hardware from a single core computer to HPC or cloud environments without changing the workflow script.

4.3.4 Licensing

Snakemake uses an MIT open-source license, whereas Apache 2.0 open-source license is used for Nextflow. These allow for the free use, development, and distribution of workflows with both workflow management systems.

4.3.5 Future support

Nextflow has a community of Bioinformatics pipeline creators, the nf-core. This growing community is a good indication of future stability and future support for Nextflow. Also, its nf-core pipelines are free and open source (https://nf-co.re/pipelines) and can be easily adapted and tailored to suit a researcher's needs. Additionally, they can help reduce the learning curve of Nextflow. Snakemake on the other hand has the Snakemake workflow catalogue (https://snakemake.github.io/snakemake-workflow-catalog/) with standardised Snakemake workflows. Under this criterion, I found both Snakemake and Nextflow to have great prospects for future support and stability with their increasing popularity and citations, see table 2.

4.4 Luminex Pipeline with Nextflow

Here I describe the running of the LuminexPipeline in the Nextflow framework. The Luminex pipeline execution is triggered by running the Nextflow script (see Appendix). Before executing the Nextflow script, the parameters for input files, instrument names, analyte references, and the directory of the container should be set up in the pipeline's configuration file. Once the pipeline is dispatched with Nextflow, processing and execution are done in the container to achieve numerical stability and reproducibility. Within the container are all software (including the

LuminexPipeline R package) needed to run the analysis. The outputs of the execution are *.rds* files and an HTML or pdf report file written to a specified output directory (Figure 18). The pipeline can be automatically dispatched by running the following command on the command line.

```
nextflow run path/to/nextflow/script
```

Alternatively, it can be dispatched by setting the parameter values alongside the command to run the Nextflow script on the command line.

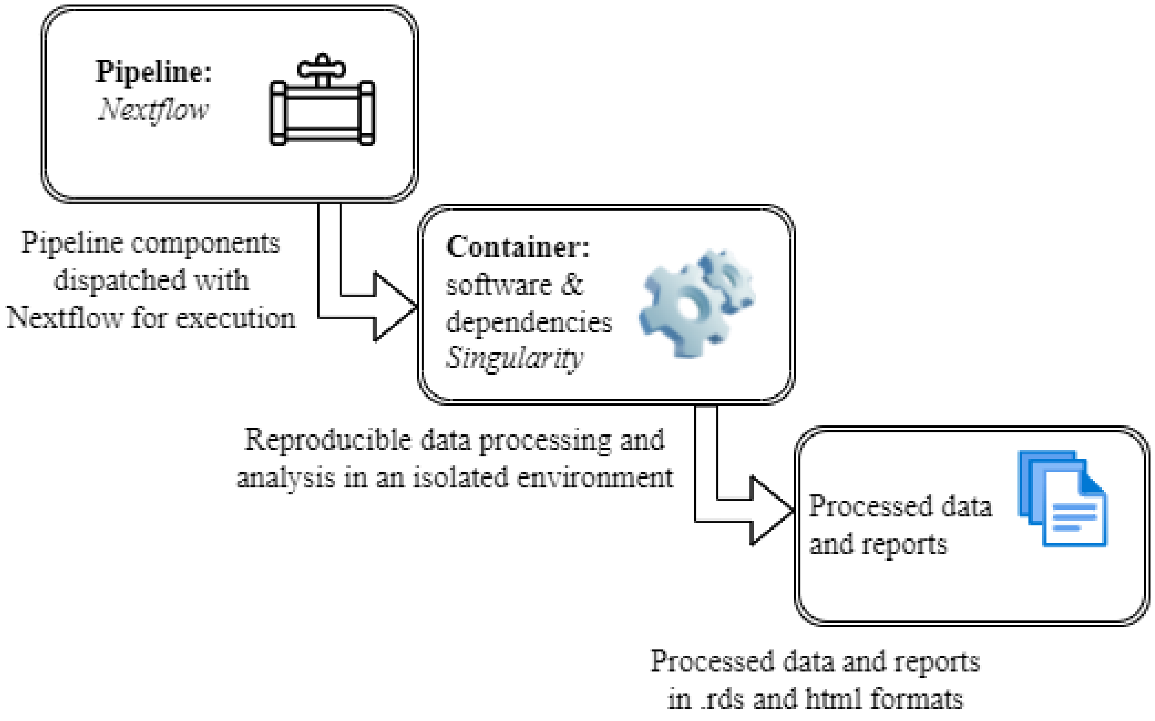

Pipeline:
Nextflow
Pipeline components dispatched with Nextflow for execution
Container:
software & dependencies
Singularity
Reproducible data processing and analysis in an isolated environment
Processed data and reports
Processed data and reports in .rds and html formats

Figure 18. Schematic flow of the LuminexPipeline execution. The pipeline is dispatched with the Nextflow workflow management system for execution in a Singularity container. The end of pipeline execution generates the pipeline outputs saved on the host computer.

4.5 Inclusion of statistical summary reports to the LuminexPipeline

One of the most important new functions to the LuminexPipeline that I developed was to program the pipeline to generate different types of statistical summary reports. Such reports would help the investigators to assess the quality of the multiplex ELISA data quickly and plan future experiments.

4.5.1 Distribution of analytes

One useful summary report would be to see the distribution of the concentrations of the analytes studied by multiplex ELISA, In Table 7, I give an example of the pipeline outputs as part of the statistical summary report, after the pipeline's execution. Here, one can get information on the range of each analyte's distribution represented with the min (minimum) and max (maximum) columns. Information on the central tendency, the representative value of the distribution, can be assessed with the mean and the q2 (median) labelled columns. Measures of variability or spread of each analyte's concentrations can be observed with the quartile columns, q1 (first quartile), q2 (second quartile), and q3 (third quartile). Missingness can be assessed with the NAs (number of missing values), available_obs (available observations) and the total_obs (total observations) columns.

Table 6. Statistical summary of seven analytes generated from the LuminexPipeline package. From this output, information regarding spread, central tendency and missingness can be assessed

							Observations (n)		
analyte	min	max	mean	q1	q2	q3	NAs	Available	Total
CCL1/I-309	62	170,579	3,539	843	1,922	3,792	2	576	578
ECM1	55	6,316	1,863	1,239	1,728	2,317	4	574	578
Ferritin	25	179,740	19,688	4,275	8,230	19,149	0	578	578
Fibrinogen	15	5,984,100	2,015,433	1,580,850	1,898,650	2,280,525	4	574	578
Haptoglobin	346	25,811	7361	4,589	7,036	9,636	4	574	578
IL-10	13	27,088	3620	1,364	2,777	4,744	1	576	577

The distribution of analyte concentrations can also be visualised in a histogram (Figure 19). The histograms are superimposed with a density function to describe the shape of the distribution of analyte concentrations and to also inform normality. Each analyte is represented in a faceted plot. The y-axis is the probability density function whereas the x-axis is the log-transformed analyte concentrations.

The R Shiny app allows for visualising the distribution after a power transformation (box-cox transformation).

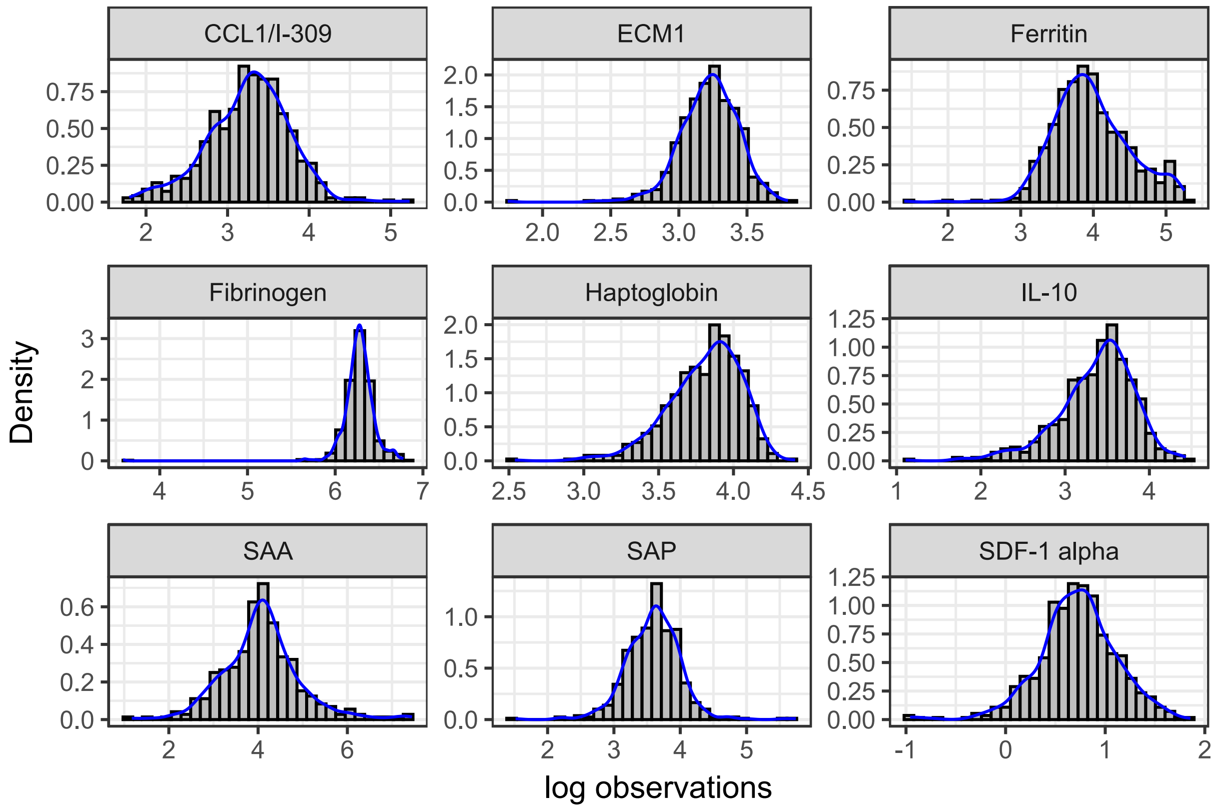


Figure 19: Distribution of analytes concentrations in a histogram

4.5.2 Reporting outliers

Another feature that I added to the LuminexPipeline is to have it generate reports on the different outlier detection methods as a percentage of the number of outliers detected (Table 8). The analyte column represents all the analytes, and the modified z-score outlier detection method is reported in the *mod_z(%)* column. The z_score detection method is represented in the *z_score(%)* column. Tukey's method of outlier detection is represented in the *Tukey(%)* column and the total number of observations are reported in the n column.

Table 7. Report on the different outlier detection methods

analyte	mod_z(%)	z_score(%)	Tukey(%)	n
CCL1/I-309	6.944	0.868	0.000	576
ECM1	1.394	1.394	0.174	574
Ferritin	14.533	3.979	0.173	578

Fibrinogen	4.355	2.613	0.697	574
Haptoglobin	0.174	0.523	0.174	574
IL-10	3.125	1.910	0.174	576
SAA	14.458	1.205	0.904	332

4.5.3 Correlation between different analytes

I also considered it important to add to the LuminexPipeline a function to report the relationships using Spearman correlation between analytes in the statistical summary report (Table 9). Here, column names, *analyte_1* and *analyte_2* are the paired analytes whose relationships are being computed. The Spearman correlation coefficient of each of the relationships is displayed in the column labelled ρ. The p-value of the relationship is presented in the p.value column.

Table 8. Correlation between analytes in a flattened correlation matrix

analyte_1	**analyte_2**	**ρ**	**p.value**
SAA	Haptoglobin	0.419	9.786e-12
IL-10	Ferritin	0.327	1.754e-14
Fibrinogen	Ferritin	0.320	5.684e-14
SDF-1 alpha	CCL1/I-309	0.303	1.082e-10
Fibrinogen	Haptoglobin	0.299	2.802e-13
SAP	Haptoglobin	0.283	7.229e-11
Fibrinogen	IL-10	0.276	2.383e-11

Visualisation of the analyte relationships was done in a correlogram (Figure 20). Figure 20 illustrates the upper triangle of the correlation matrix. The deep blue diagonal dots represent self-correlation between the same analytes. Blue dots are indicative of a positive association, whereas the light brown dots represent negative correlations. The strengths of the associations are reflective of the sizes of the dots (both blue and light brown). The larger the dots, the stronger the association and vice versa. Furthermore, the intensity of the colour of the dots also reflects the strength of the association, with deep-coloured dots representing a strong association and vice versa. The ladder on the far right of the correlogram is a legend mapping the colour of the dots to the correlation coefficients representing the strength of the association.

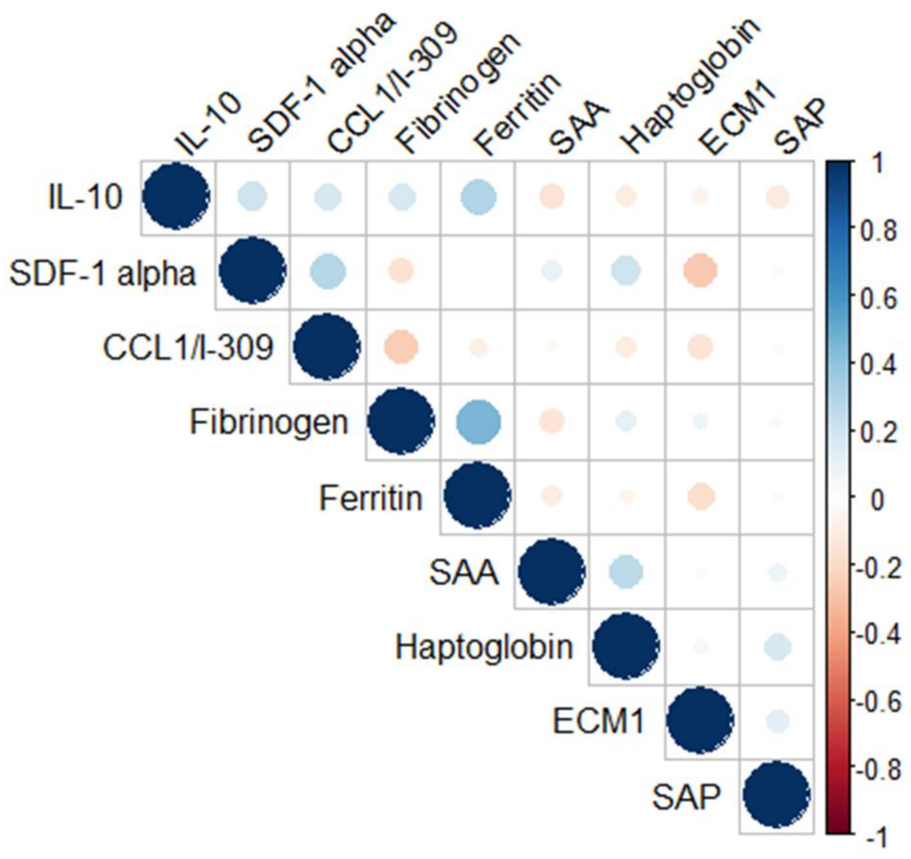


Figure 20: Visualisation of all associations in a correlogram. For details, see text.

4.5.4 Test of deviation from normality

4.5.4.1 Numerical methods

I included testing for normality in the statistical report summaries as another new feature in the LuminexPipeline (Table 10). Here, the specific analytes being tested for non-normality are in the *analyte* column. The excess kurtosis is reported in the *kurtosis* column, and the skewness of each analyte's distribution is reported in the *skewness* column, The test statistics, W', of the Shapiro-Francia test, are reported in the *Shapiro.Francia* column. Finally, the p-values associated with the Shapiro-Francia test are reported in the *p.value* column.

Table 9. Report of test of non-normality with the tests of excess kurtosis, skewness and the Shapiro-Francia test with its associated p-value

analyte	**kurtosis**	**skewness**	**Shapiro.Francia**	**p_value**
CCL1/I-309	230.873	13.616	0.239	2.6e-37
ECM1	2.113	1.132	0.937	3.1e-13
Ferritin	7.810	2.785	2.599	6.4e-30
Fibrinogen	4.793	1.686	0.876	1.8e-18
Haptoglobin	0.779	0.704	0.968	5.8e-09
IL-10	11.066	2.697	0.763	2.1e-24
SAA	109.604	10.115	0.108	2.9e-31

4.5.4.2 Graphical methods (q-q plots)

I also incorporated generation of q-q plots to the LuminexPipeline to further assess normality of the data. The results of nine analytes in a q-q plot are displayed in Figure 21. The quantiles of the observed concentrations of analytes are represented on the y-axis while the x-axis represents the quantiles of a theoretical normal distribution.

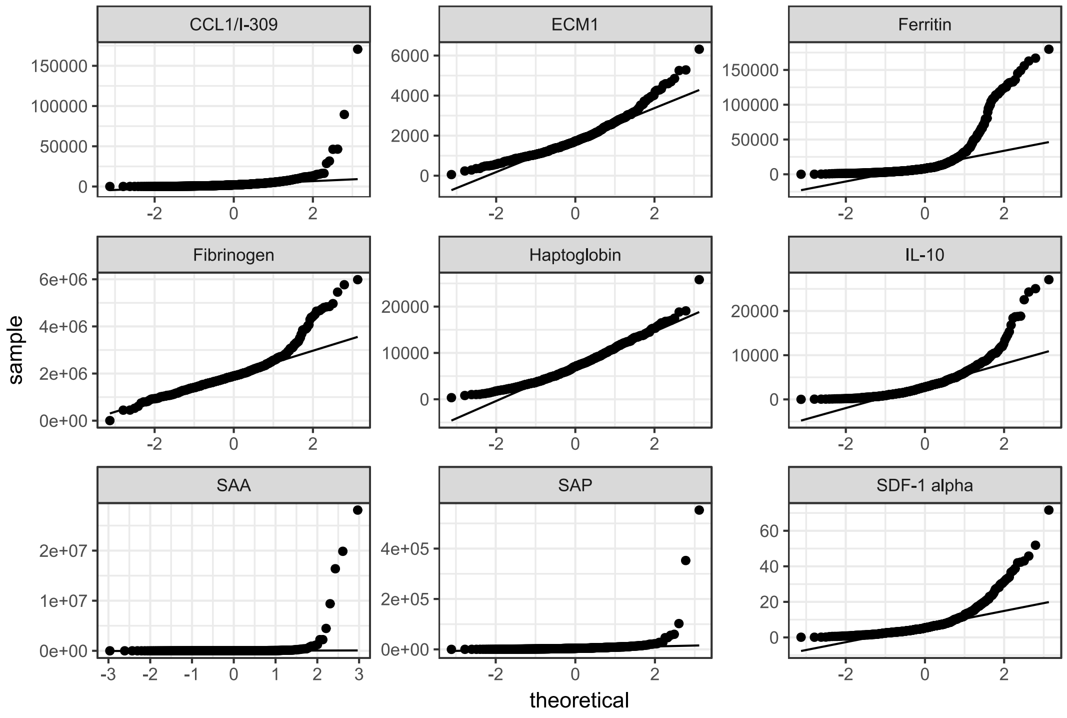


Figure 21: A faceted q-q plot of nine analytes to visually assess non-normality. The image is output as part of the Luminex pipeline's statistical summary report.

4.5.5 Comparison between study groups

Another useful feature in an analysis pipeline would be a preliminary analysis of comparing the results of different study groups. To achieve this, I implemented a function in the LuminexPipeline to generate a table of results after testing for differences between different study groups (e.g., treatment vs control). The results of this comparison are shown in the statistical summary report (Table 11). The analytes being compared are displayed in the column labelled *analyte*. The p-value of the Wilcoxon-rank sum test is given in the *p.value* labelled column and the test statistic of the Wilcoxon-rank sum test is given in the column labelled *w.statistic*.

Table 10. An output table of the Wilcoxon rank sum test between treated and untreated groups, generated from the LuminexPipeline package

analyte	p.value	w.statistic
CCL1/I-309	0.382	43,201
ECM1	0.327	39,214
Ferritin	0.142	44,689
Fibrinogen	0.458	39,693
Haptoglobin	0.595	42,217
IL-10	0.658	42,338
SAA	0.242	14,750

4.5.6 Shiny app

Another improvement I implemented for the statistical analyses in the LuminexPipeline included using the R Shiny LuminexApp. This function is useful for observing the distributions of the different analytes, determining the relationships between different analytes, and picking up general patterns in the data for further downstream analysis. The current version of the ShinyApp allows one to upload processed Luminex data (data output of pipeline) and explore certain statistical summaries: correlation (Pearson and Spearman), effects of Box-Cox and log transformation on analyte concentration as well as some visualisation (box plot, q-q plot and histograms). For training purposes, there is existing test data to help explore the functionality of the app and learn to use it effectively.

The current version of the Luminex ShinyApp is hosted on shinyapps.io and can be assessed with the link: https://asimeng.shinyapps.io/luminex_app/. The application code is also freely available on GitHub: https://github.com/Asimeng/LuminexPipeline_shinyApp .

CHAPTER 5: DISCUSSION

5.1 The LuminexPipeline and reproducibility

There has been a tremendous uptake of interest recently in making scientific methods and processes more open, rigorous, and reproducible. This is in response to addressing the current reproducibility crisis in science (Ioannidis, 2005; Baker, 2016) which, if not addressed may have grave implications on scientific research translation into mainstream use and may even dent public trust in science. The reproducibility crisis has necessitated an unprecedented demand for rigour, transparency, verification and reproducibility of scientific methods and results which are being enforced by all stakeholders of science (e.g., funders, publishers, researchers, and research institutions). For example, many funders now require that all research data, code and other tools are made publicly available to facilitate reproducibility, transparency, rigour and verification of results and to expedite scientific discovery (Committee on Realizing Opportunities for Advanced and Automated Workflows in Scientific Research, 2022).

In the wave of the current reproducibility crisis, which is also prominent in the Luminex data processing and analyses, my MSc work has contributed to attaining reproducible methods and results in this domain. I have extended the robustness, reproducibility and utility of the LuminexPipeline, an R-based pipeline, which researchers can use to improve the reproducibility of Luminex data processing and analyses. Reproducibility in scientific computing demands more than just sharing data and code. In this regard, first, the LuminexPipeline's implementation was extended using an automated WMS to automate all data processing and analytical steps in the pipeline. This is an important utility to help achieve consistency and attain reproducibility of analytical results by minimising human intervention, a major source of variation and errors in multiple-step data processing and analyses. The use of an automated WMS also helps to improve reproducibility through the ability to keep records of provenance information (all data processing steps) to allow analytical results to be traced back to their specific analytical pipeline. Second, I extended the reusability of code by incorporating and compiling newly written R functions together with existing functions into an R package (LuminexPipeline package). Reusability of code can greatly improve reproducibility, especially in the context where repetitive tasks are applied to different data sets. Third, I improved the robustness of the pipeline through automated unit testing. Unit testing is essential to improve the code structure and to efficiently handle errors (useful error messages) for ease of use. Fourth, I extended the pipeline's reproducibility by developing an isolated runtime environment (containerised environment) for the pipeline's execution. The use of containerised environments can greatly improve numerical stability during data processing and analyses by evading the impacts of variation in software versions and computational environments. For example, a common situation is that one's code does not work on a different computer. Containerisation also significantly promotes reproducibility by facilitating the publishing and sharing of analytical runtime environments, code or data alongside research findings.

Aside from attaining reproducibility of Luminex data processing and analyses, the pipeline development and extension procedures aimed at developing a robust utility, a program that runs on computers not owned by its developer and that can easily be used by other individuals other than the developer (Taschuk & Wilson, 2017). My work, therefore, utilised current best practices to develop a robust and reproducible pipeline in line with most of the (Taschuk & Wilson, 2017) ten rules for developing robust research software. Thus, when implementing the new functions to the pipeline I addressed the following points 1) there is version control during the development process with Git and Github; 2) good documentation practices (utilised Roxygen notes for package documentation); 3) unit testing and inclusion of test data set for users' exploration; 4) avoiding hard-coding file paths in the code (utilised configuration files with parameters); 5) avoiding root privileges for installation (the pipeline can be installed and run without root privileges); and 6) and ease of installation (pipeline can easily be setup and dispatched with just a single line of code on the command line).

5.2 Statistical summary extension

One of the important enhancements that I implemented in the LuminexPipeline was to include statistical summaries. Statistical summaries are important to provide general information about variables in the data and to examine relationships between variables. They are also useful for exploratory analyses, so that one can have a general idea of the spread and variation and the symmetry of the data by simply looking at the minimum value, the median and the maximum value.

Quartiles are three boundary points that divide a set of ordered data points into four equal parts. That is, the first quartile demarcates the first 25% of the data points. Thus, a 1st quartile value of

45 indicates that 25% of the data points are less than or equal to 45. Half (50%) of the data points are demarcated by the second quartile, which is also the median value. The 3rd quartile, also the 75% percentile indicates the third quarter of the data points. Assessing the quartiles can give a good indication of the spread or dispersion in the data. Additionally, the minimum and maximum values are good indications of the spread. From these values, the range can be computed as the maximum minus the minimum values.

Outliers are extreme data points or values that can uncover measurement errors especially when the observed data points are expected to be in a specific range. On the other hand, they can be a revelation of very interesting results. One of the methods of dealing with outliers is trimming and removing them. Trimming is usually done when the observed data are not in the expected range, and it is not possible to collect additional data as a replacement. When one cannot ascertain that observed values are a result of measurement errors, it is usually not recommended to remove outlying observations (Salgado et al., 2016). One of the statistical methods for dealing with such situations is winsorisation (Dixon & Yuen, 1974) where extreme values are pulled towards the centre of the data while maintaining the order of the data points. For example, the largest extreme value remains the largest value after winsorisation and the second largest value remains the second largest value, in that order.

q-q plots are graphical methods for assessing non-normality. When the data points of a q-q plot are evenly aligned in a straight diagonal line, the interpretation is that the residuals of the input data are normally distributed, see figure 22. A q-q plot may also reveal skewness in the data when the scatter points are curved instead of aligning on a straight line. It is noteworthy that, q-q plots are subjective means of assessment of non-normality and therefore, their interpretation should be subjected to expert opinion. Despite their subjective interpretation, q-q plots can prove useful when numerical methods of assessment of non-normality are oversensitive or under sensitive (Mishra et al., 2019).

Numerical alternatives for assessing non-normality are, for example, the Shapiro-Wilk, Shapiro-Francia and Kolmogorov-Smirnoff tests (Arsenault, 2020). The Shapiro-Wilk test, for example, has been shown to have similar overall power in comparison to the Shapiro-Francia test (Royston, 1993). While both have approximately the same power, the Shapiro-Francia test is relatively easier to compute because it only requires the expected ordered normal scores for its computation rather than certain special coefficients used by the Shapiro-Wilk test (Royston, 1983). Both the Shapiro-Wilk and Shapiro-Francia test can compute their test statistic and respective p-values on samples of similar sizes in R (3-5000 vs 5-5000). I, therefore, use the Shapiro-Francia test for the numerical tests of non-normality because of its easy computation and power. However, unlike the Kolmogorov-Smirnoff test, the Shapiro-Francia and Shapiro-Wilk tests compare sample data to only a standard distribution and do not allow one to compare two samples (Arsenault, 2020).

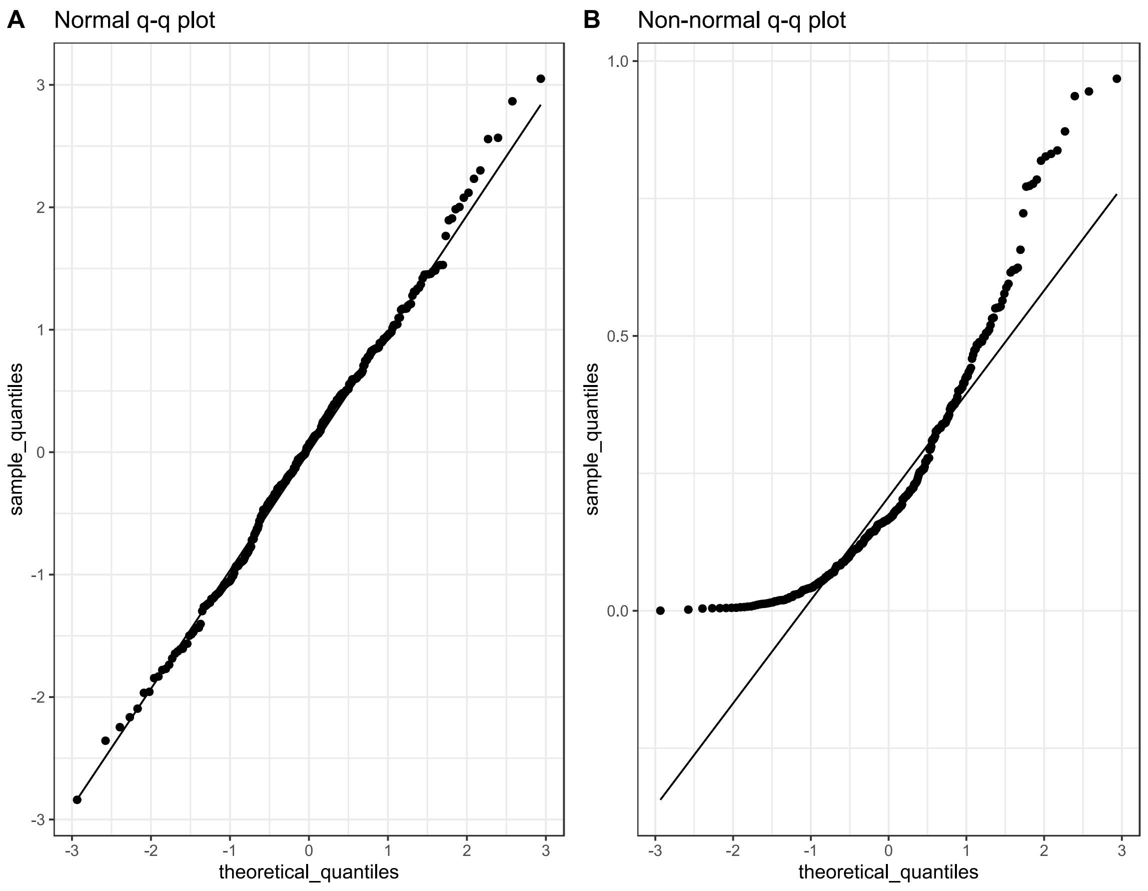


Figure 22: An example of a fairly normal distribution of residuals on a q-q plot (A). Plot (B) is an example of a non-normally distributed residuals on a q-q plot

Since we cannot determine a priori, the underlying distribution of analytes, the two-sample t-test may not be appropriate for hypotheses testing because it assumes normality of the residuals of the underlying distributions (which may not always be the case). The Wilcoxon rank sum test, on the other hand, is a rank-based and non-parametric alternative to the two-sample t-test, which makes no assumptions on the distribution of the underlying data. Its advantage is that it is less sensitive to outliers compared to the two-sample t-test (Wild & Seber, 1999) and, it is also asymptotically similar to the t-test and therefore has a similar statistical power. However, the power of the Wilcoxon rank sum test is lowered when there are ties in the data. The Interpretation of the Wilcoxon rank sum test statistic and p-value centres on whether the two samples being tested were drawn from the same distributions. This test does not test for the median difference between the two samples unless the two samples are unimodal or skewed in the same direction (Wild & Seber, 1999).

I computed only the Spearman correlation between analytes in my statistical summary report because, unlike the Pearson correlation, which assumes the distribution of analytes to be normally distributed (which is usually not the case for certain analytes), the Spearman correlation test makes no such assumptions and uses a rank-based approach in computing its coefficients. This is a generally robust approach to computing associations in both continuous and ordinal data (Pearson correlation tests applies to only continuous data). Additionally, the coefficient of Pearson correlation is a measure of only the strength of the linear relationship between two variables whereas the Spearman correlation coefficient is a measure of the monotonic relationship between two variables. Thus, the rate of increase or decrease in the relationship is not necessarily always constant as is assumed for the linear relationship measured by the Pearson correlation (Ramzai, 2021). However, in the complementary Shiny app (for exploratory analysis), one can compute both the Pearson and Spearman correlation. During the interpretation of correlation results, it is important to note that correlation does not necessarily imply causation (Rohrer, 2018).

Since Luminex data have an inherently large variance, one of the methods of dealing with this challenge is transformation. Transformation can draw extreme data points closer to the centre, significantly reduce noise and transform non-normally distributed data to be somewhat normal. However, the performance of different transformation methods may vary. I, therefore, provide the utility to explore the effects of transforming analyte distributions in the Shiny app. This utility involves the use of the log (Feng et al., 2014) or Box-Cox transformation (Sakia, 1992) to visualise the transformation effects of analyte distributions in a q-q plot, histogram, or box plot. However, this utility should be used for only exploratory purposes because of the risk of overfitting in downstream analysis.

5.3 Containerisation

There are different types of software to build containerised environments. Docker (Rad, Bhatti & Ahmadi, 2017) is the most widely used. However, executing Docker containers requires root privileges which is a major challenge for use on shared computing systems like HPC. Docker is, therefore, discouraged from being used on HPC environments, because of the risk of compromising other users' data with root privileges. To run containerised workflows on the HPC, a suitable alternative, Singularity (Kurtzer, Sochat & Bauer, 2017) is the popular go-to option. Additionally, Singularity can convert and utilise Docker images. I, therefore, developed the containerised LuminexPipeline with Singularity, which solves to accommodate execution on the HPC.

5.4 Quality control

I implemented quality control (fitting standard curves) by incorporating functions from the drLumi R package (Sanz et al., 2017) into the LuminexPipeline R package. Curve fitting is a utility for estimating unknown sample concentrations from the reported fluorescent intensities. Although most Luminex instruments automatically estimate unknown sample concentrations, having the utility to generate standard curves can prove useful as a quality assessment feature for detecting probable errors (usually pipetting) by observing the shape of the logistic growth curve (standard curve). Alternatively, depending on a specific analyte or experiment, a researcher may want to use a specific curve fitting method which may not be provided by the automated curve fitting software of the Luminex system. For example, the drLumi R package provides many alternatives for choosing the limits of quantitation on a standard curve. A specific experiment may be best suited for implementation by utilising a specific method of estimation of limits of quantitation which may not be available in the Luminex automated curve fitting software. Additionally, these are open-source tools which promote open science and remove layers of a black box.

5.5 Limitations

5.5.1 Test coverage and documentation

The coverage of unit testing and package documentation focused only on the major functions involved in importing and processing Luminex. Future work should extend the coverage of unit testing and package documentation to other minor functions in the pipeline.

5.5.2 Automating multi-stage container build process

One of the limitations of the methods described here was the use of a manual approach to the multi-stage build process to trim container size. Unlike an automated approach (script) for recording all shared library files and program files, the manual approach I utilised, may not be reproducible in other computing environments. Future work should, therefore, consider automating this process. However, in the absence of an automated script, I provide an alternative definition file which utilises the conventional container build process to achieve reproducibility but comes at the expense of having a relatively larger container size (see https://github.com/Asimeng/LuminexPipeline_container or the Appendix section for the definition or the recipe file).

CHAPTER 6: CONCLUSIONS

Reproducibility is one of the fundamental principles for doing good science. However, this tenet of science has been subjected to a serious crisis in which many published scientific findings are not reproducible. The implications of this crisis can be wrong scientific findings and conclusions which can have devastating impact when translated to the industry. This crisis also poses the risk of losing public trust in science. Some of the major contributors to the current reproducibility crisis have been identified to be cognitive bias, p-hacking, misuse of p-values, underpowered studies, publication bias and preference for novelty culture. However, in the face of these challenges, there has been a tremendous uptake of interest recently by many stakeholders of science including researchers themselves, funders, research institutions, and publishing firms, to help remedy the current reproducibility crisis. Efforts are being put in place, for example, by funders to make the scientific process more open for scrutiny and verification by requiring researchers to make their research data publicly available. Publishing institutions are developing new approaches to facilitate the publishing of negative results through pre-registration of research. Many universities are incorporating concepts of reproducibility into their curricula, and there is increased scrutiny in published results demonstrated by the increased retraction of papers (https://retractionwatch.com/).

Given the current reproducibility challenges, I present the LuminexPipeline, my contribution to help significantly improve reproducibility and robustness, specifically with the processing and analyses of multiplex ELISA data generated using the Luminex platform. The LuminexPipeline improves analytical reproducibility by utilising an automated workflow management system, a containerised workflow execution, extended robustness in utility through unit testing, and a statistical summary report to extend the overall utility of the pipeline. Furthermore, I improved reproducibility by reusing code through reusable functions in an R package applicable to all Luminex assay data.

Through my MSc research project, I have contributed to open science by making the LuminexPipeline and its related utilities freely available for use. Additionally, they are licensed with open-source licenses which allow for unrestricted use and further improvement by third-party organisations or individuals.

As part of community efforts to improve reproducibility through the FAIR guiding principles, this contribution, the development of the LuminexPipeline will help to generate, with efficiency and speed, standard compliant and ready-to-share datasets (cleaned and curated) for indexing and upload onto online data repositories (e.g., Zenodo) facilitating the findability and accessibility components of the FAIR guiding principles. The pipeline's output data file format (.*rds* file) is versatile (machine readable) and can be read by many programming languages (e.g., Python) and software packages, attaining the interoperable component of the FAIR guiding principles. Finally, given that the drLumi R package has been archived on CRAN as of April 2020, the submission of the R LuminexPipeline package to CRAN, will make it the only available package on CRAN for Multiplexed ELISA data processing and analysis with continuous support.

REFERENCES

Afgan, E., Baker, D., Batut, B., van den Beek, M., Bouvier, D., Čech, M., Chilton, J., Clements, D., et al. 2018. The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update. *Nucleic Acids Research*. 46(Web Server issue):W537–W544. DOI: 10.1093/nar/gky379.

Allaire, J.J., Xie, Y., McPherson, J., Luraschi, J., Ushey, K., Atkins, A., Wickham, H., Cheng, J., et al. 2022. Available: https://CRAN.R-project.org/package=rmarkdown [2022, November 14].

Amstutz, P., Crusoe, M.R., Nebojša Tijanić, Chapman, B., Chilton, J., Heuer, M., Kartashov, A., Leehr, D., et al. 2016. DOI: 10.6084/M9.FIGSHARE.3115156.V2.

Andrews, S., Krueger, F. & Segonds-Pichon, A. 2020. Available: http://www.bioinformatics.babraham.ac.uk/projects/fastqc/.

Arsenault, M.-O. 2020. *Kolmogorov–Smirnov test*. Available: https://towardsdatascience.com/kolmogorov-smirnov-test-84c92fb4158d [2022, November 02].

Arunika, D. 2016. Reproducibility in science. Available: https://www.ascb.org/careers/reproducibility-in-science/ [2021, December 24].

ASCB Task Force. 2014. ASCB task force report on reproducibility in science. Available: https://www.ascb.org/science-policy-public-outreach/advocacy-policy/ascb-examines-difficulty-reproducing-research-data/ [2022, November 02].

Baker, M. 2016. 1,500 scientists lift the lid on reproducibility. *Nature*. 533(7604):452–454. DOI: 10.1038/533452a.

Baker, K.S., Suu-Ire, R., Barr, J., Hayman, D.T.S., Broder, C.C., Horton, D.L., Durrant, C., Murcia, P.R., et al. 2014. Viral antibody dynamics in a chiropteran host. *The Journal of Animal Ecology*. 83(2):415. DOI: 10.1111/1365-2656.12153.

Begley, C.G. & Ellis, L.M. 2012. Raise standards for preclinical cancer research. *Nature*. 483(7391):531–533. DOI: 10.1038/483531a.

Berkowitz, C. 2014. *Pumped up*. Available: https://www.sciencehistory.org/distillations/pumped-up [2021, December 22].

Bio-Rad. n.d. *Luminex xMAP technology*. Available: https://www.bio-rad.com/featured/en/luminex-xmap-technology.html [2022, April 04].

Bishop, D. 2019. Rein in the four horsemen of irreproducibility. *Nature*. 568(7753):435–435. DOI: 10.1038/d41586-019-01307-2.

Brendan Bioanalytics. n.d. Weighting in logistic curve regressions. Available: https://www.brendan.com/curve-weighting/ [2022, July 08].

Carl, P., Ramos, I.I., Segundo, M.A. & Schneider, R.J. 2019. Antibody conjugation to carboxyl-modified microspheres through N-hydroxysuccinimide chemistry for automated immunoassay applications: a general procedure. *PLoS ONE*. 14(6). DOI: 10.1371/journal.pone.0218686.

Casadevall, A. & Fang, F.C. 2010. Reproducible science. *Infection and Immunity*. 78(12):4972–4975. DOI: 10.1128/IAI.00908-10.

Chang, W., Cheng, J., Allaire, J.J., Sievert, C., Schloerke, B., Xie, Y., Allen, J., McPherson, J., et al. 2022. Available: https://CRAN.R-project.org/package=shiny [2022, October 31].

Chawla, D.S. 2017. P-value shake-up proposed. *Nature*. 548:16–17.

Clay Ford. 2015. *Understanding Q-Q plots*. Available: https://data.library.virginia.edu/understanding-q-q-plots/ [2022, September 28].

Committee on Realizing Opportunities for Advanced and Automated Workflows in Scientific Research. 2022. *Automated research workflows for accelerated discovery: closing the knowledge discovery loop*. Washington, D.C.: National Academies Press. DOI: 10.17226/26532.

Committee on Reproducibility and Replicability in Science, the National Academies of Science, Engineering and Medicine. 2019a. Improving reproducibility and replicability. In *Reproducibility and replicability in science*. Washington, D.C.: National Academies Press (US). DOI: https://doi.org/10.17226/25303.

Committee on Reproducibility and Replicability in Science, the National Academies of Science, Engineering and Medicine. 2019b. *Reproducibility and Replicability in Science*. Washington, D.C.: National Academies Press. DOI: 10.17226/25303.

Committee on Reproducibility and Replicability in Science, the National Academies of Science, Engineering and Medicine. 2019c. Confidence in science. In *Reproducibility and replicability in science*. Washington, D.C.: National Academies Press (US). DOI: https://doi.org/10.17226/25303.

Cooper, G.S. & Meterko, V. 2019. Cognitive bias research in forensic science: a systematic review. *Forensic Science International*. 297:35–46. DOI: 10.1016/j.forsciint.2019.01.016.

Cox, K.L., Devanarayan, V., Kriauciunas, A., Montrose, C. & Sittampalam, S. 2019. Immunoassay methods. (July, 8):39. Available: https://www.ncbi.nlm.nih.gov/books/NBK92434/.

Cumberland, W.N., Fong, Y., Yu, X., Defawe, O., Frahm, N. & De Rosa, S. 2015. Nonlinear calibration model choice between the four and five-parameter logistic models. *Journal of Biopharmaceutical Statistics*. 25(5):972–983. DOI: 10.1080/10543406.2014.920345.

Di Tommaso, P., Chatzou, M., Floden, E.W., Barja, P.P., Palumbo, E. & Notredame, C. 2017. Nextflow enables reproducible computational workflows. *Nature Biotechnology*. 35(4):316–319. DOI: 10.1038/nbt.3820.

Dixon, W.J. & Yuen, K.K. 1974. Trimming and winsorization: a review. *Statistische Hefte*. 15(2):157–170. DOI: 10.1007/BF02922904.

Dudley, R.A., Edwards, P., Ekins, R.P., Finney, D.J., McKenzie, I.G., Raab, G.M., Rodbard, D. & Rodgers, R.P. 1985. Guidelines for immunoassay data processing. *Clinical Chemistry*. 31(8):1264–1271. DOI: 10.1093/clinchem/31.8.1264.

Dunbar, S.A. 2006. Applications of Luminex® xMAP^{TM} technology for rapid, high-throughput multiplexed nucleic acid detection. *Clinica Chimica Acta; International Journal of Clinical Chemistry*. 363(1):71–82. DOI: 10.1016/j.cccn.2005.06.023.

Dunbar, S. & Li, D. 2010. Introduction to Luminex® xMAP® technology and applications for biological analysis in China. *Asia Pacific Biotech*. 14:26–30.

Dunbar, S.A. & Hoffmeyer, M.R. 2013. Chapter 2.9 - Microsphere-Based Multiplex Immunoassays: Development and Applications Using Luminex® xMAP® Technology. In *The Immunoassay Handbook (Fourth Edition)*. D. Wild, Ed. Oxford: Elsevier. 157–174. DOI: 10.1016/B978-0-08-097037-0.00012-9.

Dunn, J. & Wild, D. 2013. Chapter 3.6 - calibration curve fitting. In *The Immunoassay Handbook (Fourth Edition)*. D. Wild, Ed. Oxford: Elsevier. 323–336. DOI: 10.1016/B978-0-08-097037-0.00022-1.

Eckels, J., Nathe, C., Nelson, E.K., Shoemaker, S.G., Nostrand, E.V., Yates, N.L., Ashley, V.C., Harris, L.J., et al. 2013. Quality control, analysis and secure sharing of Luminex® immunoassay data using the open source LabKey Server platform. *BMC Bioinformatics*. 14(1):145. DOI: 10.1186/1471-2105-14-145.

Engvall, E. & Perlmann, P. 1971. Enzyme-linked immunosorbent assay (ELISA). Quantitative assay of immunoglobulin G. *Immunochemistry*. 8(9):871–874. DOI: 10.1016/0019-2791(71)90454-x.

Feng, C., Wang, H., Lu, N., Chen, T., He, H., Lu, Y. & Tu, X.M. 2014. Log-transformation and its implications for data analysis. 26(2):5.

Fong, Y., Sebestyen, K., Yu, X., Gilbert, P. & Self, S. 2013. nCal: an R package for non-linear calibration. *Bioinformatics*. 29(20):2653–2654. DOI: 10.1093/bioinformatics/btt456.

Franceschi, P., Mylonas, R., Shahaf, N., Scholz, M., Arapitsas, P., Masuero, D., Weingart, G., Carlin, S., et al. 2014. MetaDB a data processing workflow in untargeted MS-based metabolomics experiments. *Frontiers in Bioengineering and Biotechnology*. 2. DOI: 10.3389/fbioe.2014.00072.

González-Beltrán, A., Li, P., Zhao, J., Avila-Garcia, M.S., Roos, M., Thompson, M., van der Horst, E., Kaliyaperumal, R., et al. 2015. From peer-reviewed to peer-reproduced in scholarly publishing: the complementary roles of data models and workflows in bioinformatics. *PLOS ONE*. 10(7):e0127612. DOI: 10.1371/journal.pone.0127612.

Goodman, S.N. 2001. Of p-values and Bayes: a modest proposal. *Epidemiology*. 12(3):295–297. Available: https://journals.lww.com/epidem/Fulltext/2001/05000/Of_P_Values_and_Bayes__A_Modest_Proposal.6.aspx [2022, October 30].

Gottschalk, P.G. & Dunn, J.R. 2005a. The five-parameter logistic: a characterization and comparison with the four-parameter logistic. *Analytical Biochemistry*. 343(1):54–65. DOI: 10.1016/j.ab.2005.04.035.

Gottschalk, P.G. & Dunn, J.R. 2005b. Measuring parallelism, linearity, and relative potency in bioassay and immunoassay data. *Journal of Biopharmaceutical Statistics*. 15(3):437–463. DOI: 10.1081/BIP-200056532.

Grubbs, F.E. 1969. Procedures for detecting outlying observations in samples. *Technometrics*. 11(1):1–21. DOI: 10.1080/00401706.1969.10490657.

Guardabasso, V., Rodbard, D. & Munson, P.J. 1987. A model-free approach to estimation of relative potency in dose-response curve analysis. *American Journal of Physiology-Endocrinology and Metabolism*. 252(3):E357–E364.

Gundersen, O.E. 2021. The fundamental principles of reproducibility. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*. 379(2197):20200210. DOI: 10.1098/rsta.2020.0210.

Hérisson, J., Duigou, T., du Lac, M., Bazi-Kabbaj, K., Sabeti Azad, M., Buldum, G., Telle, O., El Moubayed, Y., et al. 2022. The automated Galaxy-SynBioCAD pipeline for synthetic biology design and engineering. *Nature Communications*. 13(1):5082. DOI: 10.1038/s41467-022-32661-x.

Hoon, S., Ratnapu, K.K., Chia, J., Kumarasamy, B., Juguang, X., Clamp, M., Stabenau, A., Potter, S., et al. 2003. Biopipe: a flexible framework for protocol-based bioinformatics analysis. *Genome Research*. 13(8):1904–1915. DOI: 10.1101/gr.1363103.

Huang, Y. & Gottardo, R. 2013. Comparability and reproducibility of biomedical data. *Briefings in Bioinformatics*. 14(4):391–401. DOI: 10.1093/bib/bbs078.

Huls, M. 2022. *Using multi-stage builds to make your docker image 10x smaller*. Available: https://towardsdatascience.com/using-multi-stage-builds-to-make-your-docker-image-almost-10x-smaller-239068cb6fb0 [2022, November 01].

Hutson, M. 2018. *Missing data hinder replication of artificial intelligence studies*. Available: https://www.science.org/content/article/missing-data-hinder-replication-artificial-intelligence-studies [2022, August 22].

Ioannidis, J.P.A. 2005. Why most published research findings are false. *PLoS Medicine*. 2(8):e124. DOI: 10.1371/journal.pmed.0020124.

Ioannidis, J.P.A. 2014. How to make more published research true. *PLoS Medicine*. 11(10):e1001747. DOI: 10.1371/journal.pmed.1001747.

Ioannidis, J.P.A. 2018. The proposal to lower p value thresholds to .005. *JAMA*. 319(14):1429–1430. DOI: 10.1001/jama.2018.1536.

Jackson, M., Kavoussanakis, K. & Wallace, E.W.J. 2021. Using prototyping to choose a bioinformatics workflow management system. *PLOS Computational Biology*. 17(2):e1008622. DOI: 10.1371/journal.pcbi.1008622.

James Westby, Daniel Arteaga, Carlos Rodríguez Hernández, Alessandro Chitolina, Alejandro Ruiz, Joseda Rios, Beltran Rubo, John Kristensen, et al. 2022. Available: https://github.com/bitnami/minideb [2022, August 29].

Johnson, D., Cochrane, K., Davey, R.P., Etuk, A., Gonzalez-Beltran, A., Haug, K., Izzo, M., Larralde, M., et al. 2021. ISA API: an open platform for interoperable life science experimental metadata. *bioRxiv*. DOI: 10.1101/2020.11.13.382119.

Kerr, N.L. 1998. HARKing: hypothesizing after the results are known. *Personality and Social Psychology Review*. 2(3):196–217. DOI: 10.1207/s15327957pspr0203_4.

Kim, Y.-M., Poline, J.-B. & Dumas, G. 2018. Experimenting with reproducibility: a case study of robustness in bioinformatics. *GigaScience*. 7(7). DOI: 10.1093/gigascience/giy077.

Koster, J. & Rahmann, S. 2012. Snakemake--a scalable bioinformatics workflow engine. *Bioinformatics*. 28(19):2520–2522. DOI: 10.1093/bioinformatics/bts480.

Kreier, F. 2022. Orangutan genome mix-up muddies conservation efforts. *Nature*. 610(7933):617–618. DOI: 10.1038/d41586-022-03193-7.

Krueger, F., James, F., Ewels, P., Afyounian, E. & Schuster-Boeckler, B. 2021. DOI: 10.5281/zenodo.5127899.

Kurtzer, G.M., Sochat, V. & Bauer, M.W. 2017. Singularity: scientific containers for mobility of compute. *PLoS ONE*. 12(5):e0177459. DOI: 10.1371/journal.pone.0177459.

Landucci, F. & Lamperti, M. 2021. A pandemic of cognitive bias. *Intensive Care Medicine*. 47(5):636–637. DOI: 10.1007/s00134-020-06293-y.

Leipzig, J. 2017. A review of bioinformatic pipeline frameworks. *Briefings in Bioinformatics*. 18(3):530–536. DOI: 10.1093/bib/bbw020.

Lin, A.V. 2015a. Direct ELISA. In *ELISA*. V. 1318. R. Hnasko, Ed. (Methods in Molecular Biology). New York, NY: Springer New York. 61–67. DOI: 10.1007/978-1-4939-2742-5_6.

Lin, A.V. 2015b. Indirect ELISA. In *ELISA: Methods and Protocols*. R. Hnasko, Ed. (Methods in Molecular Biology). New York, NY: Springer. 51–59. DOI: 10.1007/978-1-4939-2742-5_5.

Merow, C. 2019. *Quickstart guide to R package building*. Available: https://cmerow.github.io/RDataScience/Quickstart_RPackages.html [2022, October 31].

Mishra, P., Pandey, C., Singh, U., Gupta, A., Sahu, C. & Keshri, A. 2019. Descriptive statistics and normality tests for statistical data. *Annals of Cardiac Anaesthesia*. 22(1):67. DOI: 10.4103/aca.ACA_157_18.

Mitra-Behura, S., Fiolka, R.P. & Daetwyler, S. 2022. Singularity containers improve reproducibility and ease of use in computational image analysis workflows. *Frontiers in Bioinformatics*. 1:757291. DOI: 10.3389/fbinf.2021.757291.

Mullard, A. 2021. Half of top cancer studies fail high-profile reproducibility effort. *Nature*. 600(7889):368–369. DOI: 10.1038/d41586-021-03691-0.

Munafò, M.R., Nosek, B.A., Bishop, D.V.M., Button, K.S., Chambers, C.D., Percie du Sert, N., Simonsohn, U., Wagenmakers, E.-J., et al. 2017. A manifesto for reproducible science. *Nature Human Behaviour*. 1(1):1–9. DOI: 10.1038/s41562-016-0021.

Nelson, E.K., Piehler, B., Eckels, J., Rauch, A., Bellew, M., Hussey, P., Ramsay, S., Nathe, C., et al. 2011. LabKey Server: an open source platform for scientific data integration, analysis and collaboration. *BMC Bioinformatics*. 12(1):71. DOI: 10.1186/1471-2105-12-71.

NIST/SEMATECH. 2012. Measures of skewness and kurtosis. In *NIST/SEMATECH e-handbook of statistical methods*. Available: https://doi.org/10.18434/M32189.

Nuzzo, R. 2014. Scientific method: statistical errors. *Nature*. 506(7487):150–152. DOI: 10.1038/506150a.

Nuzzo, R. 2015. How scientists fool themselves – and how they can stop. *Nature*. 526(7572):182–185. DOI: 10.1038/526182a.

Papin, J.A., Mac Gabhann, F., Sauro, H.M., Nickerson, D. & Rampadarath, A. 2020. Improving reproducibility in computational biology research. *PLOS Computational Biology*. 16(5):e1007881. DOI: 10.1371/journal.pcbi.1007881.

Plesser, H.E. 2018. Reproducibility vs. replicability: a brief history of a confused terminology. *Frontiers in Neuroinformatics*. 11:76. DOI: 10.3389/fninf.2017.00076.

Poldrack, R.A. 2022. *Statistical thinking for the 21st century*. Available: https://statsthinking21.github.io/statsthinking21-core-site/doing-reproducible-research.html [2022, October 15].

R Core Team. 2021. Available: https://www.R-project.org/.

Raab, G.M. 1983. Comparison of a logistic and a mass-action curve for radioimmunoassay data. *Clinical chemistry*. 29(10):1757–1761.

Rad, B.B., Bhatti, H.J. & Ahmadi, M. 2017. An introduction to docker and analysis of its performance. *International Journal of Computer Science and Network Security (IJCSNS)*. 17(3):228.

Ramzai, J. 2021. *Clearly explained: Pearson vs Spearman correlation coefficient*. Available: https://towardsdatascience.com/clearly-explained-pearson-v-s-spearman-correlation-coefficient-ada2f473b8 [2022, November 02].

Rattle, S., Hofmann, O., Price, C.P., Kricka, L.J. & Wild, D. 2013. Chapter 2.10 - lab-on-a-chip, micro- and nanoscale immunoassay systems, and microarrays. In *The Immunoassay Handbook (Fourth Edition)*. D. Wild, Ed. Oxford: Elsevier. 175–202. DOI: 10.1016/B978-0-08-097037-0.00013-0.

Rocca-Serra, P., Brandizi, M., Maguire, E., Sklyar, N., Taylor, C., Begley, K., Field, D., Harris, S., et al. 2010. ISA software suite: supporting standards-compliant experimental annotation and enabling curation at the community level. *Bioinformatics*. 26(18):2354–2356. DOI: 10.1093/bioinformatics/btq415.

Rocca-Serra, P., Maguire, E., Taylor, C., Field, D., Wittenberger, T., Santarsiero, A., Gonzalez-Beltran, A. & Sansone, S.-A. 2012. Investigation-study-assay, a toolkit for standardizing data capture and sharing. In *Open Source Software in Life Science Research*. Elsevier. 173–188. DOI: 10.1533/9781908818249.173.

Rohrer, J.M. 2018. Thinking clearly about correlations and causation: graphical causal models for observational data. *Advances in Methods and Practices in Psychological Science*. 1(1):27–42. DOI: 10.1177/2515245917745629.

Royston, J.P. 1983. A simple method for evaluating the Shapiro-Francia W' test of non-normality. *The Statistician*. 32(3):297. DOI: 10.2307/2987935.

Royston, P. 1993. A pocket-calculator algorithm for the shapiro-francia test for non-normality: An application to medicine. *Statistics in Medicine*. 12(2):181–184. DOI: 10.1002/sim.4780120209.

Sakia, R.M. 1992. The Box-Cox transformation technique: a review. *Journal of the Royal Statistical Society: Series D (The Statistician)*. 41(2):169–178.

Salgado, C.M., Azevedo, C., Proença, H. & Vieira, S.M. 2016. Noise versus outliers. In *Secondary Analysis of Electronic Health Records*. MIT Critical Data, Ed. Cham: Springer International Publishing. 163–183. DOI: 10.1007/978-3-319-43742-2_14.

Sanz, H., Aponte, J.J., Harezlak, J., Dong, Y., Ayestaran, A., Nhabomba, A., Mpina, M., Maurin, O.R., et al. 2017. drLumi: an open-source package to manage data, calibrate, and conduct quality control of multiplex bead-based immunoassays data analysis. *PLOS ONE*. 12(11):e0187901. DOI: 10.1371/journal.pone.0187901.

Seqera Labs. 2022. *Processes — Nextflow 22.04.0 documentation*. Available: https://www.nextflow.io/docs/latest/process.html [2022, September 10].

Shah, K. & Maghsoudlou, P. 2016. Enzyme-linked immunosorbent assay (ELISA): the basics. *British Journal of Hospital Medicine*. 77(7):C98–C101. DOI: 10.12968/hmed.2016.77.7.C98.

Sheehan, C., He, J. & Smith, M. 2013. Chapter 5.2 - method evaluation—a practical guide. In *The Immunoassay Handbook (Fourth Edition)*. D. Wild, Ed. Oxford: Elsevier. 395–402. DOI: 10.1016/B978-0-08-097037-0.00026-9.

Silberzahn, R., Uhlmann, E.L., Martin, D.P., Anselmi, P., Aust, F., Awtrey, E., Bahník, Š., Bai, F., et al. 2018. Many analysts, one data set: making transparent how variations in analytic choices affect results. *Advances in Methods and Practices in Psychological Science*. 1(3):337–356. DOI: 10.1177/2515245917747646.

Simmons, J.P., Nelson, L.D. & Simonsohn, U. 2011. False-positive psychology: undisclosed flexibility in data collection and analysis allows presenting anything as significant. *Psychological Science*. 22(11):1359–1366. DOI: 10.1177/0956797611417632.

Stark, P.B. 2018. Before reproducibility must come preproducibility. *Nature*. 557(7707):613. DOI: 10.1038/d41586-018-05256-0.

Taschuk, M. & Wilson, G. 2017. Ten simple rules for making research software more robust. *PLOS Computational Biology*. 13(4):e1005412. DOI: 10.1371/journal.pcbi.1005412.

The Turing Way Community. 2022. *The Turing way: a handbook for reproducible, ethical and collaborative research*. Available: https://the-turing-way.netlify.app/reproducible-research/reproducible-research.html.

Tighe, P.J., Ryder, R.R., Todd, I. & Fairclough, L.C. 2015. ELISA in the multiplex era: potentials and pitfalls. *PROTEOMICS – Clinical Applications*. 9(3–4):406–422. DOI: 10.1002/prca.201400130.

Tripepi, G., Jager, K.J., Dekker, F.W. & Zoccali, C. 2010. Selection bias and information bias in clinical research. *Nephron. Clinical Practice*. 115(2):c94-99. DOI: 10.1159/000312871.

Tversky, A. & Kahneman, D. 1974. Judgment under uncertainty: heuristics and biases. *Science*. 185(4157):1124–1131. DOI: 10.1126/science.185.4157.1124.

Van Weemen, B.K. & Schuurs, A.H.W.M. 1971. Immunoassay using antigen-enzyme conjugates. *FEBS letters*. 15(3):232–236. DOI: 10.1016/0014-5793(71)80319-8.

Wickham, H. 2011. testthat: get started with testing. *The R Journal*. 3(1):5. DOI: 10.32614/RJ-2011-002.

Wickham, H. 2015. *R packages*. O'Reilly Media, Inc. Available: https://r-pkgs.org/.

Wickham, H. 2021. *Mastering Shiny*. O'Reilly Media, Inc. Available: https://mastering-shiny.org/.

Wickham, H., Hester, J., Chang, W., Bryan, J. & RStudio. 2022. Available: https://CRAN.R-project.org/package=devtools [2022, October 31].

Wickham, H., Bryan, J., Barrett, M. & RStudio. 2022. Available: https://CRAN.R-project.org/package=usethis [2022, October 31].

Wickham, H., Danenberg, P., Csárdi, G., Eugster, M. & RStudio. 2022. Available: https://CRAN.R-project.org/package=roxygen2 [2022, October 31].

Wild, D. 2013. Chapter 1.2 - Immunoassay for beginners. In *The Immunoassay Handbook (Fourth Edition)*. D. Wild, Ed. Oxford: Elsevier. 7–10. DOI: 10.1016/B978-0-08-097037-0.00002-6.

Wild, C.J. & Seber, G.A.F. 1999. Chapter 10 supplements: the Wilcoxon test. In *Chance encounters: a first course in data analysis and inference*. Wiley. Available: https://www.stat.auckland.ac.nz/~wild/ChanceEnc/Ch10.wilcoxon.pdf.

Wilkinson, M.D., Dumontier, M., Aalbersberg, I.J.J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., et al. 2016. The FAIR guiding principles for scientific data management and stewardship. *Scientific Data*. 3:160018. DOI: 10.1038/sdata.2016.18.

Yalow, R.S. & Berson, S.A. 1960. Immunoassay of endogenous plasma insulin in man. *Journal of Clinical Investigation*. 39(7):1157–1175. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC441860/ [2022, October 23].

APPENDICES

Container definition file

```
Bootstrap: docker
From: bitnami/minideb:bullseye

%post

#install required utilities & dependencies necessary to add a new repo over
https (for installing latest R version)
  install_packages dirmngr gnupg apt-transport-https ca-certificates
software-properties-common wget

#Add the publicly singed CRAN repository key to the sources.list file
  apt-key adv --keyserver keyserver.ubuntu.com --recv-key
'95C0FAF38DB3CCAD0C080A7BDC78B2DDEABC47B7'

  add-apt-repository 'deb http://cloud.r-project.org/bin/linux/debian
bullseye-cran40/'

#install shared library dependencies for R packages
  install_packages libcurl4-openssl-dev libssl-dev libxml2-dev libblas-dev
liblapack-dev libfontconfig1-dev

#install R and packages
  install_packages r-base r-base-dev

  R --slave -e 'install.packages(c("dplyr", "ggplot2", "purrr", "readr",
"stringr", "rmarkdown", "knitr",
  "tidyr", "tibble", "magrittr", "lubridate", "plyr", "Hmisc", "DescTools",
"gsubfn", "reshape",
  "corrplot", "english", "minpack.lm", "msm" ),
  repos="https://cloud.r-project.org/")'

  wget
https://github.com/Asimeng/LuminexPipeline/raw/main/inst/package_source%20cod
e/LuminexPipeline.tar.gz

  R CMD INSTALL LuminexPipeline.tar.gz

%test

  R --version

%labels

  Author Jesse Asimeng
```

Shared libraries

```
/etc/alternatives/libblas.a-x86_64-linux-gnu
/etc/alternatives/libblas.so.3-x86_64-linux-gnu
/etc/alternatives/libblas.so-x86_64-linux-gnu
```

```
/etc/alternatives/liblapack.so.3-x86_64-linux-gnu
/etc/R
/lib64/ld-linux-x86-64.so.2
/lib/x86_64-linux-gnu/ld-2.31.so
/lib/x86_64-linux-gnu/libbz2.so.1
/lib/x86_64-linux-gnu/libbz2.so.1.0
/lib/x86_64-linux-gnu/libbz2.so.1.0.4
/lib/x86_64-linux-gnu/libc.so.6
/lib/x86_64-linux-gnu/libdl.so.2
/lib/x86_64-linux-gnu/libgcc_s.so.1
/lib/x86_64-linux-gnu/liblzma.so.5
/lib/x86_64-linux-gnu/liblzma.so.5.2.5
/lib/x86_64-linux-gnu/libm.so.6
/lib/x86_64-linux-gnu/libpthread.so.0
/lib/x86_64-linux-gnu/libreadline.so.8
/lib/x86_64-linux-gnu/libreadline.so.8.1
/lib/x86_64-linux-gnu/libtinfo.so.6
/lib/x86_64-linux-gnu/libtinfo.so.6.2
/lib/x86_64-linux-gnu/libz.so.1
/lib/x86_64-linux-gnu/libz.so.1.2.11
/usr/bin/R
/usr/bin/Rscript
/usr/lib/libR.so
/usr/lib/R
/usr/lib/x86_64-linux-gnu/blas
/usr/lib/x86_64-linux-gnu/lapack
/usr/lib/x86_64-linux-gnu/libblas.so
/usr/lib/x86_64-linux-gnu/libblas.so.3
/usr/lib/x86_64-linux-gnu/libbz2.so
/usr/lib/x86_64-linux-gnu/libc.so
/usr/lib/x86_64-linux-gnu/libdl.so
/usr/lib/x86_64-linux-gnu/libgfortran.so.5
/usr/lib/x86_64-linux-gnu/libgfortran.so.5.0.0
/usr/lib/x86_64-linux-gnu/libgomp.so.1
/usr/lib/x86_64-linux-gnu/libgomp.so.1.0.0
/usr/lib/x86_64-linux-gnu/libicudata.so
/usr/lib/x86_64-linux-gnu/libicudata.so.67
/usr/lib/x86_64-linux-gnu/libicudata.so.67.1
/usr/lib/x86_64-linux-gnu/libicui18n.so
/usr/lib/x86_64-linux-gnu/libicui18n.so.67
/usr/lib/x86_64-linux-gnu/libicui18n.so.67.1
/usr/lib/x86_64-linux-gnu/libicuuc.so
/usr/lib/x86_64-linux-gnu/libicuuc.so.67
/usr/lib/x86_64-linux-gnu/libicuuc.so.67.1
/usr/lib/x86_64-linux-gnu/libjpeg.so.62
/usr/lib/x86_64-linux-gnu/libjpeg.so.62.3.0
/usr/lib/x86_64-linux-gnu/liblapack.so.3
/usr/lib/x86_64-linux-gnu/liblzma.so
/usr/lib/x86_64-linux-gnu/libm.so
/usr/lib/x86_64-linux-gnu/libpcre2-8.so
/usr/lib/x86_64-linux-gnu/libpcre2-8.so.0
/usr/lib/x86_64-linux-gnu/libpcre2-8.so.0.10.1
/usr/lib/x86_64-linux-gnu/libpng16.so.16.37.0
/usr/lib/x86_64-linux-gnu/libpthread.so
/usr/lib/x86_64-linux-gnu/libquadmath.so.0.0.0
/usr/lib/x86_64-linux-gnu/libreadline.so
/usr/lib/x86_64-linux-gnu/libstdc++.so.6
```

```
/usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28
/usr/lib/x86_64-linux-gnu/libtinfo.so
/usr/lib/x86_64-linux-gnu/libxml2.so.2
/usr/lib/x86_64-linux-gnu/libxml2.so.2.9.10
/usr/lib/x86_64-linux-gnu/libz.so
/usr/local/lib/R
/usr/share/R
/usr/lib/x86_64-linux-gnu/libtcl8.6.so
/usr/lib/x86_64-linux-gnu/libtk8.6.so
/usr/lib/x86_64-linux-gnu/libXft.so.2
/usr/lib/x86_64-linux-gnu/libXft.so.2.3.2
/usr/lib/x86_64-linux-gnu/libfontconfig.so.1
/usr/lib/x86_64-linux-gnu/libfontconfig.so.1.12.0
/usr/lib/x86_64-linux-gnu/libX11.so.6
/usr/lib/x86_64-linux-gnu/libX11.so.6.4.0
/usr/lib/x86_64-linux-gnu/libXss.so.1
/usr/lib/x86_64-linux-gnu/libXss.so.1.0.0
/usr/lib/x86_64-linux-gnu/libfreetype.so.6
/usr/lib/x86_64-linux-gnu/libfreetype.so.6.17.4
/usr/lib/x86_64-linux-gnu/libXrender.so.1
/usr/lib/x86_64-linux-gnu/libXrender.so.1.3.0
/lib/x86_64-linux-gnu/libexpat.so.1
/lib/x86_64-linux-gnu/libexpat.so.1.6.12
/usr/lib/x86_64-linux-gnu/libxcb.so.1
/usr/lib/x86_64-linux-gnu/libxcb.so.1.1.0
/usr/lib/x86_64-linux-gnu/libXext.so.6
/usr/lib/x86_64-linux-gnu/libXext.so.6.4.0
/usr/lib/x86_64-linux-gnu/libbrotlicommon.so.1
/usr/lib/x86_64-linux-gnu/libbrotlidec.so.1
/usr/lib/x86_64-linux-gnu/libbrotlidec.so.1.0.9
/usr/lib/x86_64-linux-gnu/libXau.so.6
/usr/lib/x86_64-linux-gnu/libXau.so.6.0.0
/usr/lib/x86_64-linux-gnu/libXdmcp.so.6
/usr/lib/x86_64-linux-gnu/libXdmcp.so.6.0.0
/usr/lib/x86_64-linux-gnu/libXdmcp.so.6
/usr/lib/x86_64-linux-gnu/libXdmcp.so.6.0.0
/usr/lib/x86_64-linux-gnu/libbrotlicommon.so.1
/usr/lib/x86_64-linux-gnu/libbrotlicommon.so.1.0.9
/usr/lib/x86_64-linux-gnu/libbsd.so.0
/usr/lib/x86_64-linux-gnu/libbsd.so.0.11.3
/usr/lib/x86_64-linux-gnu/libmd.so.0
/usr/lib/x86_64-linux-gnu/libmd.so.0.0.4
/usr/share/tcltk/tcl8.6/init.tcl
```

Nextflow script

```
nextflow.enable.dsl=2

params.aref = "/home/jesse/lum_analyte_ref.RData"
params.data_dir = '/home/jesse/dataset01'
params.tech_reps = 1
params.instrument_names = 'bp, mp'
params.facet = 5
params.cor_type = "spearman"
//params.instrument_name2 = "mp"
```

```
process CONFIG{

  input:
  val data_dir
  val aref
  val tech_reps

  script:
  """
  #!/usr/bin/env Rscript

  setwd("${projectDir}")

  arefs <- "$aref"

  LuminexPipeline::pipeline_config(wd = "${projectDir}", dd = "$data_dir",
aref = arefs, tech_reps = $tech_reps)

  """
}
  process DATA_IMPORT{

  input:
  val data_dir

  output:
  val "${projectDir}/rds/1_dta_import.rds"

  script:
  """
  #!/usr/bin/env Rscript

  setwd("${projectDir}")

  LuminexPipeline::data_import("$data_dir")
  """
  }

  process FILENAME_SEPARATE{
    input:
    val "data_out"
    val instrument_names
    //val instrument_name2

    output:
    val "${projectDir}/rds/2_dta_separate.rds"

    script:
    """
    #!/usr/bin/env Rscript

    setwd("${projectDir}")

    dta <- readRDS("${data_out}")

    instrument_names <- c("$instrument_names")
    LuminexPipeline::filename_separate(data = dta, instrument_names =
instrument_names)
    """
  }

  process DATA_CLEAN{

  input:
  val "data_out"

  output:
  val "${projectDir}/rds/3_dta_colnames_clean.rds"

  script:
  """
  #!/usr/bin/env Rscript

  setwd("${projectDir}")

  dta1 <- readRDS("${data_out}")

  LuminexPipeline::colnames_clean(dta1)

  dta2 <- readRDS("${data_out}")
  """
}

process ANALYTE_FIX{

  input:
  val "data_out"
  val aref

  output:
  val "${projectDir}/rds/4_dta_analyte_ref.rds"

  script:
  """
  #!/usr/bin/env Rscript

  setwd("${projectDir}")

  arefs <- "$aref"

  attach("$aref")

  dta2 <- readRDS("${data_out}")

  LuminexPipeline::analyte_names_fix2(arefs, dta2)
  """
}

process DATA_SAVE {

  input:
  val "data_out"
  output:
  val "${projectDir}/rds/5_dta_raw.rds"
  val "${projectDir}/rds/6_dta_symbol_remove.rds"

  script:
  """
  #!/usr/bin/env Rscript

  setwd("${projectDir}")

  dta3 <- readRDS("${data_out}")

  LuminexPipeline::save_raw(dta3)

  LuminexPipeline::symbols_remove(dta3)
  """
}

process DATA_SPLIT {
  input:
  val "data_out"

  output:
  val "${projectDir}/rds/7_dta_list.rds"

  script:
  """
  #!/usr/bin/env Rscript

  setwd("${projectDir}")

  dta4 <- readRDS("${data_out}")

  LuminexPipeline::data_split(dta = dta4)
  """

}

process REPORTS {
  input:
  val facet
  val cor_type
  val "input_data"

  output:
  val "${projectDir}/rds/statistical_report.html"

  script:
  """
  #!/usr/bin/env Rscript

  setwd("${projectDir}")

  dat <- readRDS("${input_data}")

  library(knitr)
  rmarkdown::render(
  input = paste0(system.file(package = "LuminexPipeline"),
"/extdata/rmd/reports.Rmd"),
  output_file = "${projectDir}/rds/statistical_report.html",
  params = list(cor_type = "${cor_type}", facet = ${facet}, data = dat),
  encoding     = 'UTF-8'
)

  """
}


workflow{

  def analyte_ref_ch = Channel.value(params.aref)
  def data_dir_ch = Channel.value(params.data_dir)
  def data_out = Channel.value("${projectDir}/rds/dta.rds")
  def tech_rep_ch = Channel.value(params.tech_reps)
  def ins_names = Channel.value(params.instrument_names)
  //def ins_name2 = Channel.value(params.instrument_name2)
  def facet = Channel.value(params.facet)
  def cor_type = Channel.value(params.cor_type)


  CONFIG(data_dir_ch, analyte_ref_ch, tech_rep_ch)
  DATA_IMPORT(data_dir_ch)
  FILENAME_SEPARATE(DATA_IMPORT.out, ins_names)
  DATA_CLEAN(FILENAME_SEPARATE.out)
  ANALYTE_FIX(DATA_CLEAN.out, analyte_ref_ch)
  DATA_SAVE(ANALYTE_FIX.out)
  DATA_SPLIT(DATA_SAVE.out[1])
  REPORTS(facet, cor_type, DATA_SAVE.out[1])
}
```

Statistical summary

These functions together with old functions written by Ncité and curve-fitting functions from the drLumi package are compiled into an R package. The package is available at https://github.com/Asimeng/LuminexPipeline

```
#' Summary table
#'
#' @param dta processed dataframe
#'
#' @return a table of summaries
#' @export
#'
#'
summ_table <- function(dta) {

  options(scipen = 999, digits = 1)

  summary_table <- dta %>%

    group_by(analyte) %>%
    dplyr::summarise(
      minimum = min(conc_obs_num, na.rm = TRUE),
      maximum = max(conc_obs_num, na.rm = TRUE),
      mean = mean(conc_obs_num, na.rm = TRUE),
      first_quart = quantile(conc_obs_num,  probs = 0.25, na.rm = TRUE),
      median = median(conc_obs_num, na.rm = TRUE),
      third_quart = quantile(conc_obs_num, probs = 0.75, na.rm = TRUE),
      NAs = sum(is.na(conc_obs_num)),
      available_obs = sum(!is.na(conc_obs_num)),
      total_obs = n()) %>%

    ungroup()

  #readr::write_rds(summary_table, "rds/summary_table.rds")

  return(summary_table)

}



#' Histogram plot
#'
#' @param dta_prtc processed dataframe
#' @param facet_rows numeric value indicating number of rows to facet.
#'
#' @return a plot of histogram
#' @export
#'
#'
#'
histogram_plot <- function(dta_prtc, facet_rows = 5) {

  dta_prtc %>%
    ggplot2::ggplot(aes(x = conc_obs_num)) +
    geom_histogram(aes(y = ..density..), colour = "black", fill = "grey",
na.rm = TRUE) +
    geom_density(colour = "blue") +
    facet_wrap(~ analyte, nrow = facet_rows,  scales ="free") +
    ggtitle("Plot with untransformed data")
}



#' Test for normality
#'
#' @param dat data to generate test of normality table from
#'
#' @return a table of normality test
#' @export
#'
#'
norm_tab <- function(dat){

  options(scipen = F, digits = 1)
  tab <-  dat %>%

    group_by(analyte) %>%

    summarise(kurtosis = DescTools::Kurt(conc_obs_num, na.rm = T),

              skewness = DescTools::Skew(conc_obs_num, na.rm = T),

              Shapiro.Francia = if(sum(!is.na(conc_obs_num)) >= 5) {

                DescTools::ShapiroFranciaTest(conc_obs_num)[["statistic"]]

              } else {NA
              },

              p_val = if(sum(!is.na(conc_obs_num)) >= 5) {

                DescTools::ShapiroFranciaTest(conc_obs_num)[["p.value"]]

              } else{NA
              }
    )

  return(view(tab))
}



#' Generate quantile-quantile plots to assess normality
#'
#' @param dta dataframe from pipeline
#' @param facet_row numeric value indicating number of rows to facet
#'
#' @return A quantile-quantil graph
#' @export
#'
#'
qq_plot <- function(dta, facet_row = 5){

  dta %>%
    ggplot2::ggplot(aes(sample = conc_obs_num)) +
    stat_qq() +
    stat_qq_line() +
    ggplot2::facet_wrap(~ analyte, nrow = facet_row, scales = "free")
}



#' Comparison between groups being analysed
#'
#' @param dta_prtc participant data
#' @param dta_clin clinical data
#'
#' @return A dataframe of comparison between groups summary
#' @export
#'
#'
```

```
grp_test <- function(dta_prtc, dta_clin){

  grp_dat <- inner_join(dta_prtc, dta_clin, by = "description") %>%
    select(analyte, conc_obs_num, age, group) %>%
    mutate(group = as.factor(group)
    ) %>%

# Analytes with a single observation eg (IL-6 of test dataset) returns an
error. work around to filter such data

    group_by(analyte) %>%

    summarise(p.value = wilcox.test(conc_obs_num ~ group )[["p.value"]],

              w.statistic = wilcox.test(conc_obs_num ~ group)[["statistic"]])

  return(grp_dat)
}



#' Correlation tests
#'
#' @param dta dataframe from the pipeline
#' @param cor_type type of correlation in string format. eg "pearson"
#'
#' @return a flatten correlation matrix (table)
#' @export
#'
#'
correlation <- function(dta, cor_type = "spearman"){

  piv_dat <- dta %>%

    select(analyte, conc_obs_num) %>%

    group_by(analyte) %>%

    mutate(row = row_number()) %>%

    tidyr::pivot_wider(
      names_from = analyte,
      values_from = conc_obs_num) %>%

    dplyr::select(-row)

  cor <- Hmisc::rcorr(as.matrix(piv_dat), type = cor_type)

  flattenCorrMatrix <- function(cormat, pmat) {
    ut <- upper.tri(cormat)
    data.frame(
      analyte_1 = rownames(cormat)[row(cormat)[ut]],
      analyte_2 = rownames(cormat)[col(cormat)[ut]],
      co.eff  = (cormat)[ut],
      p.value = pmat[ut]
    )
  }
  cor <- flattenCorrMatrix(cor$r, cor$P)

  arr_cor <- arrange(cor, desc(abs(co.eff))) %>%
    filter(!is.na(co.eff))

  return(arr_cor)
}



#' Visualise correlogram - analytes
#'
#' @param dta dataframe from pipeline
#' @param cor_type character, type of correlation to compute
#'
#' @return A graph (correlogram) of correlations
#' @export
#'
#'
cor_plot <- function(dta, cor_type = "spearman"){

  piv_dat <- dta %>%

    select(analyte, conc_obs_num) %>%

    group_by(analyte) %>%

    mutate(row = row_number()) %>%

    tidyr::pivot_wider(
      names_from = analyte,
      values_from = conc_obs_num) %>%

    dplyr::select(-row)


  #res <- cor(piv_dat)

  cor <- Hmisc::rcorr(as.matrix(piv_dat), type = cor_type)

  #round(res, 2)
  #corrplot(res, type = "upper", order = "hclust",tl.col = "black", tl.srt =
45)

  corrplot::corrplot(cor[["r"]], type="upper", order="alphabet",
                      tl.cex = 0.6,tl.col="black", tl.srt=45)
}

#' render statistical summary report
#'
#' @param dta rds file with its extension
#'
#' @return an html report
#' @export
#'
#'
```

```
render_report <- function(dta) {

  rmarkdown::render(
    input = "vignettes/reports.Rmd",
    output_dir = "rds/statistical_summary_report.html",
    params = list(
      directory = "rds",
      file = dta
    )
  )

}
```

Unit testing

```
test_that("column names transform to lower cases after applying
colnames_clean
          function", {

  dta_separate <- readRDS("rds/dta_separate.rds")

  col_names <- names(colnames_clean(dta_separate))

  expect_equal(grep("^[[:upper:]]+$", col_names), integer(0)
)

})

test_that("colnames_clean function cleans white spaces in column names",{

  dta_separate <- readRDS("rds/dta_separate.rds")

  col_names <- names(colnames_clean(dta_separate))

  expect_equal(grep("\\s", col_names), integer(0))
})




test_that("data_import function throws an error if input directory is empty",
{

  expect_error(data_import("test_empty_dir/"),
               "check input directory: misspelt or empty directory")
})

test_that("files without .txt extensions throw a warning message", {

  expect_warning(data_import("test_wrong_extension/"),
                 "some files may have incorrect format")

})
```

```
test_that("data output of data_import function has correct dimensions(column
length)", {

  expect_equal(ncol(data_import("test_data_dir/")),
               39)
})

test_that("data_import function creates a filename column", {

  col_names <- colnames(data_import("test_data_dir/"))

  expect_equal("filename" %in% col_names,
               TRUE)
})

test_that("output of data_import function does not have a repeating header",
{

  dat <- data_import("test_data_dir/")

  expect_equal(which(dat$Analyte == "Analyte"), integer(0))
})

test_that("data_import function writes 3 files to rds directory", {

  data_import("test_data_dir/")

  expect_match(list.files("rds"), "dta_import.rds", all = FALSE)
  expect_match(list.files("rds"), "rep_files.rds", all = FALSE)
  expect_match(list.files("rds"), "rep_files_invalid.rds", all = FALSE)

})


test_that("filename_separate function joins columns: 'date, kit, instrument,
          plate, rerun' to output from preceding function (data_import)", {

            dta_import <- readRDS("rds/dta_import.rds")

            instrument_names <- c("bp", "mp")
            #test_colnames <- c("date", "kit", "instrument", "plate",
"rerun")

            separated_filename <- filename_separate(dta_import,
instrument_names)

            expect_match(colnames(separated_filename), "date", all = FALSE)
            expect_match(colnames(separated_filename), "kit", all = FALSE)
            expect_match(colnames(separated_filename), "instrument", all =
FALSE)
            expect_match(colnames(separated_filename), "plate", all = FALSE)
            expect_match(colnames(separated_filename), "rerun", all = FALSE)

          })

test_that("parsed dates have the correct class", {
  dta_import <- readRDS("rds/dta_import.rds")

  instrument_names <- c("bp", "mp")

  separated_filename <- filename_separate(dta_import, instrument_names)

  expect_equal(class(separated_filename$date), c("POSIXct", "POSIXt"))

})


test_that("input data without a filename column stops with an error",{

  dta_import <- readRDS("rds/dta_test.rds")

  instrument_names <- c("bp", "mp")

  expect_error(filename_separate(dta_import, instrument_names),
               "check input data: input may not have a 'filename' column")
})

test_that("output of filename_separate function is of class tibble", {

  dta_import <- readRDS("rds/dta_import.rds")

  instrument_names <- c("bp", "mp")

  separated_filename <- filename_separate(dta_import, instrument_names)

  expect_equal(class(separated_filename), c("tbl_df","tbl","data.frame"))
})

test_that("filename_separate function writes processed data to rds files", {

  dta_import <- readRDS("rds/dta_import.rds")

  instrument_names <- c("bp", "mp")

  filename_separate(dta_import, instrument_names)

  expect_match(list.files("rds"), "dta_separate.rds", all = FALSE)
})
```

Shiny

UI (user interface)

```
shinyUI(
  navbarPage(
    title = "Luminex App",
    theme = "styles.css",

    tabPanel(
      title = "Home",
      h1("About"),
      h4(about),
      imageOutput(outputId = "luminex_image" )

    ),

    tabPanel(
      title = "Analysis",

      sidebarLayout(

        sidebarPanel(

          fileInput(
            inputId = "upload_file",
            label = "Upload your luminex rds file here: ",
            placeholder = "No file selected yet",
            multiple = FALSE,
            accept = c(".rds")
          ),


          checkboxInput(
            inputId = "head_data",
            label = "Display first 10 rows of your data? ",
            value = FALSE
          ),

          selectInput(
            inputId = "analyte_1",
            label = "select first analyte for correlation test",
            choices = NULL
          ),

          selectInput(
            inputId = "analyte_2",
            label = "select second analyte for correlation test",
            choices = NULL
          ),

          selectInput(
            inputId = "cor_type",
            label = "correlation type",
            choices = c("pearson", "spearman")
          ),

          checkboxInput(

            inputId = "show_cor_tab",
            label = "show correlation table ?",
            value = FALSE
          ),

        ),

        mainPanel(
          dataTableOutput("head_data_txt"),

          h4(textOutput("cor_result")),

          dataTableOutput("show_cor_tab"),
        )
      )
    ),

    tabPanel(
      title = "Visualisation",

      sidebarLayout(

        sidebarPanel(

          selectInput(
            inputId = "graphs",
            label = "select type of graph to visualise",
            choices = graph_type
          ),

          selectInput(
            inputId = "transform",
            label = "Select transformation type",
            choices = c("log", "box-cox", "no transformation")
          ),

          selectInput(
            inputId = "analyte_plot",
            label = "select analyte to visualise",
            choices = NULL
          ),

          tabsetPanel(
            id = "switch",
            type = "hidden",
            tabPanel("histogram",

                     sliderInput(
                       inputId = "bins",
                       label = "number of bins for histogram",
                       min = 1,
                       value = 30,
                       max = 50)
            ),

            tabPanel("boxplot",
                     NULL

            ),

            tabPanel("q-q plot",
                 NULL
            )

          )
```

```
        ),

        mainPanel(
          plotOutput("individual_plot"),

          h4(textOutput("graph")),
        )
      )

    )
  )
)
```

Server

```
library(shiny)

shinyServer(function(input, output){

  output$luminex_image <- renderImage(
    list(src = "www/luminex_logo.png",
         width = "300px")
  )

  data <- reactive({

    if (!is.character(input$upload_file$datapath) &&
length(input$upload_file$datapath) < 1 ){

      data <- read_rds("www/dat.rds")

      data

      } else{

      req(input$upload_file)

    ext <- tools::file_ext(input$upload_file$name)

    switch(ext,
           rds = read_rds(input$upload_file$datapath),

           validate("Invalid file; Please upload a valid, processed and
cleaned Luminex .rds file")
    )
  }

  })


  names_analytes <- reactive(unique(data()$analyte))

  observeEvent(data(), {
```

```
  updateSelectInput(inputId = "analyte_1", choices = names_analytes())

  updateSelectInput(inputId = "analyte_2", choices = names_analytes())

  updateSelectInput(inputId = "analyte_plot", choices = names_analytes())

})


piv_dat <- reactive({

  data() %>%

    select(analyte, obs_conc) %>%

    group_by(analyte) %>%

    mutate(row = row_number()) %>%

    pivot_wider(
      names_from = analyte,
      values_from = obs_conc) %>%

    select(-row)

})


output$head_data_txt <- renderDataTable({

  if (input$head_data == TRUE){

    head(data(), 10)

  } else {

     NULL
  }
}
)


output$show_cor_tab <- renderDataTable(

  if (input$show_cor_tab == TRUE & input$cor_type == "spearman") {

    correlation_table(data(), "spearman")

  } else if(input$show_cor_tab == TRUE & input$cor_type == "pearson") {

   correlation_table(data(), "pearson")

  } else {

    NULL
  }
  )

  corr <- reactive(cor(

    x = piv_dat()[input$analyte_1],

    y = piv_dat()[input$analyte_2],

    use = "pairwise.complete.obs",

    method = input$cor_type))

  cor_text <-  reactive(paste0(

    "The ",
    input$cor_type,
    " correlation coefficient of ",
    input$analyte_1, " and ", input$analyte_2,
    " is ", round(corr(), digits = 3)))


  output$cor_result <- renderText({

    cor_text()

    })

  observeEvent(input$graphs, {

    updateTabsetPanel(inputId = "switch", selected = input$graphs)

  })


  bc_dat <- reactive(filter_analyte_bc(data(), anlyt_name =
input$analyte_plot))

  log_dat <- reactive(filter_analyte_log(data(), anlyt_name =
input$analyte_plot))

  new_dat <- reactive(filter(data(), analyte == input$analyte_plot))


  histo <- function(dat, num_var){

    ggplot(dat, aes(x = num_var)) +

      geom_histogram(bins = input$bins, colour = "black", fill = "grey",
na.rm = TRUE) +

      labs(x = "conc", y = "count")
  }

  output$individual_plot <- renderPlot({

    if (input$transform == "log" & input$graphs == "boxplot")
{boxp(log_dat(), log_dat()$log)}
```

```
   else if (input$transform == "log" & input$graphs == "histogram")
{histo(log_dat(), log_dat()$log)}

   else if (input$transform == "log" & input$graphs == "q-q plot")
{qq(log_dat(), num_var = log_dat()$log)}

   else if (input$transform == "box-cox" & input$graphs == "boxplot")
{boxp(bc_dat(), bc_dat()$bc)}

   else if (input$transform == "box-cox" & input$graphs == "histogram")
{histo(bc_dat(), bc_dat()$bc)}

   else if (input$transform == "box-cox" & input$graphs == "q-q plot")
{qq(bc_dat(), num_var = bc_dat()$bc)}

   else if (input$transform == "no transformation" & input$graphs ==
"boxplot") {boxp(new_dat(), new_dat()$obs_conc)}

   else if (input$transform == "no transformation" & input$graphs ==
"histogram") {histo(new_dat(), new_dat()$obs_conc)}

   else if (input$transform == "no transformation" & input$graphs == "q-q
plot") {qq(new_dat(), num_var = new_dat()$obs_conc)}

   else {NULL}

  }, res = 96)

  output$graph <- renderText(

    paste("You are visualising", input$analyte_plot,
          "in a", input$graphs, "with a",
          input$transform, "transformation")
  )

}
```

R

```
library(Hmisc)
library(tidyr)
library(dplyr)

correlation_table <- function(dat, type){

  piv_dat <- dat %>%

    select(analyte, obs_conc) %>%

    group_by(analyte) %>%

    mutate(row = row_number()) %>%

    pivot_wider(
      names_from = analyte,
      values_from = obs_conc) %>%

    select(-row)


  cor <- rcorr(as.matrix(piv_dat), type = type)

  flattenCorrMatrix <- function(cormat, pmat) {
    ut <- upper.tri(cormat)
    data.frame(
      row = rownames(cormat)[row(cormat)[ut]],
      column = rownames(cormat)[col(cormat)[ut]],
      cor  =(cormat)[ut],
      p = pmat[ut]
    )
  }

  cor <- flattenCorrMatrix(cor$r, cor$P)

  arr_cor <- arrange(cor, desc(abs(cor))) %>%

    filter(!is.na(cor))

  arr_cor
}


library(car)

filter_analyte_bc <- function(dt, anlyt_name){

  dat <- dt %>% filter(analyte == anlyt_name) %>%
    select(analyte, obs_conc)

  lambda <- powerTransform(dat$obs_conc)$lambda

  dat <- dat %>% mutate(bc = ((obs_conc^lambda - 1)/lambda))

  return(dat)

}

filter_analyte_log <- function(dt, anlyt_name){

  dat <- dt %>% filter(analyte == anlyt_name) %>%
    select(analyte, obs_conc) %>%

    mutate(log = log(obs_conc))

  return(dat)

}


library(ggplot2)

boxp <- function(dat, num_var){
  ggplot(dat, aes(y = num_var)) +
    geom_boxplot(na.rm = TRUE) +
    labs(y = "conc")
}

qq <- function(dat, num_var){
  qqplot <-  ggplot(dat, aes(sample = num_var)) +
    stat_qq()+
    stat_qq_line()

  qqplot
}



library(shiny)
library(readr)
library(dplyr)
library(Hmisc)

graph_type <- c("histogram", "boxplot", "q-q plot")

# data <- read_rds("www/dat.rds")


about <- tags$div(

  tags$p("The Luminex pipeline is an R-based pipeline that reads .txt
files produced by Luminex instruments and outputs experimental data, quality
control data and metadata. This app is created to visualise and explore
statistical summaries
from pipeline output (data).  More specifically, correlation analysis(Pearson
and Spearman) between analytes
and visualisation of the effects of transformation (Box-Cox and log) on
analyte distributions"),

  tags$p("USAGE: For purposes of exploration (assessing functionality of the
app), a very small underlying data
is provided. One can, however, upload their processed Luminex data for use on
the app - this will override
the underlying data."),

  tags$p("WARNING: This app/utility is meant for exploratory analysis only
and should only be used as such!")
)

luminex_image <- tags$img(src = "www/luminex_logo.png", width = "100px",
height = "100px")
```