Dissertation

submitted to the

Combined Faculties for the Natural Sciences and for Mathematics of the
Ruperto-Carola University of Heidelberg, Germany
for the degree of
Doctor of Natural Sciences

presented by
Svetlana Ovchinnikova, M.Sc.
born in Vitebsk, Belarus

Oral examination: 26th of November 2021

Visual and Interactive Exploration of Omics Data

Referees: Prof. Dr. Henrik Kaessmann
Prof. Dr. Benedikt Brors

Abstract

Today many biological studies rely on high-throughput techniques that yield data on thousands of samples or cells with tens of thousands of measured features. Exploring such an amount of data poses a visualisation challenge, that can be solved by switching from static plots to interactive ones. This provides a way of intuitive navigation through large datasets in a manner that helps the user to grasp the bigger picture visually. The field of interactive visualisation of biological data is an actively developing one. However, it is more often used only for data presentation and is still not so common during a research project's early, exploratory stages. This project is aimed to explore and propose solutions to fill this “visualisation gap”.

I investigate the possible benefits for biological studies of the combination of JavaScript and R programming languages. R is one of the most common tools in biostatistics and provides a wide variety of implemented libraries for processing omics data. JavaScript is a language for enabling user's interaction with a web page and nowadays is used by most available web resources. Thus, the two languages are very effective in their own application fields, and interactive visualisation of omics data lies precisely in their combination. As an outcome of the project, I present three R packages (one of which also provides a purely JavaScript interface) for data visualisation.

The first one, ”jrc”, is intended for package developers and serves as a foothold for further project steps. ”jrc” provides direct communication between a web page and a running R session. It allows the user to run R code from a web page and execute JavaScript code from an R session. In addition, it provides a basis for publicly available interactive apps deployed on a server. With this, ”jrc” can be used as a foundation for the packages that use JavaScript to visualise data stored and processed in an R session.

The second one is ”sleepwalk”. It is a simple but effective tool to explore distortions introduced by dimensionality reduction techniques when visualising biological data. These approaches (such as MDS, t-SNE, UMAP, etc.) are particularly but not exclusively popular in single-cell studies. There, researchers commonly visualise cells as points on 2D embedding and then study the obtained clusters and trajectories. ”sleepwalk” helps one explore the underlying patterns of such plots by interactively comparing the dis-

played neighbourhoods to the original distances in the high-dimensional feature space.

Finally, "rlc" (or LinkedCharts) is a plotting library that allows one to construct one's interactive app with minimal effort and coding skills. It is designed in a way that does not require users to learn special complex syntax. Instead, I adjusted it to routinely used practices in the omics data exploration. I also left a broad space for customisation so that users could adapt the apps to their particular tasks rather than trying to fit their data into the predefined templates. With all this, the "rlc" can be a powerful tool to facilitate exploratory data analysis by interactive visualisation. It centers on but is not limited to the idea of linking multiple charts when a user's manipulation with one plot affect another one (for example, a click on a point shows more specific information on the thus selected sample).

Overall, the packages presented here can be helpful when applied in everyday analyses and serve as a basis and inspiration for new solutions in the interactive visualisation of biological data.

Zusammenfassung

Heute basieren viele biologische Studien auf Hochdurchsatz-Techniken, die Daten von Tausenden von Proben oder Zellen erzeugen mit Zehntausenden von gemessenen Merkmalen. Derartige Datenmengen zu sichten stellt eine Herausforderung dar im Bereich der Datenvisualisierung, die gelöst werden kann, indem man statische durch interaktive Plots ersetzt. Dies bietet eine intuitive Navigation durch große Datensätze in einer Art, die dem NButzer hilft, das großere Ganze visuell zu erfassen. Das Forschungsgebiet der interaktiven Darstellung biologischer Daten entwickelt sich aktiv fort. Dennoch wird diese oft nur zur Präsentation der Daten verwendet, während eine Nutzung in den frühen Stadien eines Forschungsprojekts noch selten ist. Das vorliegende Projekt zielt darauf, diese "Visualisierungs-Lücke" zu erforschen und Lösungen zu ihrer Schließung vorzuschlagen.
Ich untersuche den möglichen Nutzen einer Kombination der Programmiersprachen JavaScript und R für biologische STudien. R ist eines der meistgenutzten Werkzeuge der Biostatistik und bietet eine große Vielfalt and Bibliotheken zyr Verarbeitung von Omics-Daten. JavaScript ist eine Sprache um die Interaktion des Nutzers mit einer Webseite zu ermöglichen und wird heute von den meisten Web-Resourcen genutzt. Somit sind diese beiden Sprachen von hohem Nutzen in ihren jeweiligen Anwendungsgebieten, und die interaktive Visualisierung von Omics-Daten liegt daher genau in ihrer Verbindung. Als Ergebnis des Projekts stelle ich drei R-Pakete zur Datenvisualisierung vor (eines davon bietet auch eine reine JavaScript-Schnittstelle an).
Das erste Paket, "jrc", ist für Paket-Entwickler gedacht und dient als Grundstein für die weiteren Schritte des projekts. "jrc" ermöglich direkte Kommunikation zwischen einer Webseite und einer laufenden R-Sitzung. Es erlaubt dem Nutzer, von der Webseite aus R-Code laufen zu lassen, und von der R-Sitzung aus JavaScript-Code. Darüberhinaus bietet es eine Grundlage für öffentlich verfügbare, auf einem Server aufgespielte interaktive Apps. Damit kann R als Grundlage genuzt werden für Pakete, die JavaScript nutzen, um Daten zu visualisieren, die in einer R-Sitzung gespeichert sind und dort verarbeitet wurden.
Das zweite Paket ist "sleepwalk". Hier handelt es sich um ein einfaches aber wirksames Wekrzeug um die Verzerrungen zu erkunden, die durch

Techniken zur Dimensionsreduktion entstehen, wenn biologische Daten auf diesem Weg visualisiert werden. Diese Techniken (MDS, t-SNE, UMAP, usw.) sind besonders in Einzel-Zell-Studien beliebt, aber nciht auf diese beschränkt. Dort stellen Forscher Zellen als Punkte in einer zweidimensionalen Einbettung dar und studieren dann die erhaltenen Cluster und Trajektorien. "sleepwalk" hilft, die solchen Plots zugrunde liegenden Muster zu untersuchen, indem man interaktiv die Nachbarschaften in der Darstellung vergleicht mit den ursprünglichen Abständen im hochdimensionalen Merkmalsraum.
Schliesslich ist "rlc" (oder "LinkedCharts") eine Bibliothek zur Datenvisualisierung, die es erlaubt, interaktive Apps mit minimalem Aufwand und geringen Programmierkenntnissen zu bauen. Es ist so gestaltet, dass der Nutzer keine spezielle komplizierte Syntax erlernen muss. Stattdessen habe ich es den Standardpraktiken der Omics-Daten-Exploration angepasst. Ich habe auch breiten Raum gelassen um Nutzern zu ermöglichen die App an die Aufgaben anzupassen, anstatt die Daten in vorbestimmt Schablonen pressen zu müssen. All dies macht "rlc" ein leistungsstarkes Werkzeug zur explorative Datenanalyse durch interaktive Visualisierung. Ohne darauf beschränkt zu sein, steht dabei im MIttelpunkt die Idee, mehrere Plots so zu verknüpfen, dass Nutzereingaben in einem Plor auch einen anderen beeinflussen kann. (Zum Beispiel kann ein Klick auf einen Datenpunkt Detailinformationen zu der so ausgewählten Probe anzeigen.)
Zusammengenommen sind die hier vorgestellten Paket nützlich für die routinemäßige Anwendung in Analysen und dient als Grundlage und Inspiration für neue Lösungen in der interaktiven Darstellung biologischer Daten.

Contents

Chapter 1

Introduction

1.1 Visualisation and big data in bioscience

Since the late 2000s, due to the advances in sequencing and high-throughput techniques, the amount of generated biological data has been growing exponentially [1]. The automatisation in data generation called for automatisation also in the data processing. Tools that previously only facilitated research work now turned into an absolute necessity. A researcher cannot realistically perform calculations for hundreds of samples and thousands of measurements without delegating at least some of this work to various algorithms.

However, besides challenges in data logistics, storage and exploration [2], the growing amount of information also poses a question of trust both in the data and in the conclusions drawn from it. Here, we are not talking about intentional mispresenting of results or any fraudulent behaviour. Rather, in a manually performed experiment, the researcher pays attention to every single sample and has a reasonable chance to notice if something is wrong. By contrast, in a high-throughput experiment, one has to rely on automatic quality assessments to filter out spurious data. Yet, there are just too many things that can theoretically go wrong to envisage all of them in a pipeline. Therefore, it is essential for the researchers to be able to look at their data. Since it is no longer possible to do so directly when dealing with big data, various visualisation techniques have become of extreme importance.

Nowadays, in biological studies, visualisation is not anymore just about sharing the results with the audience. It is of vital importance during most steps of any research work. The goal of this thesis is to develop tools to assist the exploratory analysis of big data with the least possible effort from the user's side.

1.1.1 Basics of visualization

The fundamental visualisation principles have been evolving since the first attempts to draw a map or other geometric diagrams. Long before any studies on cognitive perception, people already intuitively understood that information in the form of an image could be grasped more effectively than text or numbers. Moreover, history knows some examples of visualisation playing a pivotal role in problem-solving [3]. Yet, it still took centuries for today's most common ways of visualising information to appear and then to be acknowledged in most areas that involve research and communication [4]. Only in the 20th century, first attempts to approach the problem of effective visualisation became systematic, which was probably a result of computer graphics becoming widely available [5]. Today it is pretty hard to imagine a scientific biological paper without compelling and informative images.

The scope of this thesis lies in the visualisation of biological data of a particular structure. Here, we talk about some objects (samples, cells, patients, etc.) and a set of related measurements, which we call features. For such data, the visualisation task is to encode objects as graphic symbols (dots, lines, bars, etc.) and features as their properties (colour, position, size, etc.). The overview of possible static visual channels in such plots is given in Figure 1.1 which is based on [6].

1.1.2 Limitations of visualisation

Even from the definition of the visualisation type we want to concentrate on, one can see the main limitations for the data that can be effectively visualised.

The first one is the amount of data. The number of objects one can add to a plot is limited by the size of features that the human eye can distinguish [9]. By now, we have reached the point when, in an average study, printing a plot on a poster-size sheet of paper or using a modern 4K display may not be enough to distinguish all the dots of a scatter plot or all the cells of a heatmap. Of course, some kinds of charts allow objects to be put on top of each other, yet it works only up to a certain scale, and the problem of overplotting is quite well-known [10].

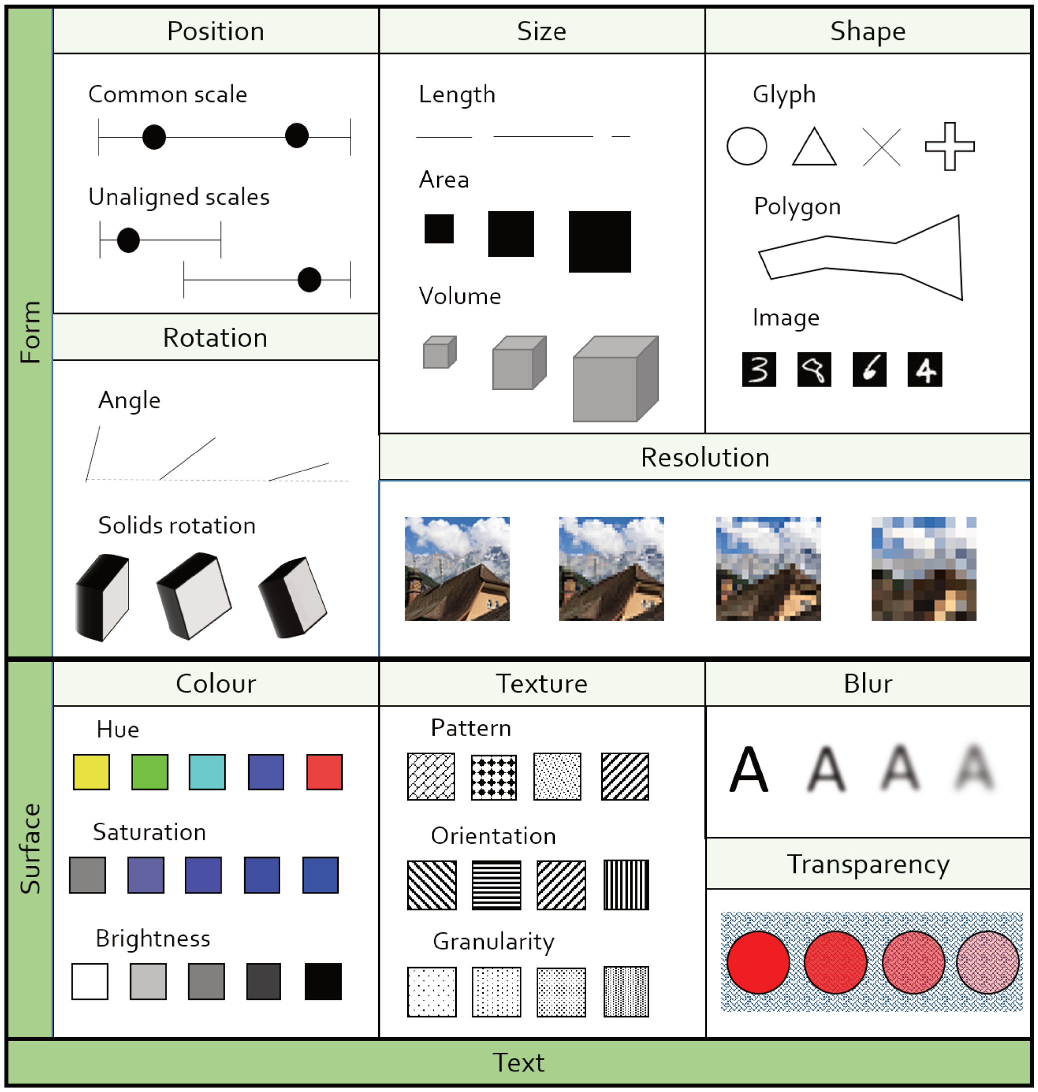


Figure 1.1: Possible visual channels to graphically encode information on data instances. The figure is a summary of the chapter "Aesthetics" from [6] with some additional details from [7]. Examples of images in the "Shape" section are taken from the MNIST dataset of handwritten digit images [8], the example photo for the "Resolution" is taken by me.

Secondly, there is only a certain amount of visual channels one can use to display known characteristics of the objects. Not only the size but also the complexity of information grows with the increase of collected and processed data. Today, we are still in the process of understanding how complex are biological systems that we study. And with this grows the number of factors that one may want to display to get a complete picture. Yet, overburdening a plot with too much information makes it harder to read. There are attempts to find creative solutions to present complex data [7, 11]. Still, even if the employed visual channels do not overlap, one may find it confusing to follow too many patterns at once.

Thus, when presenting data as a static image, a researcher has to decide what information to use and what to put aside to convey the message better [7]. It is, doubtless, an essential step since data often contain a lot of noise and information irrelevant to the point one tries to make. Yet, it may be helpful to provide the readers with a way to estimate the relevance of the omitted piece of data on their own to avoid biases [12] and boost confidence in the reported data patterns.

Limitations of visualisation during the exploratory phase of a study are even more pressing. Early in the project, the researcher does not yet know what information is relevant and what is noise. The urge to put as much information into the plot as possible is only natural at this point since omitting something may lead to missing a critical pattern. Even though in exploratory analysis, one may be willing to do with plots that are hard to read, the boundaries of how much information can be put there cannot be moved endlessly.

1.2 Interactivity

Today, engaging interactivity offers a solution to the problem of excessively complex and detailed plots. By now, the vast majority of scientific plots are no longer drawn by hand but generated with the help of some software. And the image, generated and stored electronically, does not have to be static as a printed one. Instead, various interfaces allow the user to manipulate plots with the most basic mouse actions, thus making the image interactive.

Attempts to utilise interactivity to facilitate plots' exploration started since the technology first allowed that [13, 14], and till nowadays, this field undergoes active development. In an interactive figure, there is no need to fix all the parameters or to exclude any data that do not contribute to the main idea. Instead, the user gets a chance to experiment with data and details quickly and intuitively, concentrating at once on the plot's most exciting or suspicious parts.

The advantage of interactivity lies beyond just simplifying navigation through big or complex data. When it takes just a click or two to add changes to a plot, it urges a researcher not to put aside ideas or concerns and thus go through the data more thoroughly. At the same time, readers can check the conclusions and claims of a paper on the fly without going through all the scripts and analysis, thus making the findings more credible. Therefore, I believe that further integration of interactive tools in a researcher's routine can significantly improve the quality of studies [15, 16].

1.2.1 Existing solutions

Numerous tools [17] now provide means of creating interactive visualisations not only for research [18–21] but also for any area that involves data analysis and presentation of any kind [22, 23].

Overall, we can classify the existing tools into three major categories.

- Low-level grammars such as D3 [24], or Vega [25] offer principles and the most basic tools, rather than ready-made solutions. These packages require good coding skills and some time for practice. Yet, since only the basic primitives are predefined, one can use these grammars to design an interactive app almost exactly to fit one's ideas. They can be used directly by experienced users to design tailored graphics solutions or become building material for more high-level packages.

- There is an assortment of more high-level, but still generally purposed packages that offer their primitives in the form of basic plot types [26, 27]. They also come with a predefined logic of how interactivity is ensured. Generally, they can be customised to quite a great extent, yet the syntax, accepted form of input data, and variety of available kinds of plots remain fixed. One of the earliest examples of this category is Vega-Lite [28], which allows specifying multiple plots and interactions between them in JSON format, the multi-language plotting library Plotly and the R package "shiny" [29]. The latter two can now be combined to provide more functional apps [30].

- The most abundant category of interactive apps are highly specialised packages that are usually designed for a particular task or to facilitate exploration with a specific type of data. These packages generally offer one or a couple of very easy-to-use functions or work just as a web resource. However, the amount of possible customisation is very limited if at all present. They can be quite helpful if one wants to apply them to the specific task they were designed for, but an attempt to even slightly change their functionality can become problematic. Therefore, the most popular tools of this category are the ones that bolster the most common steps during the research work. Here is just a couple of examples: ReconMap allows one to explore human metabolic network [18], Bandage visualises de novo genome assembly [19], Cerebro works with single-cell RNA-Seq data [20].

1.2.2 Visualization gap

Today many authors accompany their papers with an interactive resource to present their data and results (for example, [31–33] and many more). Though these visualisations are often useful and provide a comprehensive insight into the data, they chiefly serve presentation and communication purposes. Only after most of the work on the project is finished, researchers spend a couple of days deploying a fancy interactive app to share their data and results with the scientific community. Such a tendency can be called an interactive visualisation gap during the exploratory analysis [34], and it remains despite the growing diversity of tools for interactive visualisation. I believe that interactivity should become a part of everyday routine to facilitate data exploration. To this end, the tool to produce interactive apps should be both simple and highly customisable.

Simplicity should serve as an incentive to use interactivity whenever there is any slightest hint that it may be helpful. If a tool is too complicated, one may prefer to do most of the analysis by more habitual static means and wait for a special occasion when it is worthwhile to invest time and effort into an interactive app. It should also be similar in design to most common visualisation means since tools with too specific interfaces tend to be used by people with more extensive programming skills. Even if a researcher with less expertise in coding has an eager-to-help colleague, he or she may be unwilling to ask for assistance in petty everyday tasks and wait for something big.

Still, an attempt to simplify a tool may end in hardcoding and presetting too many parameters. As a result, a simple-to-use tool can require a particular data structure and be fit only for precise data flow patterns. Any attempt of going beyond in-built limitations, if at all possible, can cost a considerable effort. It is not necessarily a disadvantage since much of the routine work in the lab involves steps and data types typical for the given research area. Yet, it may be useful if one does not try to force the problem in the predefined mould but instead makes the visualisation tool fit the given task. For that, a reasonable degree of flexibility is required.

This work attempts to overcome the interactive visualisation gap with the tools that were designed and polished in collaboration with several biological labs to most effectively fit their everyday needs. As a result, I here present three R packages:

jrc	a tool for package developers to build interactive apps;
sleepwalk	a package to interactively explore 2D embeddings by comparing them to the original distances in the feature space;
rlc	an interactive plotting library based on the concept of linikng charts.

Chapter 2

Methods and datasets

2.1 Types of datasets

Visualisation is an extensive area even within the limits of bioscience. It is impossible to address all kinds of visualisation tasks, from complex interaction networks to 3D structures of big molecules, with the same approach. Therefore the scope of this thesis has to be limited to only a specific kind of data, which I will call “feature data”. In machine learning, a feature is an individual measurable property or characteristic of a phenomenon being observed [35]. So as “feature data” I define a set of objects of similar type (samples, cells, patient, etc.), each described by multiple measured or calculated features (specific molecule abundance in a sample, methylation level at a given position, phenotypical data, etc.). These data can be as simple as the weight of a group of mice or as complex as methylation levels of millions CpGs in hundreds or thousands of samples.

Feature data comprise quite a significant fraction of available biological knowledge. They are very diverse and present scientists with various tasks and problems that have to be solved to uncover biological patterns. Approaches to visualise this type of data can also be quite diverse since many researchers make a considerable effort to present their findings most convincingly. Yet, the most basic principles of visualisation here generally remain the same. Biological objects are commonly represented as graphical ones of a specific type. They can be lines, dots, circles, squares, etc. Their measured or calculated characteristics (features) define their position, colour, opacity, size, etc. (See Figure 1.1 for possible visual channels.) That is, probably, the most common and the most straightforward way of visualising data, yet there is still a demand for new tools to generate plots of this kind.

Since not all biological data can be visualised in such a manner, here I will mention some most common types of feature data in bioscience.

2.1.1 Omics data

"Omics" is a relatively new and still informal term in biology. According to The American Heritage Medical Dictionary [36], "omics is an analysis of large amounts of data representing an entire set of some kind, especially the entire set of molecules, such as proteins, lipids, or metabolites, in a cell, organ, or organism or any of the fields employing this approach." [37] defines omics the following way:

> "Since the process of mapping and sequencing the human genome began, new technologies have made it possible to obtain a huge number of molecular measurements within a tissue or cell. These technologies can be applied to a biological system of interest to obtain a snapshot of the underlying biology at a resolution that has never before been possible. Broadly speaking, the scientific fields associated with measuring such biological molecules in a high-throughput way are called "omics"."

As one can see, omics studies are one of the primary sources of big data in bioscience and, hence, also the area where the proper visualisation tools may be crucial for understanding underlying biological patterns.

Here, I am primarily interested in data that describe the abundance of molecules rather than in their structure or interactions. Such data can come from various omics fields.

In transcriptomics, the number of transcripts for a given gene in the sample is estimated by converting mRNA molecules into complementary DNA, followed by sequencing the obtained cDNA molecules. The reads are then aligned to the known genome and, thus, the number of transcripts per gene is generated [38–40]. This technique is known as RNA-Seq (RNA Sequencing) [41]. Earlier, the hybridisation-based microarray technique was used to estimate gene expression [42]. For microarray studies, one would need a chip tiled with cDNA probes, each occupying a particular region on the chip. mRNA is first converted to cDNA and then transcribed again in the presence of ribonucleotides labelled with some dye or biotin. The amount of mRNA in the sample is estimated by the observed density of the dye at the region that corresponds to a certain probe [43, 44].

Much of the efforts in proteomics is dedicated to reconstructing protein interaction networks [45]. Though this area also depends on good visualisation tools and benefits from interactivity, I am more interested in data on protein presence or abundance in the sample. Such studies are commonly performed with a combination of liquid chromatography and tandem mass spectrometry (LC-MS/MS, [46]). In this experimental setting, the proteins of a sample are first broken into smaller peptides. LC-step is used to separate peptides based on their retention time to ensure that only a limited number of different peptides enter the mass spectrometer at any given moment. The first MS-step scans the entire sample and produces peaks that correspond to each of the detected peptides. The peak area allows one to quantify the peptide, and the mass of the peptide is defined by the peak's position. The second MS-step is used to identify the peaks. The peptides that enter the second mass spectrometer are broken into even smaller pieces, and their mass spectre is used to reconstruct the structure. Only one at a time kind of peptide can be scanned during the second MS-step, and that's why the initial separation in the LC-step is important. As a result, one gets a dataset with the amount of each detected peptide in all the samples, which also fits my definition of "feature data". The examples of such datasets can be found in [47–49].

Translatomics aims at estimating the translation efficiency of a given protein within the sample [50]. Translatomics studies can be based on sequencing techniques or mass spectrometry analysis. The first type of translatome studies tries to separate free mRNA in the sample from mRNA that is being translated at the moment. These mRNA molecules are then sequenced and aligned to learn what genes are actively translated in the sample. In ribosome profiling (Ribo-Seq) [51], for example, only small pieces of RNA that are protected by ribosomes are retained. The full-length translating mRNA sequencing (RNC-seq) [52] proposes a way to capture entire mRNA molecules that are bound to the ribosome-nascent chain complex (RNC). Eather of the techniques gives a snapshot of translational activities at the moment when the cells were lysed [53, 54]. Another approach to translatome studies is to measure the amount of translated proteins directly, the same way it is done in proteomics studies. One can label and then detect all newly synthesised proteins over some period of time [55] or only nascent proteins at a given moment [56].

Epigenomics studies modification of genetic material that can affect genome accessibility or transcription factor binding efficiency and, as a result, the transcription profile of a cell. The most common epigenetic factors are various histone modifications, and DNA methylation [57]. In this thesis, I am more interested in the latter since it involves "feature data" that require visualisation. In the genome, any cytosine followed by a guanine (a CpG site) can be methylated by DNA methyltransferases. A feature in a methylome dataset is a single CpG site, defined by its position in the genome, and the measured value is the fraction of methylated reads of this cite in the sample relative to all reads. Detection of methylated sites is achieved by bisulfite treatment of the extracted DNA. Bisulfite replaces cytosine with uracil, but only if it is unmethylated. After DNA amplification, the introduced uracil will be replaced with thymine. Now, one needs to find out which of the known CpG sites are intact and which have thymine instead of cytosine. As in transcriptomics, earlier approaches involve methylation chips (such as, for example, the Infinium HumanMethylation450 BeadChip with over 480000 CpG probes [58]), with direct sequencing gaining more popularity over time (known as bisulfite sequencing [59]). Examples of methylation datasets can be found in [60, 61].

This list is in no way complete and only provides examples of the most common areas and techniques. The "omics" field undergoes active development as new technologies come to life, making it possible to quantify more features that describe cell life cycle and reaction to various stimuli. Yet, even from the provided examples of omics "feature data" one can see how diverse and ubiquitous are these data. With the advances of single-cell studies that measure the same features, but now for thousands and even hundred thousands of cells in a sample [39, 40, 61], visualisation becomes even more critical. For such vast datasets, there is just no other reliable way to explore data rather than visualising them with an appropriate tool.

2.1.2 High-throughput screening of biologically active compounds

According to [62], "high-throughput screening (HTS) is the use of automated equipment to rapidly test thousands to millions of samples for biological activity at the model organism, cellular, pathway, or molecular level." It means that most of the techniques discussed in the previous section can also be called HTS. In this section, however, I want to mention another important source of “feature data”: the screening of chemical compounds. In simple words, one can describe this kind of assays as putting together hundreds of chemicals and substrates to check which of them react with each other. Technologically, there are just three things that are required to make HTS possible. The first one is a robot capable of automatically pipetting hundreds of samples on a microliter plate. The second one is a way to measure the read-out automatically. To this end, plate readers that measure fluorescence, absorbance, luminescence, etc. in each well are used. And finally, a reaction in question must be designed in a way that leads to the accumulation of some dye that a plate reader can detect. Overall, HTS is used in biology since the late 1980’s [63], and further technological advances scaled its efficiency significantly.

There are various published datasets generated by HTS assays. Drugs can be tested against cell cultures in cancer studies [64, 65] or to estimate their toxicity [66, 67]. One can also perform an HTS assay to find compounds that bind to a certain molecule [68, 69].

A peculiar thing about these datasets is that there is no fixed separation into “features” and “objects”. Suppose multiple drugs are tested against numerous samples. Should we say that samples are characterised by their interaction with various drugs, or drugs are described by their effect on different samples? Commonly, that is defined by the aim of the study. In [64], for example, the drugs are in the focus of the research and tested cell lines are features to measure their efficiency. At the same time, [65] uses a set of drugs as another way to characterise patient samples in addition to several omics assays. It is also true that in omics studies, one can inverse a dataset and turn features into objects, yet there is some agreement in the community on what is the straight way and what is inversed.

2.1.3 Medical data

Medical data is a collection of records about one’s health state and received medical treatment. Such logs have been accumulated since the dawn of centralised health care systems, yet for such data to become of interest for data scientists, they had to be digitalised first. Digitalisation started to happen more systematically since the late 2000s with the introduction and adoption of electronic health records (EHR) [70, 71]. Although authorities have recommended the use of EHR in many countries, its adoption is still an ongoing process due to various pending issues [72, 73]. Nevertheless, EHR has already been proved to be helpful in clinical research [74], which can also benefit from the right visualisation tools, as any study that involves the exploration of big datasets.

From the point of view of visualisation tasks for medical data, two points should be highlighted.

The health data now are not just digitalised more actively but are also generated at a tremendous rate. People are now using their smartphones, smartwatches, and other wearable gadgets to measure and record their pulse, blood oxygenation rate, physical activity and other health parameters in real-time. Therefore, it is easy to obtain vast amounts of so-called Patient-Generated Health Data from multiple subjects [75, 76]. Here, we are facing the same challenge as with the growing amount of omics data. When there are too many samples and measurements, visualisation turns from being one of many exploration instruments to the only way of getting a data overview. And interactivity here is required to prevent overburdening plots with too much data without omitting it.

Another point refers to more traditional clinical studies. There is some segregation in training and expertise between medical practitioners and researchers. However, a study based on clinical records generally requires collaboration between these two worlds, and people who work in medicine are known to have too much work and not enough time. There are even attempts to come up with recommendations for EHR visualisation that address this issue: lack of time a medical doctor could spare [77]. Thus, a collaboration with medical practitioners poses a visualisation problem. To get medical input in such a study, one should present data in a form that combines the most challenging aspects of exploratory analysis and delivering final results. The person who looks at visualisation should grasp the central message of the data, yet the message itself is still unknown to the one who generates this visualisation. Thus, this highly collaborational field is one of the most demanding for the right interactive tools. And the collaborations themselves are vital if one needs to make an impact on the medical field as fast as possible, as it happened, for example, during the recent Covid-19 pandemic [78, 79].

2.2 Example datasets

In this thesis, various datasets are used to illustrate the presented approaches to visualisation. Here is a list of all these datasets with a short description.

- Data from [39] are based on a single-cell RNA sequencing of a human cord-blood sample with some spiked in mouse epithelial cells. It contains transcriptomes of 8617 cells with 36280 detected genes or non-coding RNA transcripts (before filtering out suspicious cells or too lowly expressed genes). In addition, epitope sequencing (CITE-seq) provides data on the abundance of 13 protein markers for each sequenced cell. The dataset allows one to explore major blood cell types, such as T cells, B cells, Monocytes, etc. Surface proteins provide a robust way to assign cell type labels and thus to estimate the performance of various clustering approaches or dimensionality reduction techniques. In this thesis, I will refer to this dataset as “cord-blood data”. The data are available at Gene Expression Omnibus (GEO) via accession GSE100866

- In [38] samples from 19 patients with oral cancer have been taken. From each patient, three tissue samples were obtained: one of the normal oral mucosa, one of epithelial dysplasia (i.e., abnormal but not yet malignant tissue), and one sample of the tumour. RNA from each of the 57 samples was extracted and sequenced to estimate the expression of 58037 genes and non-coding RNAs. These data are a typical example of the ones used in differential expression studies to find genes that are up- or downregulated in a specific type of tissue or cell type. In this thesis, I will refer to this dataset as “oral cancer data”.

- [53] is a study on co-evolution of translation and transcription. To check whether there is any connection between evolutionary changes in gene expression and translation efficiency, three samples of three tissue types (brain, liver and testis) were taken from six species (human, macaque, mouse, opossum, platypus and chicken). Each sample was split in two, and for one part, RNA was extracted and sequenced. In the other part of each sample, RNA was lysed except for the pieces protected by attached ribosomes (Ribo-Seq [51], see Section 2.1.1). The remaining RNA molecules were also sequenced, thus, providing a snapshot of the ongoing translation in the sample. To make a comparison between the species, only 1:1 orthologues across all the studied species were left, leaving 5060 genes and in total 103 sequenced samples (50 RNA-Seq samples and 53 Ribo-Seq samples). This data can be seen as an example of a non-typical study that requires a customised visualisation approach. In this thesis, I will refer to this dataset as “translatome data”.

- [80] focuses on the study of the developing murine cerebellum. Samples of the forming cerebellum tissue were taken at 8 embryonal stages from e10 to e17, followed by four postnatal stages between P0 and P21. Each sample is supported by a biological replicate from a separate pregnancy. Samples were processed to obtain single-cell transcriptomes, yielding 400-3000 sequenced cells per sample. Single-cell RNA sequencing of a developing tissue allows one to study developmental trajectories of differentiating cell types. As in most single-cell studies, dimensionality reduction and further visualisation of the uncovered trajectories plays an important role. Here, I use only three samples for demonstrational purposes: two replicates from day e13.5 and one from day e14.5. In this thesis, I will refer to this dataset and “murine cerebellum data”. The data can be downloaded from the European Nucleotide Archive under the accession number PRJEB23051.

- As an example of a high-throughput drug screening experiment, I used data from a study on drug interaction [64]. Yet, for the sake of simplicity, only a subset of generated results is used. Instead of interactions between drugs, I am using only single drug tests. Overall, 527 drugs were tested against 21 pancreatic cancer cell lines. The viability of cells in the presence of each drug was evaluated with a cell viability [81] and a cell toxicity [82] assay in 5 different concentrations for each drug. The obtained values were used then to calculate a Drug Sensitivity Score (DSS, [83]) that is used as a measure for drug’s effect on a cell culture. This dataset is used to explore the possible benefits of interactive visual exploration of a typical pipeline output. In this thesis, I will refer to this dataset as “drug screening data”.

2.3 Tools and dependencies

2.3.1 The R programming language

The Programming language R [84] is commonly used for statistical computations by bioinformaticians and biologists. It is free and is accompanied by an integrated development environment “RStudio” [85], which also offers a free, fully-functional version and can be used for computations on a remote machine (with the “RStudio Server” version).

R’s high-level syntax is quite different from many other programming languages, which is often considered a disadvantage since the learning curve for a beginner can be steep. However, the differences stem from the fact that the creators of R had a particular purpose in mind, which is data analysis [86]. Therefore, the syntax is subjected particularly to this single goal: to facilitate statistical computing. And while it can confuse newcomers from other programming areas, beginners with a biological background can find R easier to learn and use than existing alternatives (such as Python [87] or Julia [88], for example). And this compromise probably is one of the reasons behind R’s popularity: Experienced programmers are more likely to invest much time into learning a new language than those who are simply looking for a more professional alternative to processing data with *Microsoft Excel* or other similar software.

Another advantage of R is its comprehensiveness. Since it has already gained vast popularity among statisticians in academia and industry, any new algorithm is likely to be implemented as an R package. The Comprehensive R Archive Network (CRAN) offers more than seventeen thousand packages (as of March 2021) and is very easy to contribute. There is also a separate repository for expressly biostatistical packages called “Bioconductor” [89], which shows how important R is for research in bioscience. For more experienced users R offers packages such as “reticulate” [90], “rJava” [91], “Rcpp” [92–94], and others to exploit functionality of other programming languages.

R also has some disatvantages. It can be relatively slow and has some memory issues since it loads all variables (including full datasets) into the operative memory. Yet, there are community efforts to solve these and other occurring issues. For example, “tidyverse” [95] facilitates and speeds up work with scientific data by offering a systematized approach to treating datasets and reimplementing some functionality. “future” [96] introduces parallel computations that are not implemented in base R. “hdf5r” [97] provides an interface for working with very large datasets.

Overall, R is an instrument for everyday use for many bioscientists all over the world. As such, I believe that developing visualisation tools in the form of R packages can greatly impact data analysis in bioscience. Though R provides a number of established packages for data visualization both static (“ggplot2” [98], “pheatmap” [99], etc.) and interactive (“shiny” [29], “plotly”[30], etc.), I feel like there is still plenty of room for improvements.

2.3.2 The JavaScript programming language

Though, as described in Section 2.3.1, R is a prevalent choice for people interested in data analysis, its possibilities to develop a graphical user interface (GUI) are limited. Therefore to utilise the full power of interactivity, one should look for some other tool. Especially since, as was mentioned above, R offers several ways to employ other programming languages.

Here, one may want to have a closer look at JavaScript. In the same way as R was designed for data analysis, JavaScript was introduced to add dynamic behaviour to previously static web pages. Nowadays, various JavaScript (or TypeScript, which is an extension of JavaScript) libraries, such as “Angular” or “react.js”, hold leading positions when designing a front-end of a website. Currently (March 2021), above 97% of existing sites are using JavaScript at the client-side.

All this makes JavaScript a natural choice when one wants to make an app to combine interactivity and data visualisation without much care about complex computations and performance (for this, we will use R). In addition, HTML pages containing JavaScript can be run in any modern browser without installing additional software and loaded directly in the viewer of RStudio. Potentially, such an app can also be converted into a single HTML page that contains all the data and functionality. This page can then be used as a supplement to a paper or sent to collaborators.

To my knowledge, there is no direct link between R and JavaScript, e.g. a way to directly utilise JavaScript functionality from R in the form of a native R interface. However, R can open web pages in a browser or the RStudio Viewer. If the loaded web page contains some JavaScript, it will be executed automatically, and all in-built reactions to user events will be in place. Therefore, to be able to use JavaScript for interactive visualisation of data that are loaded into the R session, we need two things:

- A way to generate a web page, populate it with the relevant data, and open it on request from an R session.
- A maintained double-sided connection between the R session and the opened web page. R should be able to react to user interactions with the web page by starting new calculations or storing information for further use. It should also push new data or results to the web page without restarting the app. Even though many JavaScript-based interactive visualisations do without such a connection, I feel that it is important for genuinely customisable apps to facilitate exploratory analysis.

2.3.3 Web server

Web servers are generally used for generating web pages and sending them back to the client. The term "server" often refers to a stand-alone piece of computer hardware; however, it can also mean a program. By definition, anything that provides functionality to other scripts and hardware is a server, whether it is a script itself or a separate device. There are various types of servers based on their purpose: computing, mail, file storage and sharing, proxy, etc. Web servers are the ones that host websites and, as such, are the ones that most of us interact with daily.

To explain what is a web server and how it works, let us go step by step through what happens when someone tries to open a web page *myWebsite.com/somePage*.

First of all, computers connected to the Internet do not identify each other by name. A numerical label called an IP address is used instead. Internet Protocol version 4 (IPv4, [100]) standard defines an IP address as a 32-bit number, which is usually displayed as a set of four numbers from 0 to 255 separated by dots. In version 6 (IPv6, [101]), an IP address uses 128 bits and is commonly displayed in hexadecimal format as eight quartets separated by semicolons. Therefore, *myWebsite.com* needs to be resolved as an IP address. To this end, the local computer exchanges messages with one of 13 name servers located all over the world to comprise the Domain Name System (DNS). The IP address of a name server must be known in advance and is usually a part of the operating system settings. When the IP address is known, the local computer can establish a connection with the web server, hosting the requested web page.

Internet Protocol ensures message exchange between any two machines worldwide by defining how information packages travel through hubs and routers. However, successful communication also requires a standardised form of the messages themselves. IP defines their structure to some extent, but only to ensure that the message reaches the recipient. The body of the message can contain arbitrary data and has to be specified by another set of rules so that the two remote computers can understand each other. The two most common protocols for that are TCP (Transmission Control Protocol, [102]) and UDP (User Datagram Protocol, [103]). TCP is more reliable and more commonly used: It makes sure that each piece of the sent data reaches the recipient, and the receiving application will interpret them in order. Yet, it is also more heavyweight and supports only communications between two endpoints. UDP is more straightforward and, therefore, more lightweight. It does not ensure reliability but can be used for broadcasting the same data to multiple recipients.

The IP protocol belongs to the so-called Network layer of the Open Systems Interconnection model (OSI model, [104]). It is responsible solely for delivering a piece of information to the specified address. TCP and UDP are both parts of the Transport layer. As a part of the Network layer they send not just some bits of information but a specified message that will be received and processed on the other side. For instance, these protocols can split messages into pieces and assemble them back upon receiving. TCP, in addition, can check the integrity of the received message and request the sender for the missing parts, if any. After an IP/TCP connection is established, we can be sure that, in our example, the browser can send and receive messages to and from the server.

The next step is to define how exactly the browser will ask the server for the *somePage* at *myWebsite.com* and in what form it will get it. To this end Hypertext Transfer Protocol (HTTP, [105]) or its secure encrypted version HTTPS is used. This protocol is request-response based. It means that the client (the browser) makes various requests to the server, and the server responds to them with data (plain text, HTML, images, data, etc.). The received responses are processed by the browser and displayed to the user.

Therefore, the steps to open *myWebsite.com/somePage* are the following:

- *myWebsite.com* is resolved into an IP address. To do that, the browser first checks the cached list of names and, if *myWebsite.com* is not there, sends a request to the DNS server specified in the system settings.

- A TCP/IP connection is established between the local computer and the server hosting the website (based on the server's IP).
- The browser sends an HTTP GET request for *somePage* and displays the HTML code that the server sends in response.
- If the page contains some additional resources (such as stylesheets, images, scripts), the browser will request them in separate GET requests.

Note that HTTP (or HTTPS) can be used not only to ask the server for the content but also to send some data to the server with a POST request. HTTP also defines other types of requests (PUT, DELETE, OPTIONS, TRACE, CONNECT, PATCH and HEAD), but POST and GET are the most common ones.

With all this, if we have a server app running in the background, the R session can send some data to it and request the generated HTML page with the JavaScript code to visualise the data and ensure interactivity. Previously, we talked only about communication between remote computers over the Internet, but the server app can run on the same machine as the R session. In modern operating systems, the *localhost* name is used to define the local computer. Any messages to the *localhost* will not be sent outside but instead will be received by the host itself via the loopback network interface. From the browser's point of view, there is no difference between connecting to a local or a remote server. For this project, it means that the visualisation apps based on the message exchange with the local server can also be easily accessed from the outside. Therefore, it is effortless to make such apps publicly available.

The last thing we need to take care of is to make sure that the exchange of messages between the app and the R session does not interfere with other network services. Here, we need to take a step back to the transport protocols described above. Both TCP and UDP introduce a logical construct that is called port. A port is identified by a 16-bit number and defines a communication endpoint. Ports allow a computer to maintain multiple independent connections simultaneously. Messages sent to different ports can be processed in different ways. For example, several network apps (such as browsers, messengers, etc.) can run simultaneously by using different ports and not knowing of each other. A server can use different ports for different kinds of communication (mailing service, HTTP requests, file transfer via the File Transfer Protocols, etc.). So it is enough to make sure that our app uses a free port and the R session knows its number.

2.3.4 Websocket connection

As mentioned in the previous section, HTTP is a request-response based protocol. It means that the server (the R session in our case) can only respond to requests from the client (the browser) but not send data on its own accord. Therefore it is not suited for a double-sided conversation. For the R session to have any effect on the web page, the latter has to send requests continuously, asking if there are any pending changes.

However, in 2011 WebSocket protocol [106] was introduced. Like HTTP, it is based on top of TCP, and the connection is established by sending a specific HTTP request and getting a response. Therefore a Websocket connection can use the same port as the initial HTTP connection. But unlike HTTP, a Websocket connection is fully double-sided. The client and the server can send messages at any moment, and they do not need to be followed by responses.

A WebSocket connection can also be faster than HTTP. Any exchange of a request and a response via HTTP or HTTPS happens within a separate TCP connection, which is closed after the client receives the response from the server. A new connection must be established for the subsequent request, even if it takes place immediately after the first one. The WebSocket protocol, on the contrary, maintains the same TCP connection.

With an established WebSocket connection, not only the app can trigger new calculations in R, but also the R session can at any moment execute JavaScript commands on the web page and, thus, keep the visualisation up-to-date.

2.3.5 "httpuv" package

All the functionality, mentioned in Sections 2.3.3 and 2.3.4, is already implemented in R as the "httpuv" package [107]. This package relies on the ability of R to run C and C++ code with the "Rcpp" package [92–94]. It is build on top of "libuv" and "http-parser" C libraries and implicitly, via the "websocket" R package [108], depends on the "websocketpp" C++ library.

"libuv" is used to run a server process and enables asynchronous handling of incoming and outgoing messages. "http-parser" facilitates work with

HTTP requests and responses by parsing them into more suitable data format. “websocketpp” enables double-sided communication via the WebSocket protocol.

Directly from an R session, “httpuv” can start a TCP server that will listen to the specified port. It is possible to define an R function to process all HTTP requests coming through the port and generate custom responses. “httpuv” is also capable of establishing and maintaining any number of WebSocket connections, calling R functions in response to incoming messages, and sending messages via a specific WebSocket.

With all the abovementioned functionality, “httpuv” is a perfect foundation for this project.

2.3.6 D3.js

On the JavaScript side of the project, there is also no need to start from the basics (so-called “Vanilla JS”). JavaScript is well known for a large number of various libraries, from minimalistic helper packages for specific functionality to huge frameworks that introduce new syntax principles and have communities and plugins of their own. There is even an entire programming language, TypeScript, that is an extension of JavaScript.

One of such major JavaScript frameworks, developed for interactive data visualization, is *D3.js* [24]. “D3” stands for Data-Driven Documents. D3 offers not just functionality but the entire concept of how to approach the problem of data visualisation. In D3, every instance of the visualised dataset is bound to a single element of the plot. After that, its characteristics are defined based solely on the values stored in the corresponding data instance in a user-defined fashion. Therefore, any changes in the dataset can easily be traced and lead to changes in the visualisation.

To understand how D3 works, one needs to know how an HTML document is structured. Any HTML page can be represented with a Document Object Model (DOM). DOM is a tree-like structure, where each node is a specific element of the document. For example, a table, a single cell of a table, a paragraph, a navigation menu, or a single line of that menu. More general elements can contain others. For instance, an entire table is a DOM element. All its rows are also DOM elements and children of the table element. In turn, each row contains individual cells that are also DOM elements. Each has a set of attributes that define its properties, such as border width, size, font, actions to perform when clicked, a name or an ID. In addition to the attributes, CSS (Cascading Style Sheets) offers a way to set styles for each element. Generally, styles are used to define all kinds of visual decorations, while attributes set other options. But in fact, the border between them is quite vague, and the two often overlap. In an HTML document, DOM is defined by tags, where each tag corresponds to a DOM element, and the tree-like structure is achieved by putting children tags inside their parents.

Introduced in 2014, HTML version 5 supports Scalable Vector Graphics (SVG), which plays an essential role in data visualisation tasks. SVG is a graphics format that is also based on DOM. Its base elements are various graphics primitives: circles, rectangles, custom paths. Their attributes define colour, border width, opacity, etc., and their position inside the parent SVG element. In practice, there is no difference in functionality between SVG elements and HTML tags. They are defined and modified in the same fashion, which is important to *D3.js* since most of its functionality concerns various manipulations with DOM elements.

D3 is based upon three major building blocks.

- A *selection* is simply a collection of one or several DOM elements. Selections are defined with CSV selectors (using a tag name, a class name, an element ID, etc.) and allow easy access to attributes and styles of the selected elements. A very similar concept of selections exists in a well-known JavaScript framework, "jQuery", and thus is familiar to many of those who worked with JavaScript.

- *Transitions* offer a simple in-built interface to trace any changes of styles and attributes of the elements inside a selection. The changes are displayed as a smooth transformation from one state to another. In this way, the user can easily trace any effect on the visualised data that his or her interaction caused.

- *Data binding* is the core concept of D3, as follows from its name. The idea is that every element of any selection can be bound to a specific instance in the dataset. One can define what should happen if the instance is removed or a new one is added to the dataset (for example, to remove the corresponding element and add a new one). D3 also offers an easy way to access the data bound to the element and use them to specify the element's attributes and styles.

Besides the very basic functionality linked to its central concepts, D3 also offers several primitives that can be used for data visualisation. For example, continuous and categorical scales can be used separately or linked to axes, some predefined layouts to make force networks, parsers to process the most common data types, and many other plugins. Yet, even with all the diverse possibilities and functions, it is still required to have some coding experience with JavaScript to create D3 visualisations.

Chapter 3

Results

In this section, I will go through the three packages developed in the course of this project. The packages tackle the problem of combining R and JavaScript for interactive visualisation from different sides. They are applicable to real-life practical tasks and represent each its own idea of how to approach visualisation challenges. All the packages described in this section are written for the R programming language ("rlc" also provides a JavaScript interface). They are open source and are freely available on CRAN and GitHub.

- I start from the "jrc" package (Section 3.1). It is not directly intended for visualising biological or any other kind of data (though still capable of it), but rather designed to be used by developers as a basis for other packages. It serves as a bridge between an R session and a web page and allows one to run any JavaScript code from the R session and vice versa.

- The second package in the scope of this thesis is "sleepwalk" (Section 3.2). It is a simple interactive tool to explore distortions in distances caused by dimensionality reduction techniques. It utilises interactivity to help the user to explore the effect of the dimensionality reduction on the neighbourhoods and make his or her own conclusions about it. "sleepwalk" is written almost entirely in JavaScript and only waits to receive data from the R session with the underlying "jrc" package. In this way, the app does not rely on an active connection and can be easily saved as an HTML file. However, such a use of the predefined JavaScript template makes it hard to customise the app.

- Finally, Section 3.3 tells about LinkedCharts. This toolbox is available both in R (as the "rlc" package) and in JavaScript (as the *linked-charts.js* library). LinkedCharts is a general-purpose plotting library for generating interactive apps. It is not bound to a specific task and only provides the user with a set of plotting primitives and a way to link them together to ensure interactivity. As "sleepwalk", it also is based on the "jrc" package, but unlike it, it is based on a clear task separation between the utilised languages. All the data processing happens exclusively on the R side, while JavaScript only visualises data and reports the user's activity to the R session. This separation requires a constant message exchange between the R session and the web page, and as a result, the app cannot be directly encapsulated within an HTML file. Instead, the provided JavaScript interface should be used. (See Section 3.3.3 for more details.) However, such an approach ensures flexibility since R users can fine-tailor the behaviour of the app without ever touching JavaScript.

3.1 The "jrc" package

Unlike the two other packages, described further in this thesis, "jrc" is intended for people with a computer science, rather then a biological background. Therefore, this section is much more technical then the rest of the thesis.

For all further work to be concentrated solely on visualization tasks, we first decided to establish an efficient bridge between the R and JavaScript programming languages. As explained in Section 2.3.5, the existing "httpuv" package solves the technical part of the problem, employing several C and C++ libraries. Nevertheless, "httpuv" is still a very low-level tool. It provides a possibility of communication but does not regulate the communication itself. A WebSocket connection can be used to exchange messages between an R session and a web page, but the rules on how to process the messages still have to be defined on both sides. To this end, I devised a package called "jrc", which stands for which stands for **J**avaScript-**R** **C**onnection. It is based on the "httpuv" package, and its main goal is to allow for exchange of any JavaScipt code between a running R session and a web page. Since "jrc" is supposed to work on both sides of the connection, it consists of two parts: R and JavaScript. A TCP server is set up by the "httpuv" package and is defined in a way that it inserts the JavaScript part of the "jrc" package to any served HTML page, thus ensuring the package's full functionality. More details on web servers and TCP/IP connections can be found in Section 2.3.3.

To give a feeling of how "jrc" works, here is a very basic example:

```
1  k <- 0
2  openPage()
3  sendCommand(paste0(
4     "button = document.createElement('input');",
5     "button.type = 'button';",
6     "button.addEventListener('click', function() {",
7        "jrc.sendCommand('k <<- k + 1')",
8     "});",
9     "button.value = '+1';",
10    "document.body.appendChild(button);",
11    collapse = "\n"))
12 closePage()
```

This example opens a web page with a single "+1" button. Each time the button is pressed, it increases the value of the variable `k` by one. At any moment, the variable and its current value can be accessed in the R session. Line 1 of this example sets a default value of `k`. Lines 2 and 12 initialise and close the app, respectively. The most important part of this example app, in particular, and the "jrc" package, in general, is in Lines 3 and 7. There, one can find the `sendCommand` function that is responsible for sending custom code from the R session and back and executing it on the other side. `sendCommand` from Line 3 takes a chunk of JavaScript code and runs it on the web page, adding a button. `jrc.sendCommand` (Line 7) is a JavaScript function that runs a piece of R code in the R session, increasing `k` by one. The `sendCommand` function and its alternatives in the "jrc" package are described in Section 3.1.1. Lines 4-10 are responsible for adding the button to the web page. The JavaScript code here may seem overly lengthy. However, partly it is caused by the use of "Vanilla JS" for the sake of generality. Modern JavaScript frameworks can do the same with much more concise commands. Alternatively, one does not need to use JavaScript for static elements. "jrc" can open a pre-made HTML file with all the elements, such as this button, already added to it.

```
<input type="button" value="+1"
   onclick="jrc.sendCommand('k <<- k + 1');" />
```

Having a *myFile.html* file with the code above, one can get the same app simply by running

```
k <- 0
openPage(startPage="myFile.html")
```

3.1.1 Message exchange

Messages between R and JavaScript are exchanged via a WebSocket connection. (See Section 2.3.4 for more details.) In "jrc", messages are sent as plain text, then they are interpreted and processed based on their type. Overall, there are four types of messages:

- A *command* (COM) is a piece of R or JavaScript code that is sent to the other side of the WebSocket connection and directly evaluated there. It is the most straightforward way to control a web page or an R session. The code is passed as a plain text and executed as-is, no matter the result. It is up to the user to make sure that the code can run without errors.

- A *function call* (FUN) contains a name of an R or JavaScript function, a list of arguments and a name of the variable where to store the returned value if any. If a function with the specified name is defined, it is called upon receiving the message with the provided arguments. In a request to the R session, one can also set a package where to look for the function. The package must be already installed but not necessarily loaded.

- *data* (DATA) are transferred in JSON format. Such a message also contains a name of the variable where to store the data. Since browsers are not usually effective in storing large amounts of data, it is recommended not to send big datasets at once from the R session. It will make an app more robust if necessary data are transferred on demand.

- *HTML* is the only type of message that can go only in one direction: from an R session to a web page. Such a message contains plain HTML code that will be appended to the "body" element of the page. Even though JavaScript code is allowed inside a "script" HTML tag, this code will not be processed by the browser. Therefore this type of message cannot be used instead of COM messages.

To handle each of these four message types "jrc" offers an R and a JavaScript function. R functions are `sendCommand`, `sendData`, `callFunction` and `sendHTML`. JavaScript functions are `jrc.sendCommand`, `jrc.sendData`, `jrc.callFunction`. The full list of their arguments can be found in the package's documentation on CRAN. JavaScript functions are described together with the corresponding R functions in the "Details" section.

3.1.2 Multiple sessions

There are two ways of using an interactive app. It can be run locally when the same person runs the R script and interacts with the web page. In this case, there is typically a single WebSocket connection, through which all the messages are sent and received; all the messages are trustworthy since they are generated by the user who controls both sides of the connection; and at any given moment, there is just one state of the app that has to be stored.

Things change when we put the app online. Now several users can access the app simultaneously. It means that for each message, "jrc" has to know where it comes from and where to send the reply. In addition, each connection may require its own set of variables to correctly react to the user's actions. Finally, if the app is published outside of some protected network, the incoming messages can be harmful. "jrc" tackles all three issues. The security measures are described in Section 3.1.4. This section is about the inner structure of the "jrc" app and its role in managing multiple connections to the same app (several opened web pages).

Partially, the problem is already solved by the "httpuv" package. Multiple connections can be established, and corresponding handles are generated. The storage and usage of the handles, however, is governed by the "jrc".

An entire "jrc" app is encapsulated inside an "R6" class object [109]. This object of class `App` contains full information about a jrc app. It stores the server handle and information about all active connections. Methods of this object are used to monitor and force close connections, control data accessibility and the security permits of the app within the current R session. All the essential for the app's functionality information is stored within this single object, making it possible to have several apps running simultaneously within the same R session.

Similarly, all the information that is related to a single connection to a web page is stored in a separate object of class `Session`. Instances of this class are initialized whenever a new WebSocket connection is established and are stored inside the `App` object. After a connection is closed, the corresponding object is removed from the memory. `Session` holds a connection to the web page, processes all the incoming and outgoing messages, stores local variables and controls memory usage.

`Session` objects are completely separated from each other and the corresponding `App` object. There is no shared information and no way for one session to influence another. The `App` object, however, has complete control over each session. An object of class `Session` can be retrieved only through the App object via the randomly generated ID that is assigned to each session upon initialization. Therefore number and functionality of simultaneous connections are limited only by resources available to the R session and the app's settings. (See Section 3.1.4 for more details.)

However, 'jrc" is also intended to serve as a basis for easy-to-use interactive apps for everyday exploratory analysis. It can be too much to expect people to learn about the "R6" classes, which are not part of the casual R usage for the data analysis, or to keep in mind the structure of a jrc app. Therefore, the "jrc" package also provides a set of wrapper functions for local usage. If there is only one active connection (for example, to the RStudio viewer), the app can be fully controlled with these functions that internally access all the required R6 class objects. In this way, the "jrc" can be used simply as a bridge between the RStudio and a single opened web page without any knowledge about Sessions and Apps. The wrapper function can also control multiple sessions, but then they will need to get a session ID as one of their arguments.

3.1.3 Local environments and function evaluation

Briefly mentioned in Section 3.1.2, but very important for understanding principles upon which the "jrc" package is built, is non-standard function evaluation [110]. In "jrc", it solves the problem of local variables, i.e. the variables defined for each client session to store some specific information, such as the app's current state for the given user.

Each object of class `Session` has an environment assigned to it. I am going to call them local environments. Another environment is assigned to the entire app (app environment). The app environment is a parent of all the local ones. By default, the environment where the app was initialized is used as an app environment, but the app's settings can change it.

All session variables are stored inside the local environments. Each session can access its own variables, data from the app environment and its parents, if any. There is no way for one session to get to the variables of another session. One can also protect global variables by masking them (creating a local variable with the same name) or by orphaning the app environment. In the latter case, the entire app will not affect anything outside of the app environment.

A specific procedure was designed to control how the messages incoming through the WebSocket are processed to achieve this effect. As it was mentioned before, it is the `Session` object that is responsible for receiving and sending messages. This object has the most direct access to the corresponding local environment.

- Commands are evaluated inside the local environment. Therefore both search for a variable and assignment occurs according to common R principles. Any assignment with the usual `<-` operator happens in the local environment, and `<<-` operator searches for the variable with the given name along the parent chain. A default R search path is used to read a variable, e.g. R searches for a variable with the specified name first in the local environment, then in the app environment and then in its parents, until the variable is found or the global environment is reached.

- Data are assigned to the variable with the specified name in the local environment if there is a session variable with such a name and in the app environment otherwise.

- For functions, "jrc" changes their enclosing environment to the local one, unless the function is defined inside some package's namespace. The arguments are evaluated inside the local environment, and the result is assigned the same way as the incoming data.

Even though non-standard function evaluation may lead to some confusion and unexpected results for the user, especially when trying to solve some non-trivial tasks, in "jrc" it is still employed to ensure simplicity for the users with less experience in programming. Message evaluations and function calls are organized to make it possible to use the same app locally and as a public app with only minimal changes in the code. Specifically, in most cases, one should only provide all the session variables with their default values to the `sessionVariables` argument of the `openPage` function (which starts any "jrc" app). Everything else will be taken care of automatically.

3.1.4 Security restrictions

If a jrc app is made public, anyone who can open it in a browser also gets access to the WebSocket connection to the R session running somewhere on a server. The client can then send his or her own messages that will be processed by the R session, as are those that come from the app. And the R session, in turn, has a wide range of possibilities to access local files and interact with the operating system. The problem is especially relevant for the direct code evaluation, but other "jrc" functions (enumerated in Section 3.1.1) can also be maliciously misused by someone creative. Therefore some security restrictions have to be set for an app. Though the "jrc" package provides similar capabilities for both ends of the WebSocket connection, security limits are, in fact, asymmetric.

At any moment, the R session has complete control over each opened connection. Specifically, any JavaScript code can be evaluated on the web page, local variables can be inspected and manipulated, information on the duration and activity of the connection can be obtained, and the connection can be closed at any moment. This is permissible because a jrc app is incapable of doing anything more than running some JavaScript code in a browser. Thus, the privacy of the person who requests a page from the jrc app is protected by the in-built browser limitations for JavaScript, which are quite strict.

On the other hand, many of the commands incoming from the web page to the R session must be authorized before execution. Specifically, any custom code always requires authorization. As for the function calls and variable assignments, one can specify a list of functions that a web page is allowed to call and variables that may be overwritten. Everything else will require manual authorization. There are also limitations for the local files that the server can access: by default, these are only files that are stored in the root directory of the server. One can also add other directories to the list of allowed directories. Attempt to access files outside of the specified locations will cause a ”403: Forbidden” response.

3.2 The “sleepwalk” package

In Section 3.1, I have described the “jrc” package as a foundation for other packages for interactive visualisation. Here and in Section 3.3, I am going to give examples of what kind of packages can be made on top of “jrc”. First, we will talk about “sleepwalk”: a tool to interactively explore 2D embeddings of high dimensional data.

The text and figures of this section (the entire Section 3.2) are taken from [111] with only minor changes. The paper is currently distributed under the Creative Commons Attribution-Non-Commercial 4.0 International License and was originally written by myself.

3.2.1 Distances transformation in dimensionality reduction

Whenever one is presented with large amounts of data, producing a suitable plot to get an overview is an important first step. So-called dimension reduction methods are commonly used if the data have a matrix shape. In Section 2.1, I refer to this kind of data as “feature data” and also provide examples of the assays that produce them. When working with feature data, it is common practice, especially if the dataset contains many objects, to perform principal component analysis (PCA) on a suitably normalised and transformed feature matrix and then plot the objects’ first two principal components as a scatter plot. Of course, PCA has more uses than just providing such an overview plot (See [112] for a primer.), but nevertheless, the user’s expectation is often simply that objects with similar feature profiles should appear close together (“cluster together”), while objects with substantial differences should appear farther apart. PCA’s popularity in biology notwithstanding, the literature offers many methods designed specifically with this goal in mind, with the best-known classic example perhaps being classical multidimensional scaling (classical MDS, also known as principal coordinate analysis, PCoA), Kruskall’s non-metric multidimensional scaling [113] and Kohonen’s self-organising maps (SOM) [114].

The recent rapid progress of single-cell RNA-seq methods, now enabling the measurement of expression profiles of thousands of individual cells in

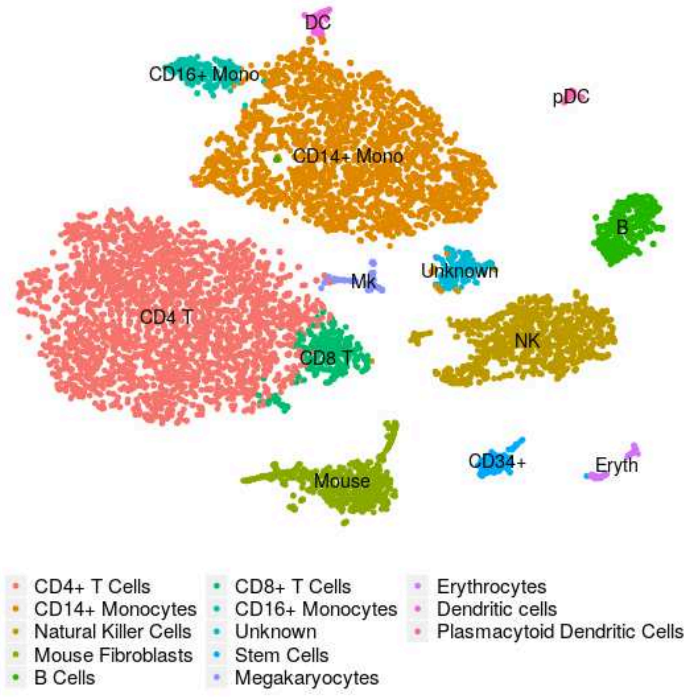


Figure 3.1: Example of a t-SNE plot: These are cord-blood mononuclear cells studied by [39]. The embedding and the assignment of cell types have been taken from the Seurat [115] tutorial that uses this dataset as an example [116]. See Section 2.2 for more details.

a sample, has renewed biologists' interest in dimension reduction methods. Here, t-distributed stochastic neighbour embedding (t-SNE, [117], Figure 3.1) and Uniform Manifold Approximation and Projection (UMAP, [118]) have become a de-facto standard. Other dimension reduction methods, developed specifically for single-cell RNA-Seq include Destiny [119] (a method based on diffusion maps [120]), the Monocle methods [121, 122], DDRTree [123] and more. (See [124] for a review.) Due to the popularity of these methods for single-cell RNA-Seq studies, in this section, I will use this type of datasets as examples. Therefore, if not stated otherwise, we are discussing feature data with individual cells as objects and gene expression values as features. Section 3.2.7 goes through other possible applications of the "sleepwalk" package.

These varied methods have been developed with different design goals: For example, some methods strive to primarily preserve the neighbourhood, others to represent the overall structure or larger-scale relations. Nevertheless, when using any of them in the field of single-cell transcriptomics, the practitioner's primary expectation is usually that cells depicted close to each other or within the same apparent structure or cluster have more or less similar expression profiles. In contrast, cells depicted in different regions of the plot or in different structures are more different. In other words, it is the preservation of neighbourhood relationships that is of importance. The term "neighbourhood" should here be understood as follows: We consider a high-dimensional space, the so-called feature space, in which each dimension corresponds to one gene and each cell is represented by a point, whose coordinates along the many dimensions are given by the expression strength of the corresponding gene. Two cells with similar expression profiles will hence have similar coordinates and thus will be close to each other in feature space. Around each cell, we can imagine a hypersphere of nearby points and consider all cells within the hypersphere as neighbours.

Any attempt to provide a two-dimensional representation of the neighbourhood relations in this high-dimensional space will have to face what in [117] is called the "crowding problem". The volume of a high-dimensional sphere is exponentially larger than the area of a two-dimensional disk. Therefore, a cell can easily have many more close neighbours in feature space than cells can be drawn within a sufficiently small circle around the point representing the cell in two-dimensional space.

This is, of course, not the only obstacle in achieving a faithful two-dimensional representation of feature space, and the many possible kinds of distortions have been widely discussed in the literature. (See e.g. [125] and [126].) However, in single-cell sequencing, this is of particular relevance: given a dimension-reduced representation such as a t-SNE or UMAP embedding, how can we know for a specific cell of interest how far its neighbourhood reaches? Knowing this is of paramount importance to correctly interpret an embedding.

As a part of this project, we devised the "sleepwalk" package, an interactive tool that provides an intuitive solution to the task just outlined.

It works as follows: The user provides an embedding, i.e., the two-dimensional coordinates output by a dimension-reducing method, as well as information on the distances between cells in feature space in some suitable

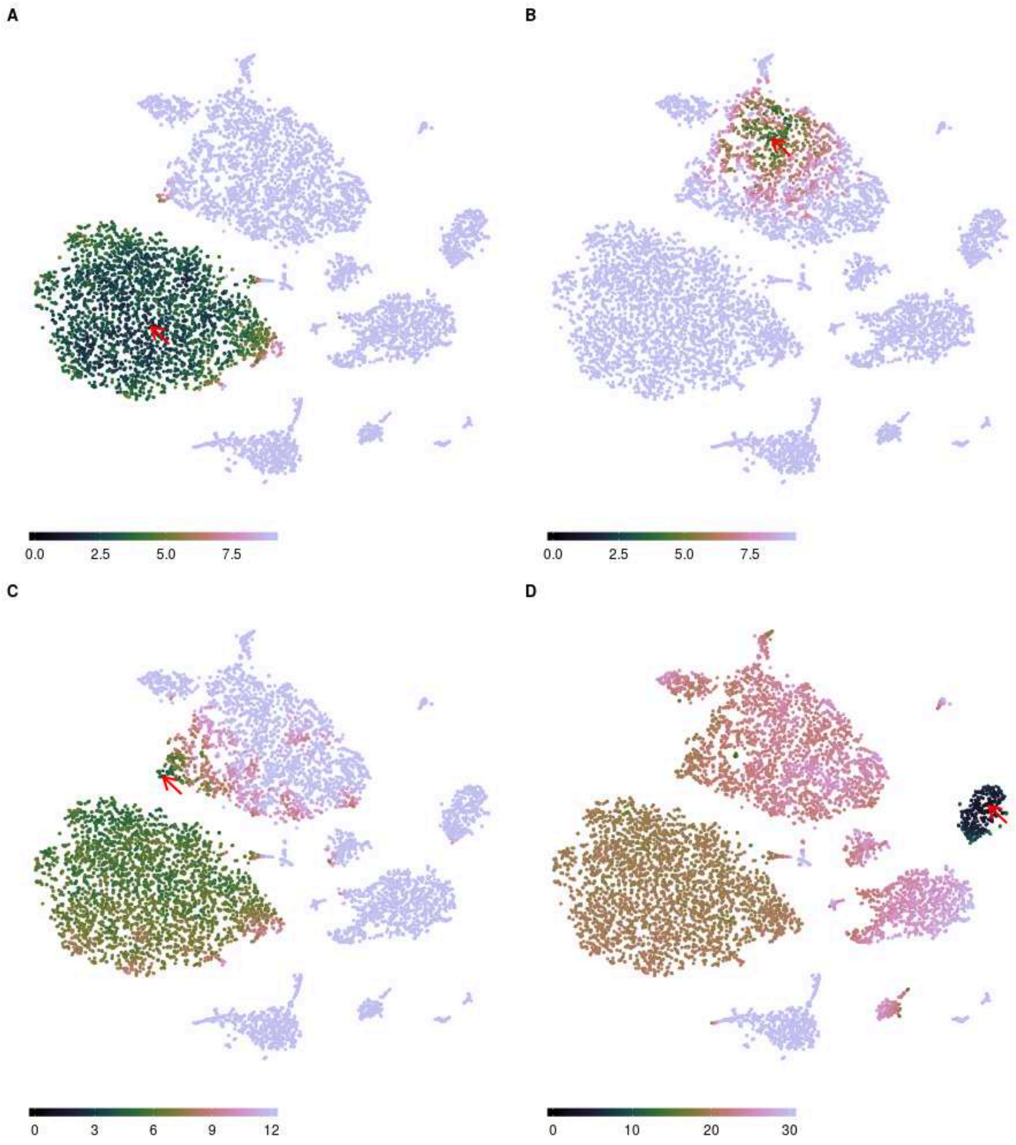


Figure 3.2: (Caption on the following page.)

Figure 3.2: The "sleepwalk" app, being used to explore the t-SNE rendition of the cord-blood dataset from Figure 3.1. The plots here are snapshots of a running "sleepwalk" app. The red arrow shows the current mouse position. (A) By moving the mouse cursor through the embedding, we find, e.g., that the CD4+ T cell cluster is very tight and homogeneous, as can be seen from the fact that all cells show a colour indicating that they are all close to each other. (B) The monocyte cluster, in contrast, shows much more heterogeneity when comparing the colouring at the same colour scale: now only a few monocytes are coloured green and are hence as similar to the cell under the mouse cursor as most of the T cells were in (A). (C) Placing the mouse on this small tip of the monocyte cluster reveals that the cells there are more similar to the T cells than to the other monocytes, indicating that the cluster boundary might be inaccurate in both the t-SNE rendition and the SNN clustering on which the Seurat workflow's cell-type assignment is based. (D) With the colour scale set to a wider distance range, we can assess similarities *between* clusters: As expected, B cells are somewhat similar to T cells, less so to NK cells and monocytes, and distant to erythrocytes and the spiked-in mouse cells.

metric or their coordinates in an appropriately transformed feature space. Whenever the user moves the mouse cursor over a cell, all cells are coloured according to their distance to this cell in feature space, thus indicating the cell's closest neighbours with the strongest colour (Figure 3.2A-C). By moving the mouse over all the cells in the plot, the user can quickly obtain an intuitive overview of how neighbourhoods may have been rendered differently in different plot regions. Buttons are provided to adjust the colour scale so that the user can choose which feature-space distance should be considered a "close neighbourhood" and hence given the strong (dark, green) colours.

One can pause here and try Sleepwalk himself or herself. At `https://anders-biostat.github.io/sleepwalk`, there is a short description of the package with some basic commands to run the app and live examples, including the one demonstrated in Figure 3.2. Any "sleepwalk" app runs in any Javascript-enabled web browser, i.e., it suffices to open the page in a browser without installing anything.

3.2.2 Exploring an embedding

Sleepwalk makes aspects visible that are not apparent from a dimension reduction alone. For example, the two large clusters under the cursor in Figure 3.2A and Figure 3.2B have quite different characteristics. In the T cell cluster (Figure 3.2A), most of the cells are very close to each other: the cluster shows up as a large green cloud no matter where one points the mouse. The monocyte cluster (Figure 3.2B), however, spreads over more considerable distances: only a part shows up in green, which "follows" the mouse. In a static t-SNE plot (such as Figure 3.1), this cannot be seen.

We can also check the cluster borders and discover, for instance (Figure 3.2C), that some cells in the monocyte cluster are more similar to those in the T cell cluster than to those in their own cluster. They may have been assigned the wrong cell type, or might be doublets. Thus, the Sleepwalk exploration can alert the analyst to the need for further investigation of possibly misleading features of a dimension-reduced embedding.

In Figure 3.2A-C, the colour scale was left at the automatically chosen range of only very small distances. When switching the colour scale to a wider distance range, we can also see here how relationships *between* clusters (Figure 3.2D) appear in the supplied distance values: we see which clusters are more and which are less similar to each other – information that a static t-SNE does not show, due to the method's design focus on faithful representation only of neighbourhoods. Care is needed here, however: once the considered distances exceed what one might recognise as "close neighbourhood", the choice of distance metric used will strongly influence interpretability of the visualisation, as was discussed in Section 3.2.3.

3.2.3 Feature-space distances

The colours in Sleepwalk are meant to indicate similarity or dissimilarity between the cells' expression profiles, quantified as distances. There are multiple suggestions for useful distance measures in the literature, and the users can provide whichever they prefer. To produce the t-SNE embedding in Figure 3.1, I followed the Seurat workflow [116], which calculates distances in a specific manner, and these are then also used by the t-SNE routine. In more details, the data preparation is described in Section 3.2.9. I have also used these exact distances to colour the points in the Sleepwalk rendition (Figure 3.2), thus allowing us to see directly where t-SNE succeeded and where it failed in its design goal of preserving the neighbourhood relation in its input data.

t-SNE uses a flexible approach to define the distance scale over which cells are considered neighbours: it adjusts the distance scale for each cell such that all cells have approximately the same number of neighbours (the so-called perplexity). Sleepwalk, in contrast, uses a fixed distance scale. This is on purpose: it allows us to note where the neighbourhood has a longer or shorter range (as shown in the comparison of Figures 3.2A and 3.2B). The app offers two buttons to increase or decrease the scale of the distance-to-colour mapping, allowing the user to manually choose what distance should be considered as the close ones.

3.2.4 Comparing embeddings

With the availability of choice in dimension reduction methods, the question arises of which one to use. Benchmark comparisons may address this question in general; see for example [127] for a comparison of UMAP with t-SNE and related methods. When working on a specific dataset, however, simply calculating multiple embeddings and comparing them side by side might be even more helpful. I demonstrate this here using murine cerebellum data [80]. (See section 2.2 for more details on the dataset.) In Figure 3.3, I show cells from development time point E13.5, first visualized with t-SNE (Figure 3.3A), then with UMAP (Figure 3.3B).

To compare the two embeddings, we need, at minimum, a way to see which points in the two plots correspond to the same cells. A classical approach is "brushing" [14]: selecting with the mouse a group of adjacently depicted cells in one plot causes them to be highlighted in the other one, too. Sleepwalk adapts this idea, but instead of the usual brush, it simply uses in all embeddings the same colour for points corresponding to the same cell. Moving the mouse over points in one plot then highlights the neighbourhood structure induced by the feature-space distance chosen for that embedding not only there but also in all displayed embeddings and so links them. It allows us to see for a structure in one embedding whether there are corresponding structures in the other embeddings.

In the example shown in Figure 3.3, there is a clear correspondence between the major structures generated by t-SNE and by UMAP. Even the arrangement of cells within these structures is the same, which one can follow in the life version of the app. The app can be found on the package's

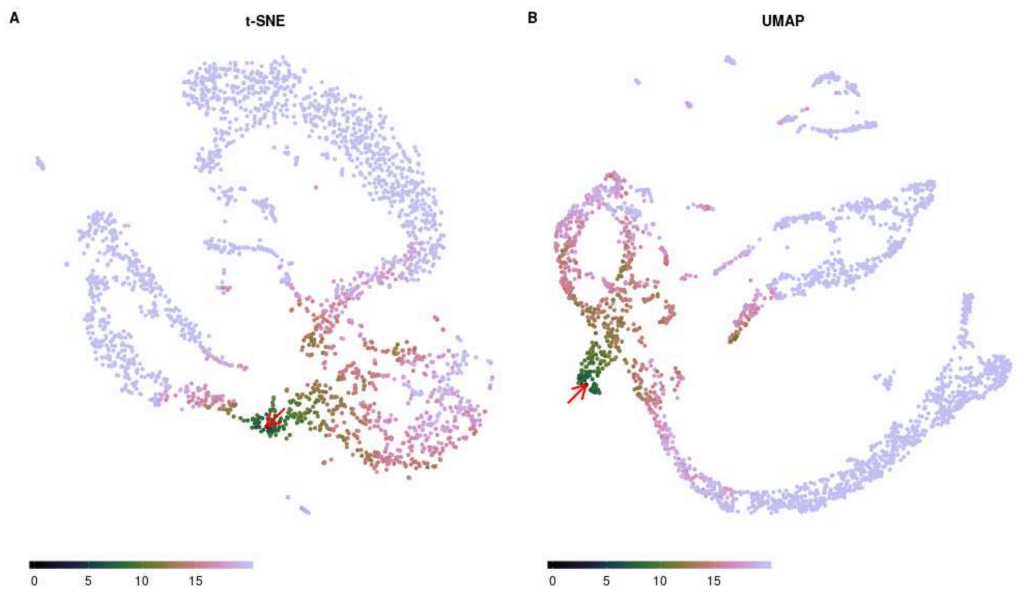


Figure 3.3: Sleepwalk being used to compare two embeddings of the same single-cell data of a developing murine cerebellum at embryonal time point E13.5 [80]: t-SNE on the left and UMAP on the right. The user can explore one embedding in the same ways as in Figure 3.2, while all other embeddings that are displayed concurrently are "slaved" to the one under the mouse cursor: each cell has the same colour in all embeddings. The red arrow shows the current mouse position.

web page at `https://anders-biostat.github.io/sleepwalk/`. There are, however, also differences: The cells at the mouse position in Figure 3.3 are part of the connecting "filament" in the t-SNE embedding but lie in an external "protrusion" in the UMAP. Further exploration in the live version of Figure 3.3 can suggest that UMAP forced the two branches to intersect while still trying to repel cells of different lineages away from each other (note the gap in the highlighted branch in Figure 3.3B). It is another example of dimensionality reduction artefacts that are hard to notice from a static image but can be uncovered with Sleepwalk.

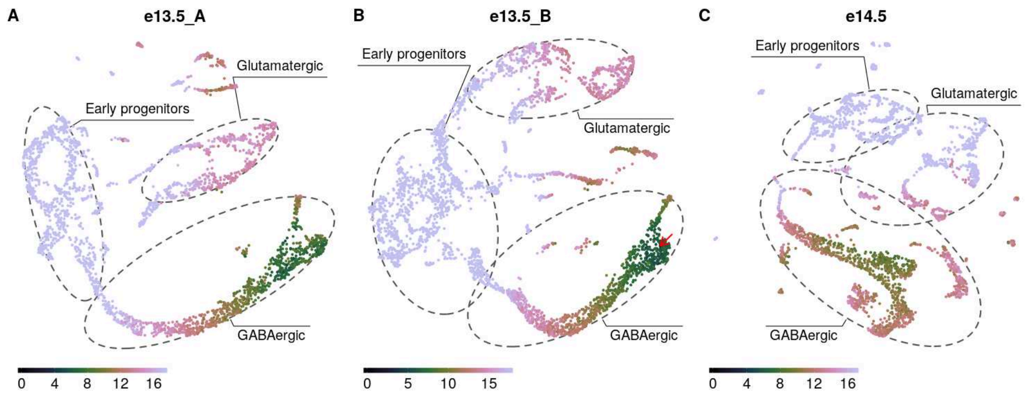


Figure 3.4: Sleepwalk in multi-sample mode, comparing three samples of a developing murine cerebellum: two samples of two different mice embryos at time point E13.5 (A, B) and the third (C) from E14.5. The red arrow shows the current mouse position. The dashed grey lines roughly indicate two different lineages and their common progenitor cells (more details on the marker genes used to draw these boundaries and their expression one can find in Section 3.2.9). By following the GABAergic branch, one can notice that its very tip in the E13.5 samples corresponds to cells in the middle of the branch in E14.5, indicating that the cells have differentiated further during the elapsed day.

3.2.5 Comparing samples

Until recently, most single-cell RNA-seq studies analysed only a single sample comprising many cells. However, the full value of the technique might become apparent only when it is used to compare between many samples. One currently popular approach to do so visually is to simply combine the data from the cells of all samples into one large expression matrix and perform t-SNE or UMAP on this. Often, global differences between samples, typically due to technical effects [128], will prevent similar cells from different samples to appear in the same cluster or structure in the dimension-reduced embedding. Methods to automatically remove such sample-to-sample differences (e.g., the CCA-based method in [115], and the MNN method in [129]) address this issue but will not always work and may risk also removing biological signal.

A visual alternative is to produce a dimension-reduced embedding separately for each sample and then try to find correspondences between the features in these. In Figure 3.4, it is shown how Sleepwalk allows to perform such an exploration comparing UMAP renderings for the two E13.5 and one of the E14.5 samples of the mouse cerebellum dataset. (See Section 2.2 for more details.) Exploring the data with the mouse shows the two E13.5 samples (Figures 3.4A and 3.4B) are almost identical. The two branches (GABAergic and glutamatergic neurons) can be clearly followed from the early progenitor cells to the most differentiated ones. Comparing the two E13.5 replicates reveals which aspects of the peculiar two-pronged shape of the glutamatergic branch are simply due to random variation and what seems reproducible. In the later E14.5 sample, the branches have disconnected from the progenitor cells, but Sleepwalk still allows us to identify corresponding cells. Sleepwalk can show that the GABAergic lineage is differentiated further in E14.5 than in E13.5 samples, as the endpoint of the branch in E14.5 corresponds to an intermediate point in E13.5. Sleepwalk allows one to discover such details immediately, with minimal effort. Of course, such a visual exploration cannot replace a tailored, detailed analysis, but it does provide a starting point and a first overview.

Crucially, using Sleepwalk's multi-sample comparison mode does not require any removal of global sample-to-sample differences with batch-effect correction methods. If the user selects a cell with the mouse in one sample, the cells that are similar to it will be highlighted, both in the same sample as well as in all other samples. It works even if the cells in the other samples seem more distant due to the additional sample-to-sample distance; we only might need to increase the scale of the distance-to-colour mapping for the cross-sample comparisons.

3.2.6 Comparing distance metrics

In the examples discussed so far, I have always coloured cells according to the default distance calculated by the Seurat workflow, namely the Euclidean distance in the space spanned by the first few principal components according to a PCA performed after certain preprocessing. The reason for this was not that this specific distance metric should be considered more correct or more "true" than any of the alternatives discussed in the literature, but simply because it is the distance metric that has been used as input to t-SNE and UMAP when calculating the discussed embeddings.

While this specific distance metric is popular due to its appearing in standard workflows such as Seurat's, this is, of course, no reason to consider it as more correct or "true" than possible alternatives or modifications. For instance, we may either choose to use all genes in the distance calculation or only some genes, which may either be chosen for having high expression or high signal-to-noise ratio or perhaps chosen, via manual curation, as especially informative with respect to cell type or state. We may use the genes as they are or aggregate them before into meta- or eigen-genes, e.g., by a principal component analysis (as done in the Seurat workflow). The way how the expression data has been transformed, normalised or preprocessed can be understood as part of the choice of distance metric. The last but not the least choice is, of course, the metric itself. There are numerous ways how to calculate distances from the selected and possibly preprocessed set of features. Besides Euclidean, one can calculate angular ("cosine") or correlation distance or use kernel functions [130] as it has been done, for instance, in [131]. Metrics can even be learned to suit the specific task researcher has in mind [132]. Dimensionality reduction techniques are typically based on the assumption that, in feature space, cells are located on the surface of a smooth manifold. The methods attempt to learn the manifold and then to replace the original distance with a geodesic one (i.e., distance within the manifold) [133, 134]. Diffusion distances [120] are a popular way of obtaining a manifold-following distance simply and efficiently.

Some of these metrics may yield similar results; others can drastically change cell-to-cell distances. In order to study the impact of the metric choice, Sleepwalk offers a variant to the mode for comparing embeddings described above, in which points that correspond to the same cell will get different colours in different panels of the app, each showing the same embedding but having a different distance matrix assigned to it. By hovering the mouse over a cell, the user can see how the cells' neighbourhoods differ between the distance metrics.

Figure 3.5 shows Sleepwalk in the distance comparison mode. Once again, I use the murine cerebellum dataset (specifically, stage E13.5, visualised with UMAP) as an example. As before, I used the 2131 genes chosen by the Seurat workflow as "variable", and then calculated distance matrices using four metrics: (i) Euclidean distance based directly on the normalised and logarithmised expressions of these genes, (ii) Euclidean distance in the space spanned by the first 50 principal components of a PCA performed using the variable genes, (iii) diffusion distance based on directly on the genes' expression or (iv) on the first 50 principal components. As expected, Euclidean distance calculated on all variable genes (i) is almost useless when applied to so many dimensions. Most of the distances are condensed around some median value, making it almost impossible to distinguish any patterns in the data. However, all other distances are already good enough to see the two developmental branches, with perhaps the diffusion distance separating them most clearly.

One should keep in mind that, in the example of Figure 3.5, we are comparing the four metrics not just to each other but also to the fifth one: The distance that was used to generate the embedding. UMAP is one of the manifold learning dimensionality reduction techniques. As such, it might be more similar to diffusion distances than to the Euclidean metric, even though the Euclidean distance in PCA space (distance (ii)) was used as input for the UMAP process. Taking this together, one might expect the combination of diffusion distance and PCA to correspond especially well with the embedding. However, which of the four metrics should be considered "best" is an entirely different question, as the suitability of a metric will depend on the task at hand. While a metric can be effective in separating specific cell types, it might at the same time fail to arrange cells by their cell cycle stage [135], and this can be considered a good or a bad thing, depending on whether differences due to cell cycle are considered a nuisance or a topic of interest in one's experiment. Therefore, any opinions or guidance on this question would be out of the topic of this thesis, which is the visualisation of omics data. What Sleepwalk does offer here is a means to explore differences between metrics and embeddings and understand them, not necessarily to perform benchmarks. Once a researcher is aware of such differences, it is up to him or her to decide if they affect data interpretation.

3.2.7 Beyond single-cell transcriptomics

In all the examples discussed so far, the points correspond to individual cells in samples assessed with single-cell transcriptomics. However, dimensionality reduction methods can be used for any kind of feature data described in Section 2.1. Clearly, Sleepwalk can also be helpful to explore these dimension-reduced embeddings as well. For example, prominent use case for dimension-reduction methods are large-scale studies comprising dozens or even hundreds of samples. In [136], for example, a collection of 131 bulk RNA-seq datasets comparing organ samples from several species is described and provide an

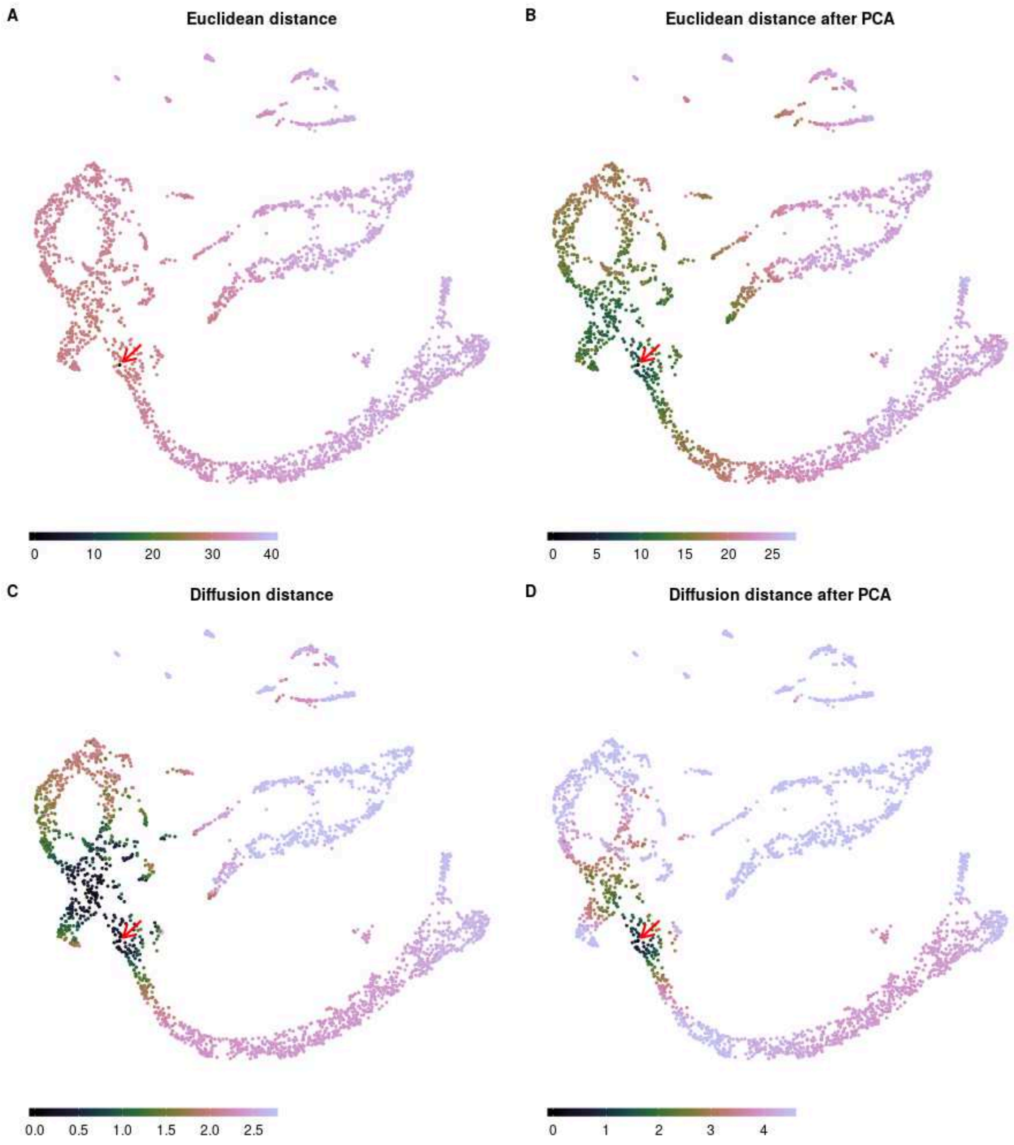


Figure 3.5: (Caption on the following page.)

Figure 3.5: Metric comparison using Sleepwalk. All four panels now show the same embedding: A UMAP visualisation of the E13.5 sample of murine cerebellum. The cells are now coloured based on four different metrics: Euclidean distance-based directly on the normalised and logarithmised genes' expressions (A); Euclidean distance in the space spanned by the first 50 principal components of a PCA performed using the variable genes (B); diffusion distance based on directly on the genes' expression (C) or on the first 50 principal components (D). The red arrow shows the current mouse position, which is at the intersection of GABAergic and glutamatergic lineages. Colour scales were adjusted so that they roughly stretch along the entire selected branch. The spread of colouring onto the other developmental branch shows how good is the metric in separating the two lineages.

overview PCA plot is provided as their Figure 1. In [137] a t-SNE plot is used to illustrate similarities and differences between their 246 blood cancer samples.

Research on dimension reduction originated in the machine learning field, with the original applications being the study of training data for machine learning applications. Of course, in this area, as well as in other applications of dimension reduction, Sleepwalk should also prove useful.

3.2.8 Implementation and usage

The R package "sleepwalk" is built on top of the "jrc" package, and thus relies on the principles described in Section 3.1. All essential "sleepwalk" features are implemented in JavaScript, while R is used mainly to format and send the input data to the web page. It means that a "sleepwalk" app can remain functional even without maintaining a WebSocket connection. (See Section 2.3.4 for more details.) As a result, it is possible to easily store an offline version of an app as an HTML file that can be opened in any modern browser without an R session running in the background or even installed. It can be helpful when an analyst wishes to share a Sleepwalk visualisation with colleagues or provide it on a web page or in a paper supplement. Moreover, such a possibility is also implemented in the package.

The central function of the package is also called "sleepwalk". The user provides it with the 2D coordinates for each object (cell) in the embedding, a square matrix of cell-to-cell distances, or, alternative to the latter, a data matrix from which Sleepwalk can calculate Euclidean or angular distances. For both these parameters, the user can also supply multiple matrices to display multiple embeddings concurrently for comparison. It can be done either such that each embedding represents the same objects (as in Figure 3.3), or that each embedding represents a different set of objects, but distances are also given between objects in different embeddings (as in Figure 3.4).

"sleepwalk" can easily be used in combination with other single-cell analysis frameworks. To visualise, for example, a Seurat data object after running `RunPCA` and `RunTSNE` one can use one of the following lines of code:

- for Seurat version 2.x:

  ```
  sleepwalk(seu@dr$tsne@cell.embeddings,
     seu@dr$pca@cell.embeddings )
  ```

- for Seurat version 3.x:

  ```
  sleepwalk( Embeddings(Reductions(seu, "tsne")),
     Embeddings(Reductions(seu, "pca")) )
  ```

This code takes the t-SNE embedding stored in the Seurat data object `seu` and displays it with "sleepwalk". If `RunUMAP` was used instead of `RunTSNE`, occurrences of `tsne` in the code above should be replaced with `umap`.

Though a WebSocket connection to a running R session is not required, it can provide some additional useful features that are unavailable in the "offline" mode.

One of them is "lasso": The user can encircle a group of points with the mouse, and the indices of these points are then reported back to the R session, where they can be queried with a callback function. It can be helpful if the analyst spots an interesting set of cells while exploring an embedding and wishes to perform further analysis on them.

There is also a `slw_snapshot` function that also requires a WebSocket connection. This function queries for a specific state of the app and generates a static plot, such as ones that are used as figures in this section. The plots are generated with the "ggplot2" package [98].

The colour scheme used to depict distances is the "cubehelix" palette, a colour map initially developed for astronomy and optimised for good visual separation between levels throughout its dynamic range [138].

For a description of further arguments of the `sleepwalk` function, please see the documentation. The package is open-source and is available on CRAN at `https://CRAN.R-project.org/package=sleepwalk`.

3.2.9 Data processing

To illustrate possible applications of the "sleepwalk" package two of the datasets described in Section 2.2 were used. Specifically, "CITE-seq data" [39] and "murine cerebellum data" [80].

CITE-seq data were used to produce Figures 3.1 and 3.2. To this end, the raw UMI counts were processed following the Seurat workflow proposed for exactly this dataset [116]. Data were normalised and log-transformed. 976 variable genes were detected with `y.cutoff = 0.5`. These genes were scaled and used for principal components analysis. For further analysis, the first 13 principal components were used, which explain around 23% of the total variance. The t-SNE (Figures 3.1, 3.2) and UMAP (Figure 3.2) embeddings were calculated using the default functions from the Seurat package. The assignments of cell types to clusters was taken, too, from the Seurat tutorial workflow [116]. The resulting Seurat object can be downloaded from Figshare (doi:10.6084/m9.figshare.7908059).

The raw reads of the murine cerebellum data were aligned and counted using the Cell Ranger [139] software (output files are accessible from Figshare, doi:10.6084/m9.figshare.7910483; this has been done by Kevin Leiss). Some genes and droplets were filtered out following the Methods section of [80]. All the cells with more than 10% of all UMIs coming from mitochondrial genes were removed. After that, all ribosomal and mitochondrial genes were excluded as well. Next, only cells that contain from 3500 to 15000 UMIs were kept. Lastly, I omitted all genes with zero expression in all the remaining cells. The filtered raw data were then used to create Seurat objects that can be found at Figshare, doi:10.6084/m9.figshare.7910483). Seurat was used to normalise and log-transform raw counts and find variable genes. The "irlba" R package [140, 141] was used to generate a PCA embedding of the data (each sample separately, only variable genes). The first 50 principal components were used for further analysis. The t-SNE embeddings were rendered with the "Rtsne" package [142], a wrapper around the code from [143]. The "uwot" package [144] was used for UMAP embeddings. Distances between cells from different samples (Figure 3.4) were calculated based on the variable genes shared between all the samples and a PCA embedding

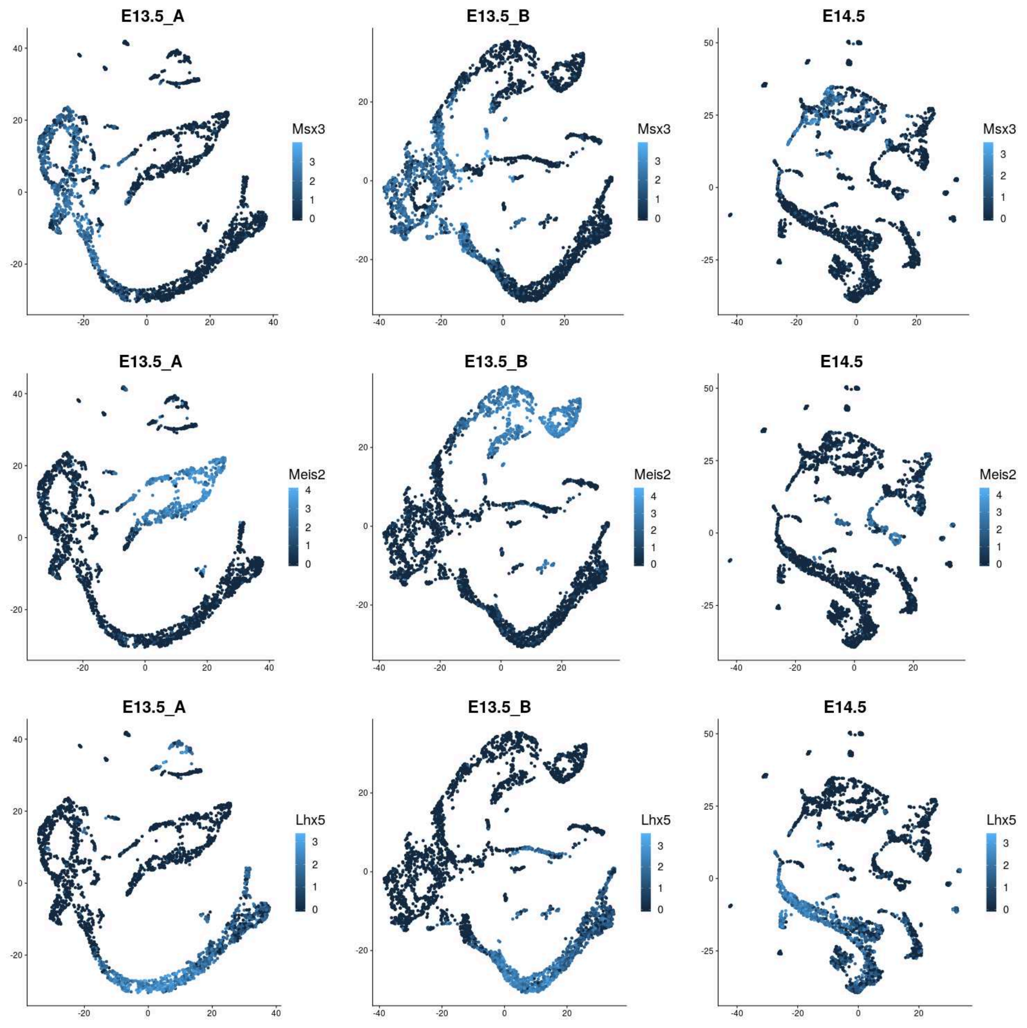


Figure 3.6: Gene markers used to identify lineages in Figure 3.4. In order to recognise patterns visible in UMAP visualisations of the murine cerebellum data, the expression of the three established gene markers was used: *Msx3* for early progenitors (top row), *Meis2* for the glutamatergic lineage (middle row), and *Lhx5* for the GABAergic lineage (bottom row). The colour scale shows the normalised and logarithmised expression values in each cell. Generally, a single marker is not enough for robust identification of the cell type in single-cell RNA-Seq studies. However, the discussion in Section 3.2.5 involves only approximate detection of main cerebellum cell lineages.

for all the cells. Euclidean distances in the space defined by the first 50 principal components are used to colour the points. To distinguish early progenitors from further differentiated cells of glutamatergic and GABAergic, the following marker genes were used: *Msx3* for early progenitors, *Meis2* for the glutamatergic lineage, and *Lhx5* for the GABAergic lineage (Figure 3.6). Contours in Figure 3.4 are drawn to include around 90% of cells that express each of the markers above a certain threshold using the `geom_mark_ellipse` function of the "ggforce" package [145].

Calculation of diffusion distance in Figure 3.5 is based on the "destiny" package [119]. Internal functions of the package were used to find nearest neighbours, to calculate local diffusion scale parameters `sigma` and to get initial transition probabilities. Then the diffusion was manually propagated with 16-time steps and calculated the resulting distances.

3.3 The "rlc" package (LinkedCharts)

In Section 1.2, I have outlined the benefits of interactive data visualisation compared to more traditional static plots and gave a short overview of this actively developing area. However, interactivity is still rarely used during the exploratory phase of a research project due to the amount of effort it usually takes to generate a customised interactive app. Some packages offer simple shortcuts to the benefits of interactivity. Nevertheless, the simplicity is often reached by hardcoding and presetting too many aspects of an app and thus making it applicable only to some specific data types.

With LinkedCharts, we tried to find a balance between the complexity of usage and possibilities for customisation to make it suitable for interactive data exploration. It requires only basic coding skills to produce fully functional apps for *ad hoc* analysis. With a little more effort, one can make a nicer looking app and customise the most commonly used plot settings (such as colours, labels, axes, etc.). Furthermore, with the time and effort generally required for the same task with other packages, one can use LinkedCharts to make a presentable app deployed on a server. Since the library is JavaScript-based, it can be combined with various existing web solutions. One can also write custom scripts that will change even hardcoded aspects of the library without making changes to the source code, making LinkedCharts extremely flexible.

LinkedCharts is not fixed on any specific task. It is a toolbox, and its blocks can be combined in any manner, the same way as one combines plots for a complex paper figure. All blocks share the same interface and very similar interactivity capabilities, which means that understanding one of them is enough to grasp the entire concept of LinkedCharts.

With all these, I hope that LinkedCharts can become a valuable asset for a scientific community that can be used both for everyday routine and for presenting one's research to a greater audience. LinkedCharts is available as an R package ("rlc") and as a JavaScript library. The R implementation of LinkedCharts, which is the focus of this thesis, is also referred to as R/LinkedCharts.

3.3.1 Linking charts

The central concept behind LinkedCharts is, as follows from the name, linking and focusing [147]. We connect two or more plots so that manipulations with one of them affect the others. It is easier to understand how this concept works in LinkedCharts with a simple example. Here, I use the oral cancer data briefly described in Section 2.2. It contains gene expression values from several tissue types (normal, cancerous, and dysplasia) from multiple patients, and the first natural question to ask based on these data is about differential expression between various tissue types. Several packages offer functionality to answer such questions [146, 148]. Here, the function `voom` from the "limma" package was applied to compare normal and cancerous tissues. It is common to visualise such a comparison with an MA plot [149] showing the average gene expression on the X-axis and log fold change between the two groups on the Y-axis (Figure 3.7A). Red dots correspond to genes that are considered significantly different between the two conditions (adjusted p-value < 0.1). However, how does the difference in expression look like for every single patient? Is it consistent across all the patients or only detected in some of them? Are there any artefacts or outliers that cause the p-value to be too small?

To find answers to these questions, one can add another plot that shows expression values (CPMs) for all the patients (Figure 3.7B). This plot can show expression for only one selected gene at a time, but LinkedCharts allows to link it to the MA plot. Now, any click on a point from the MA plot makes the plot to the right show expression of the corresponding gene. Figure 3.7 is based on a real LinkedCharts app that can be easily reproduced by anyone who has R and the "rlc" package installed. An interactive version of the app

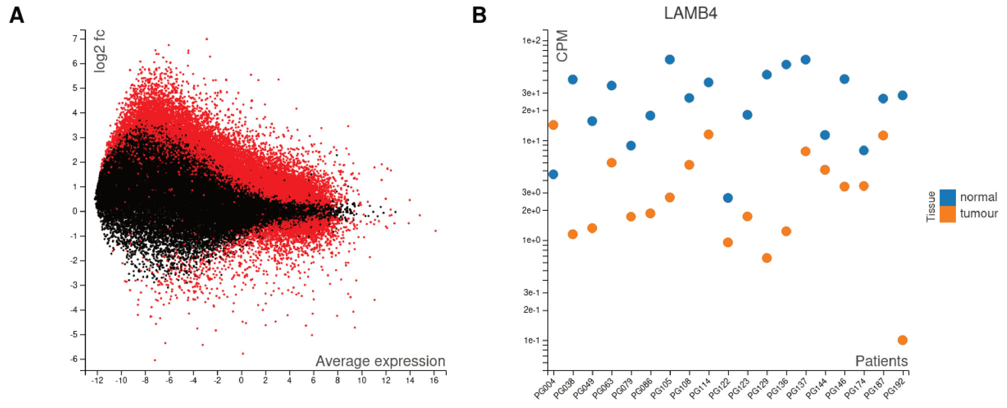


Figure 3.7: An overview of genes differentially expressed in cancerous and normal tissues from [38]. (See Section 2.2 for more details on the dataset.) The MA plot (A) shows all the sequenced genes with their average expression on the X-axis and log-fold change on the Y-axis. Red indicates genes where the difference was reported as significant by the “limma” [146] package. The plot to the left (B) shows expression values (CPMs) for a selected gene (in this case, LAMB4) and all the patients. This figure is based on a LinkedCharts app. When the user clicks on any point of the MA plot, the expression plot changes, showing the new selected gene. In this way, one can check immediately whether the genes labelled as significantly different are interesting for further study.

with a detailed explanation to take the reader through the entire process of its generation can be found in the tutorial at `https://anders-biostat.github.io/linked-charts/rlc/tutorials/oscc.html`.

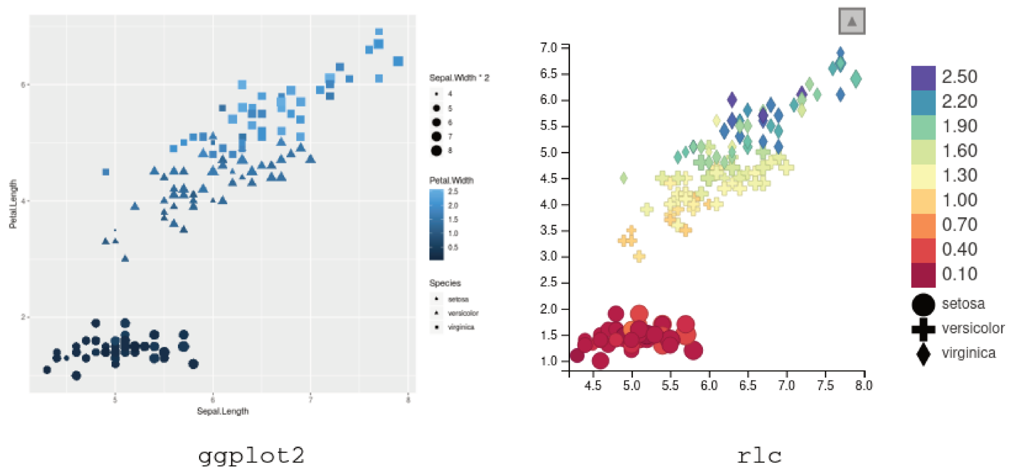
To explore the linking mechanism, we can look at the code that generates an app similar to the one from Figure 3.7. It is a minimal but complete code for the app. For now, I will concentrate only on the highlighted lines.

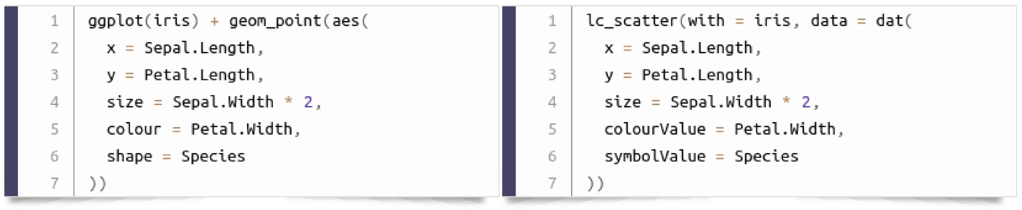
```
openPage(layout = "table1x2")
gene <- 1

lc_scatter(dat(
    x = AveExpr,
    y = tissuetumour,
7      colour = ifelse(adj.P.Val < 0.1, "red", "black"),
8      on_click = function(k) {
9         gene <<- k
10        updateCharts("A2")
11     }),
12  "A1", with = voomResult)
13
14  lc_scatter(dat(
15     x = patient,
16     y = normCounts[gene, ],
17     colourValue = tissue,
18     logScaleY = 10),
19  "A2", with = sampleTable)
```

It works as follows. In Line 2, an index of the gene to show in the expression plot is stored in the `gene` variable. This index is used to tell the chart which line of the `normCounts` matrix (where the normalised counts are stored) to use as y values of the expression plot (Line 16). Almost every chart of the R/LinkedCharts library has the `on_click` argument, which allows the user to define a function that will be called each time someone clicks on an element of the plot (point, line, cell of a heatmap, etc.). In this example, whenever this happens, the value of the `gene` variable is changed to the index of the clicked point (Line 9). Then R/LinkedCharts is told to update the second plot (Line 10, "A2" is its ID set in Line 19 of the example code). Updating means that the package will reevaluate all arguments inside the `dat()` function and change the chart accordingly. In this case, a new value of `gene` will yield new y values for the expression plot.

This simple logic is not limited to just two plots and provides a base to create various simple and complex apps. For example, the tutorial at https://anders-biostat.github.io/linked-charts/rlc/tutorials/citeseq1.html gives detailed instructions to generate an app for single-cell data exploration. The app consists of four charts, three of which are scatter plots, and one is an information table to show genes that define a selected cell cluster.

Besides a click, LinkedCharts can react to other events, such as moving the mouse cursor over or out of an element, selecting or deselecting elements with the *Shift* key pressed, clicking on any position of a plot or a heatmap label. The complete list can be found on the man page of any function of the "rlc" package. Understanding how to define these functions (above




```
ggplot(iris) + geom_point(aes(
  x = Sepal.Length,
  y = Petal.Length,
  size = Sepal.Width * 2,
  colour = Petal.Width,
  shape = Species
))
```

```
lc_scatter(with = iris, data = dat(
  x = Sepal.Length,
  y = Petal.Length,
  size = Sepal.Width * 2,
  colourValue = Petal.Width,
  symbolValue = Species
))
```

Figure 3.8: Typical syntax of an R/LinkedCharts plot with comparison to the “ggplot2” [98] package, one of the most commonly used plotting libraries. Lines of code are arranged to put the same aspects of the charts next to each other. “iris” dataset, which is one of the R built-in datasets, was used for this example. Both pieces of code are fully functional, and their output is shown above the code.

is shown a very typical example) is everything one needs to generate apps that run locally. More profound knowledge is required only to make an R/LinkedCharts app public. Section 3.3.3 explains it in more details.

3.3.2 Basic syntax

Overall, I tried to make R/LinkedCharts simple and familiar to any user with only fundamental knowledge of R. Any chart has a set of properties to define each of its specific aspects. One can already notice this from the previous example, where vectors of values were used as x and y coordinates of points or their colours. The same principle works in most plotting libraries. For example, Figure 3.8 shows a comparison of the syntax in R/LinkedCharts ("rlc" package) and ggplot ("ggplot2" [98] package) for a simple scatter plot. Lines are arranged to match the same aspects of the plots; above each code block, there is its output. One can see that the input data structure is identical, and there is hardly any difference between the two.

An important thing to notice here is the `dat()` function. One can set properties both inside and outside of it, but only those that are inside the `dat()` function will be evaluated on each `updateCharts` call. Everything outside this function will remain constant. There is a small example that can illustrate the effect of the `dat()` function.

```
lc_scatter(
   dat(x = rnorm(30)),
   y = rnorm(30))
```

Running this code will produce a scatter plot with 30 randomly located points. Now, every time one calls the `updateCharts` function, the x coordinates of each dot will change to new random values, but all the y coordinates will remain the same.

So far, I have mentioned only scatter plots, but R/LinkedCharts is not limited to them. There are 15 main functions in the "rlc" package. Each generates a specific type of plot (such as scatter plot, heatmap, bar plot, etc.) or a navigation element (such as sliders or text fields). Figure 3.9 shows them all together with some basic examples. Each plot, as it has been already mentioned, is defined by its properties: some of them are required (such as `x` and `y` for a scatter plot or `value` for a heatmap) many others are optional (`palette`, `title`, `ticks`, etc.). A full list of all the properties with live examples is available at `https://anders-biostat.github.io/linked-charts/rlc/tutorials/props.html` and also on the R man page of each plotting function. Many of the properties accept minor variations in spelling (`colour`/`color` or `labels`/`label`).

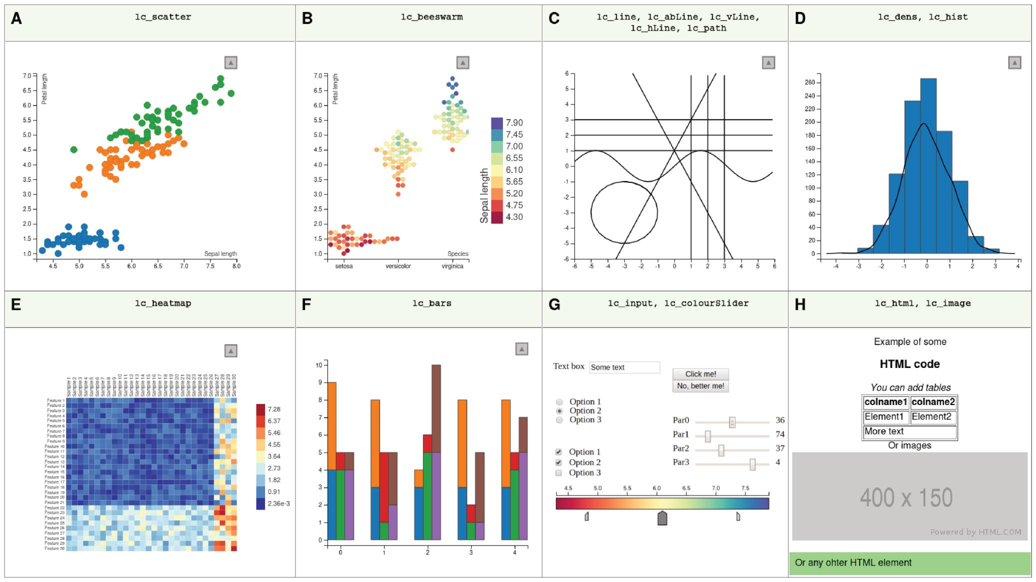


Figure 3.9: Gallery of all available plotting functions the in the "rlc" package. A scatter plot (A); a bee swarm plot (based on d3-beeswarm plugin [150]) (B); a collection of various lines (C); a histogram and a density plot (density was multiplied by a factor of 500 to be visible on the same plot as the histogram) (D); a heatmap (E); a bar chart (F); a collection of interactive elements to gather input from the user (G); functions to add custom HTML code and static plots to the page (H).

3.3.3 Use cases

Quality check

Summarising data is of great importance in any research project that concerns big data. It is troublesome to see patterns and make meaningful conclusions from raw readouts and measurements. Researches routinely count aligned mRNA reads, select only informative features, calculate various scores, etc. An appropriate generalisation is a key to uncovering biological patterns hidden in the data, but it also means loss of information.

Figure 3.7 nicely illustrates the process. An MA plot is generated to see the overall difference between the tissue types defined by thousands of genes. It immediately shows that there is a big difference between the normal and cancerous tissue samples, noticeably more genes with higher expression levels in the cancerous tissue and that the difference in genes downregulated in tumour samples seems to be more pronounced. For each gene, the data from 38 samples (dysplasia samples not included) are summarised to just three numbers: average expression, logarithmised fold change in expression between normal and cancerous tissues, and the significance estimate of this difference. Such a generalisation allows one to see the bigger picture but loses details on individual samples.

Similar approaches are a part of almost any pipeline used in a biological study. Each step takes the output from the previous one and modifies it, usually by summarising it to produce more interpretable data. However, some information is inevitably put aside. And from the resulting picture alone, it is hard, if not impossible, to see whether the raw data had any problems that could influence further conclusions. Various quality checks are devised to detect any possible issues with the omitted data. These are often quite helpful in pointing the researcher's attention towards some spurious artefacts in the data. Yet, especially in studies involving big data, these checks are fully automated. The researcher only looks at some summarised reports and may also have a look at some random examples. If these look reasonable, the general assumption is that all the data not filtered out are valid. However, while there is only one way for everything to be correct, there are numerous ways for the data to be either wrong or not as one has expected. Therefore, it is essential to have a possibility to look back before making any conclusions from a figure that shows a condensed result.

To illustrate this idea, we can have a look at a typical drug screening pipeline based on the drug screening data which have been described in Section 2.2. In this study [64], a cohort of drugs was tested against pancreatic cancer cell lines and the value of interest was a score to measure durgs' efficiency. A pipeline was established to transform raw read-outs into more interpretable scores and then use them to explore drug profiles. Its flow is shown in Figure 3.10. The experiment starts from a cell viability assay on a microplate. The measured value is an intensity of fluorescence from each well that can be visualised as a heatmap with each cell representing a microplate well (Figure 3.10A). Then the raw values are normalised to obtain a percentage of metabolically active cells in each well. These values are now used to score each tested drug with the Drug Sensitivity Score (DSS, [83]). The score is based on the area under the sigmoid curve fitted to all five tested concentrations of the drug. Figure 3.10B shows some of these curves. Now

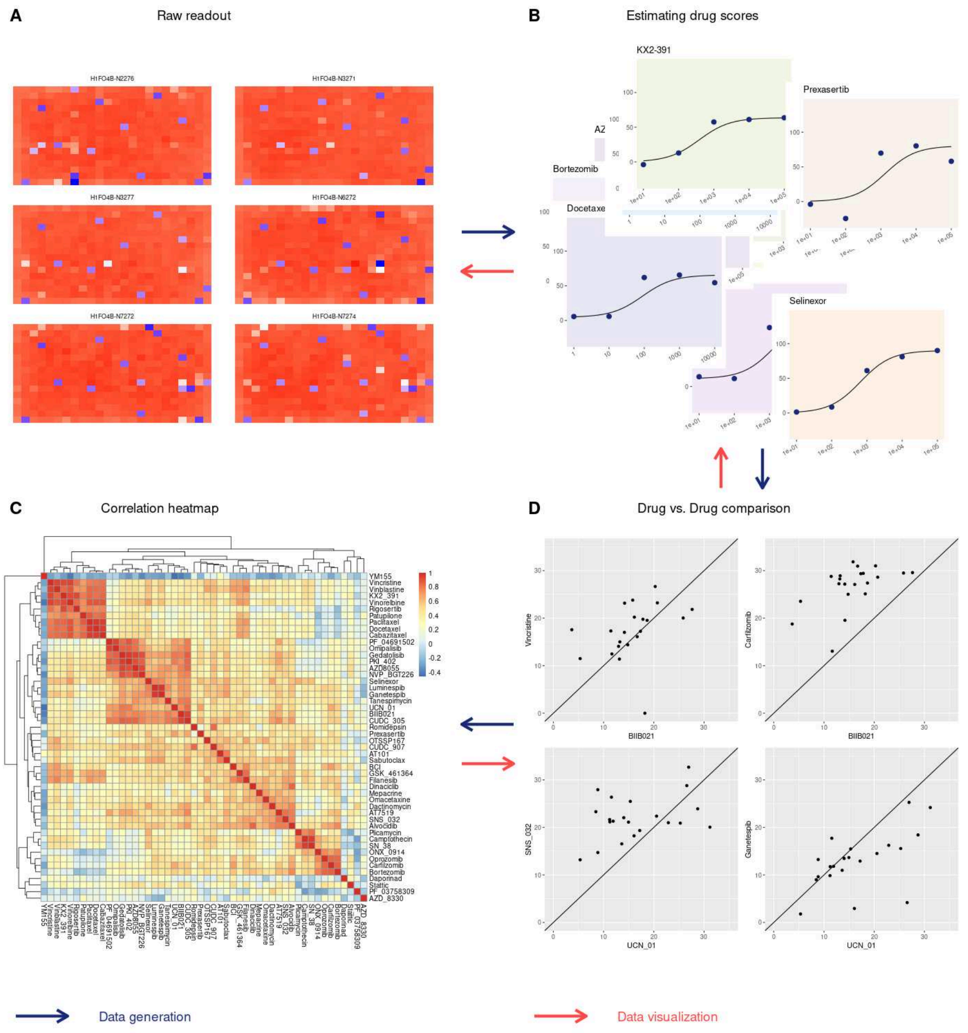


Figure 3.10: (Caption on the following page.)

Figure 3.10: The main idea behind the LinkedCharts library is shown based on a drug screening experiment. The blue arrow shows the direction of a typical pipeline used in drug screening experiments. (A) We start with reading intensity values from plates with different cell lines grown in the presence of studied drugs. These values are then normalised and turned into a fraction of the cells that remained metabolically active (RealTime-Glo^{TM} assay, [81]) or maintain membrane integrity (CellTox^{TM} assay, [82]). (B) A sigmoid curve is fitted to the obtained viability or toxicity values at different drug's concentrations, and the area under the fitted curve yields a single score for each drug. (D) Different drugs' scores are compared to each other across all the tested cell lines. (C) A drug-drug correlation heatmap is then produced to identify clusters of similar drugs. Red arrows illustrate the visualisation direction. It starts with the summary heatmap plot (C). Suppose the researcher is interested in a particular drug combination or a cluster of drugs. In that case, he or she can examine the corresponding drug scores simply by clicking on the heatmap cell (D). Then, she or he can examine the exact viability values for any given drug (B). And finally, if needed, it is possible to take one more step back and to look at raw readouts to inspect them for the presence of any artefacts (A). Though this figure illustrates a theoretical concept on which the LinkedCharts library is based, I have also made a working app based on the pipeline described above. It is available at `https://anders-biostat.github.io/linked-charts/thesis/drug_full.html` and is followed by the code to generate it in R or JavaScript and links to download all the necessary data.

there is a single value per drug-cell line combination. To find drugs with a similar efficiency profile, we can plot all the DSS values for each of the 21 tested cell lines against each other with two selected drugs as x and y axes (Figure 3.10D). Such a comparison produces a correlation value for each pair of drugs. Figure 3.10C shows them as a heatmap with clustered rows and columns, which allows one to study general patterns of drug groups with a similar effect on the cell lines. In Figure 3.10 this flow of data generation is shown with the blue arrows.

To visualise the data with LinkedCharts, we should go backwards (red arrows in Figure 3.10). We start with the most generalised plot, which, in this case, is a correlation heatmap 3.10C. This is what a researcher would usually rely on to draw conclusions. However, one may want to manually check an interesting or suspicious pattern in the data before making any further claims. To this end, LinkedCharts allows to easily add another explanatory plot, that for each heatmap cell display all the DSS values for the two corresponding drugs against each other (Figure 3.10D). A pair of drugs is selected by simply clicking on a heatmap cell. The drug score is also a generalised value that may be influenced by hidden artefacts in the data. If the researcher sees something out of place, he or she may want to check that as well. So we add another plot, showing the viability percentage for all five concentrations of the two selected drugs and the chosen cell line with the fitted sigmoid curve (3.10B). Just as the drugs are selected with mouse clicks on the heatmap, the cell line is picked by clicking on a point in the drug-drug plot. Finally, it is also possible to add the raw readouts as the fourth plot (3.10A). In this example, it is linked to the drug-drug plot since all five concentrations reside on the same plate. But under other conditions, it can be linked to a viability plot and display the plate with the specific test.

Such a chain of charts where each one represents a major step of the data pipeline and is explained in detail by the next one is what R/LinkedCharts is particularly good at. These apps would allow a quick and easy spot check of uncovered data patterns and give the researcher a better understanding of inner connections between the data. For instance, to what extent noise can influence the signal or what is the scale of changes in the value of interest is typical to the data.

Exploratory analysis

While interactivity is already increasingly used to present results of finished studies to the research community, with R/LinkedCharts, I would like to point out its usefulness for everyday exploratory analysis. To this end, R/LinkedCharts is designed in a way that requires one to spend not more time designing an app than it would take to perform a similar analysis with static plots. One may think of R/LinkedCharts as a container where the researcher can put his or her code to turn it into an interactive app. It will not perform a complex analysis automatically, but it will also not ask for much more than the usual routine coding, one should do with or without interactivity. Basic yet custom and functional apps do not require any special knowledge of the package's underlying structure or HTML layout. With this, I have attempted to encourage researchers to try out interactivity as a more improvised and need-based approach.

Let us return to the example from Figure 3.7. There, I show a typical part of a differential expression study: an MA plot (Figure 3.7A) and a plot with expression values for each sample to explore it (Figure 3.7B). I have already shown in Figure 3.8 (Section 3.3.2) that making a static plot in R/LinkedCharts is not much different in comparison to popular plotting libraries. It will take the same effort and same data structuring and preprocessing to make these plots with, for example, the "ggplot2" library, as with "rlc". Now, the plot in Figure 3.7B serves for spot-checking. It allows one to dive into the details of expression patterns, but only for one gene at a time. Of course, one may decide to only rely on the general overview that is provided by the MA plot (Figure 3.7A). However, a better practice is to make sure that there are no unexpected expression artefacts in the genes of interest. To do that, one would usually try to get a list of genes to check by filtering, using some special R tools, or somehow else, and then make a plot like the one in Figure 3.7A for each of them.

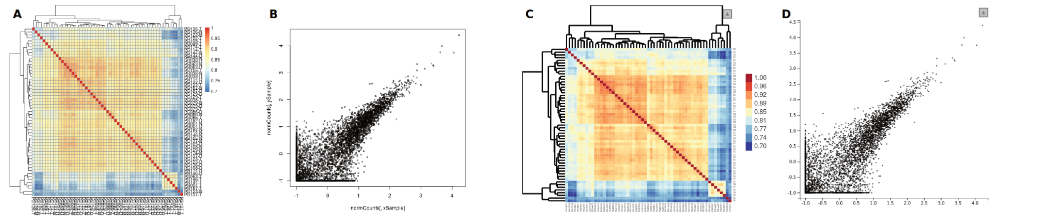
In the most simple case it would take copying and pasting the code for the spot check plot and replacing `y = normCounts["MyGene", ]` with `y = normCounts["AnotherMyGene", ]` (See the code chunk in Section 3.3.1). A little improvement to that would be to write `y = normCounts[myGene, ]` and keep updating the `myGene` variable: `myGene <- "MyGene"`, then `myGene <- "AnotherMyGene"`. After each update one should stil copy a piece of code that generates the plot into the console. Finally, the most effective way is to place this constantly copied and pasted code into a function, let say, `makePlot` and to call it, when necessary: `makePlot("MyGene")`, then `makePlot("AnotherMyGene")`.

Now, one pause here and return to Section 3.3.1, where the code to produce the app from Figure 3.7 is given. The `onClick` function does the same thing that is described in the previous paragraph. It stores the new gene into the variable `k`. It then performs an equivalent of copying and pasting to the console the code for the second plot, which in the "rlc" library can be done with just the `updateCharts` function for the sake of simplicity. The core idea here is so identical to the usual spot-checking routine that one can add our imaginary `makePlot` function to the "rlc" chart, and it will still work. It is only essential to be aware that in this case, the argument to the `makePlot` will be an index of the selected gene and not its name.

Thus, there are simply no additional requirements to the data or environment, no new concepts that one should adopt to perform the usual exploratory analysis or even to convert an existing script into an interactive app. Figure 3.11 shows such a transformation in details for another example of a common spot check practice.

This example is also based on the oral cancer data [38]. (See Section 2.2 for more details.) Above, we have looked at differentially expressed genes between normal and cancerous tissues. Still, before getting there, a researcher who has just obtained these data may want to get some overview of the samples. He or she may want to check for the presence of batch effects, clusters or outliers. It is also useful to check whether the samples group together by origin or by type. One of the ways to do so is to generate a 2D embedding of the high-dimensional data. This approach is in the focus of Section 3.2.1. Another option to get an overview is to have a look at a correlation heatmap (Figure 3.11A and 3.11C). In Figure 3.11AC, one can, for example, easily spot a small but tight cluster of samples and two outliers: the two samples to the right and bottom that are further from their nearest neighbours than most samples are from the majority of all others. The next most natural question to ask is how exactly these two samples are different from others. To answer it, one can plot gene expression values for several pairs of samples against each other (Figure 3.11B and 3.11D). It will give the researcher a feeling of what “similar” means for the particular dataset and how the outliers do not fit this pattern. Therefore, the visualisation task at hand reminds the one described above and illustrated by Figure 3.7. Figure 3.11 shows how to solve this problem with commonly used static plots (a heatmap from the “pheatmap” [99] package and a base R scatter plot, Figure 3.11AB) and with R/LinkedCharts (Figure 3.11CD). Below each set of charts, there is the code necessary to generate and update them. Neither the output nor the required commands are much different between the two. However, the R/LinkedCharts app is interactive. Besides linking, it provides other useful features such as zooming in and out, reclustering heatmap and showing sample names when the mouse hovers above them.

This similarity makes R/LinkedCharts a helpful tool for exploration. The required effort to produce an interactive app is the same as to make traditional static plots. The only significant difference in the coding style that the user needs to get used to is to update one of the plots with a custom function and not by copying and pasting the same piece of code. However, such an approach is usually considered a better practice. In addition, the on-the-fly draft apps that were used for exploratory analysis can later be combined into a more complex app to present final results without a need to start from scratch.



```
xSample <- 1
ySample <- 2
pheatmap(corMat)
plot(normCounts[, xSample], normCounts[, ySample],
   cex = 0.5)
```

```
xSample <- 1
ySample <- 2
lc_heatmap(value = corMat, clusterRows = TRUE, clusterCols = TRUE)
lc_scatter(x = normCounts[, xSample], y = normCounts[, ySample],
   size = 1.5)
```

To update the scatter plot:

(repeat for any pair of saples)

```
xSample <- 10
ySample <- 15

plot(normCounts[, xSample], normCounts[, ySample],
   cex = 0.5)
```

(add once to the heatmap arguments)

```
onClick = function(x, y) {
   xSample <<- x
   ySample <<- y
   updateCharts()
}
```

Figure 3.11: An example of an R/LinkedCharts app (C, D) that can be used during exploratory analysis and the code to generate it in comparison with the static plots (A, B) for the same purpose. The heatmaps (A, C) show Spearman correlation of gene expression for all samples from [38] (the oral cancer data from Section 2.2). Here, we can see two clear outliers in each heatmap's right and bottom and some more or less pronounced clusters of samples with similar gene expression levels. The scatter plots (B, D) show expression values for two samples plotted against each other. Browsing through several such plots can help the researcher get a feeling of the data and explore unexpected patterns like the outliers mentioned above. The code is split into two pieces, where the upper one is responsible for generating the plots, and below there are instructions of how to update the expression plot (B, D). For static plots, one has to execute the same lines of code for any pair of samples he or she wants to compare, while for R/LinkedCharts, the provided code should be added to the list of arguments for the heatmap. After that, switching between pairs of samples is done simply by clicking on the corresponding cell of the heatmap. The static heatmap (A) was generated with the "pheatmap" [99] package; scatter plot (B) was made with a base R function.

Public apps

All R/LinkedCharts apps start a server that listens to a given or randomly chosen port. To any request by a browser for the start page, this server answers with all the charts the user has added so far. More details on the principles of communication with the server in general and specifically in the “rlc” package can be found in sections 2.3.3 and 3.1.2. In practice, it means that multiple users can access any R/LinkedCharts app if they can send a request to the server. Any of the examples I have shown in this thesis can be made public by running them on a machine that can be accessed via the Internet. The code can run without any changes setting additional parameters, and already multiple users can access these apps, but their sessions will not be independent.

For the complete functionality, there is one more thing that has to be set. In a LinkedCharts app, one or several “state” variables are used. These are generally global variables that store currently selected genes, samples, etc. and are changed inside callback functions: `gene` from the example in Section 3.3.1, `xSample` and `ySample` from the example in Figure 3.11. If several users are working with these apps simultaneously, each click will change the state variables and consequently change the current state of charts for all the users. It can nevertheless be helpful if, for example, several people are using the app as a visual addition to an online meeting. Still, in most cases, one would like interactivity to be independent for each user.

To this end, it is only needed to enumerate state variables within the `openPage` function and give them some default values. For instance, the example from Figure 3.11 can be turned into a public app simply by adding the following line in the beginning:

```
openPage(sessionVars = list(xSample = 1, ySample = 2))
```

Now, multiple users can work with the app.

Of course, there are other settings to customise a public app. In addition to the charts, one can specify other default content for the page or scripts to be run for each new user. R/LinkedCharts provides tools to control each client session: for example, to close sessions inactive for specified time, limit memory usage or the number of simultaneously active connections. Still, all these parameters are optional. More information about possible options can be found on the R man pages for classes `App` and `Session` of the “jrc” package. (See Section 3.1 for more details.)

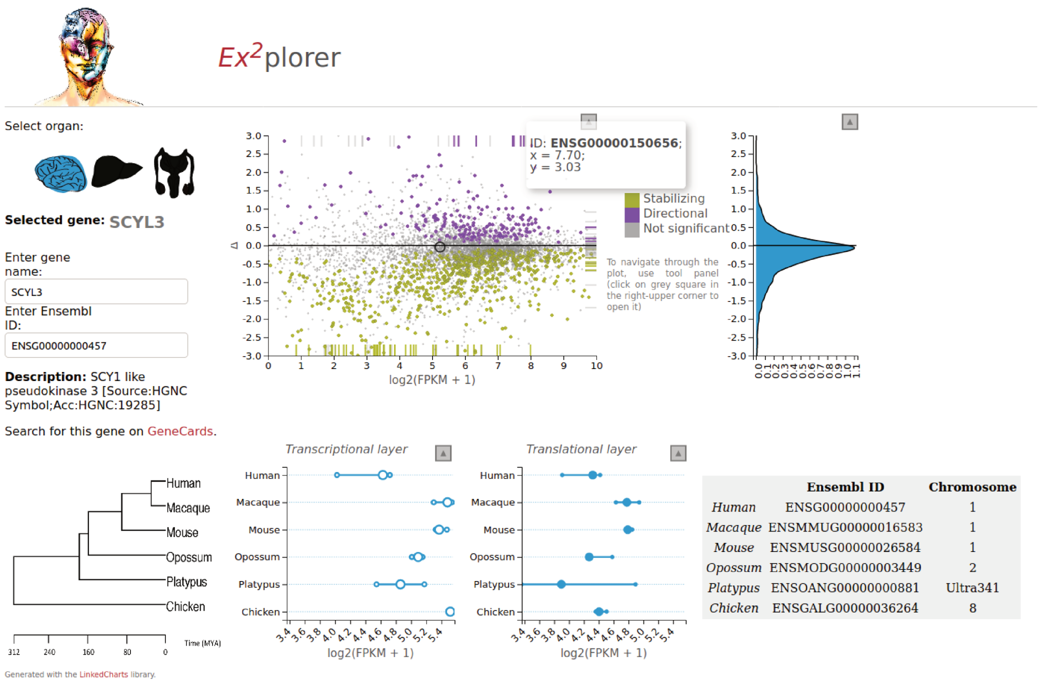
Stand-alone apps

Any app made with R/LinkedCharts ("rlc" package) requires a connection to a running R session. However, the R session is only responsible for sending the data to the app and updating them, while all the visualisation and interactivity handling happens on the JavaScript side. This fact makes it possible to use LinkedCharts for generating stand-alone apps in the form of an HTML page. Such a page can then be sent to a collaborator or used as a supplement file for a paper. Unlike a link to an app deployed somewhere on a server, this kind of interactive supplement will be available to any user with an installed web browser at any moment, without a need for the research team to maintain a running app on the server.

Such an app was made for [53] and is published alongside it. A screenshot of this app is shown in Figure 3.12. Data for this example is described in Section 2.2 as the translatome data.

The goal of the study was to check whether there is any connection between evolutionary changes in transcription and translation patterns. To this end, multiple samples from three tissues and six species were collected. RNA-Seq and Ribo-Seq assays were used to estimate transcription and translation efficiency, respectively (for more details about the assays, see Section 2.1.1 or the original paper [53]). If there is no connection between the evolution of transcription and translation, then the changes should accumulate for both of them at an equal pace. As a result, one can expect to see more variance on a translational layer than on the transcriptional one since translatome, in addition to its own variance, is also affected by the changes in the transcriptome. Therefore, to check if it is the case, we need a way to compare the amount of between species variance for any gene. As such, the so-called Δ-score was introduced as a standardised measure of the difference between the amount of variance in transcriptome and translatome. It now can be easily shown that the distribution of the score is noticeably skewed towards negative values for all three tested tissues, which means that there is even less variance in the translatome to the transcriptome. It suggests that the variation in translation efficiency, in fact, tend to compensate for the changes in the transcriptome.

From the visualisation point of view, this study is interesting because it does not rely on established analysis pipelines or any well-known approaches. Therefore it is of particular importance to make sure that the reader understands the methods behind the conclusions, and one of the most complicated



Selected gene: SCYL3

Enter gene name: SCYL3

Enter Ensembl ID: ENSG00000000457

Description: SCY1 like pseudokinase 3 [Source:HGNC Symbol;Acc:HGNC:19285]

Search for this gene on GeneCards.

To navigate through the plot, use tool panel (click on grey square in the right-upper corner to open it)

	Ensembl ID	Chromosome
Human	ENSG00000000457	1
Macaque	ENSMMUG00000016583	1
Mouse	ENSMUSG00000026584	1
Opossum	ENSMODG00000003449	2
Platypus	ENSOANG00000000881	Ultra341
Chicken	ENSGALG00000036264	8

Generated with the LinkedCharts library.

Figure 3.12: An example of a stand-alone app with LinkedCharts. The app was used as a supplement for [53]. Detailed information on the data, study goal and the source code for the app can be found in the related publication and also in Section 2.2. The app is written in JavaScript and, thus, can be downloaded and opened in any modern browser without installation requirements. The app is based on the same principles as other examples throughout this section. The main chart (upper row, centre) shows for every gene in the study its average expression and so-called Δ-score, which indicates whether evolutionary changes in translatome compensate for changes in transcriptome or introduce new variance. The two plots below show expression values for the selected gene in all the tested samples. The user selects a gene by clicking on the corresponding point of the main plot or by entering the gene name (left upper corner). The density plot to the right shows the distribution of Δ-scores, and its Y-axis is linked to the Y-axis of the main plot. In the upper right and bottom left corners, some additional information on the selected gene is displayed. Icons in the upper left corner allow switching between the studied tissues.

concepts here to grasp from the text description alone is, probably, Δ-score. However, it becomes clear if one spends some time going through examples: some genes with positive Δ-scores and some with negative scores. And an interactive app is a perfect tool for such an exploration. The app is available at `https://ex2plorer.kaessmannlab.org/`.

Such apps are possible since underneath "rlc" there is a fully functional and user-friendly JavaScript library *linked-charts.js*. "rlc" simply offers a wrapper around *linked-charts.js* and by means of the "jrc" package uses its JavaScript interface. Now, one can decide to do without the wrapper and use the *linked-charts.js* library directly, and this is how stand-alone apps are made. Of course, they require some familiarity with JavaScript syntax and basic concepts. The extensive knowledge may only be necessary to perform complicated calculations directly in JavaScipt or define complex reactions to interactive events. "rlc" and *linked-charts.js* are based on the same principles since one utilises the other. Because of this similarity, the conversion for simple apps is pretty straightforward. Every chart property turns from an argument of a plotting function into a method of the corresponding chart object. In most cases, neither property name nor input requirements changes, but callback functions to define property values are recommended for smoother interactivity. To get a feeling of these changes, one can look at the example gallery at `https://anders-biostat.github.io/linked-charts/`. There, every example is provided together with R code and its equivalent in JavaScript.

The most complicated part of LinkedCharts stand-alone apps is loading the data. Due to security restrictions, JavaScript in a browser cannot access local files without the user explicitly uploading them. This makes data input quite tricky, despite a variety of existing parsers for most commonly used formats. The easiest way for an R user to solve this problem is to convert data into JSON format and directly insert it into the JavaScript code. JavaScript can then interpret the data as an array or an object. There are also other ways of loading data into a JS/LinkedCharts app, described in our tutorial `https://anders-biostat.github.io/linked-charts/js/tutorials/data.html`.

Besides the "Data input" tutorial, on the same website, one can find several others that can walk an interested person through both basic and complex aspects of the *linked-charts.js* library.

- *"Properties"* describes the most important part of *linked-charts.js*: properties (they play the same role as arguments of the plotting functions in the "rlc" package). It will explain how to set and retrieve a property's value, how static and dynamic properties are different from each other, how to manipulate properties of several layers of the same chart, and even how to define a custom property.

- *"Types of charts"* goes through available chart types and provides examples of how to use each of them.

- *"Layers"* explains the concept of layers in *linked-charts.js* and how to manipulate them. In the end, it goes through the structure of a layer and shows how to define a custom type of chart.

- *"Data input"*, as mentioned above, shows examples of how to load data in a JavaScript app.

All tutorials are accompanied by interactive examples that allow one to change and rerun the code directly on the tutorial's page without downloading or installing anything (note run and update buttons in the right-upper corner of code chunks).

Unfortunately, I cannot provide users with an automated converter of R/LinkendCharts apps into their JavaScript counterparts. Unrestricted customisation is an essential part of the "rlc" package. We want to provide researchers with a way to wrap any scientific ideas into an interactive app rather than fit the problem into a preset pattern. To this end, we do not limit users by a predefined set of callbacks, strict rules for data structuring or any other manipulations with the app. From a running R session, the user gets truly complete control over the app. Therefore, for reliable transformation of a "rlc" app into an HTML file, one needs a converter of R language to JavaScript, which, to my knowledge, does not exist.

Overall, one may find it complicated to get used to an unfamiliar language; however, I would like to conclude this section by pointing out the benefits of LinkedCharts stand-alone apps:

- no R session needed;

  Not all hosting services allow to run R in the background, but a stand-alone app can be deployed almost everywhere. For instance, all the tutorials and examples mentioned in this thesis are hosted on GitHub

Pages, a free and easy-to-use way to share one's project. In addition, stand-alone apps can be incorporated into an HTML presentation (made with *reveal.js* [151] or a similar framework).

- no server needed;

  In fact, a stand-alone app does not even need a server to be deployed since it can be downloaded by a user. A server generally requires some maintenance, renewed subscription, etc. Something may change over time that makes the app no longer accessible, and the research group may be no longer interested in keeping it running. A stand-alone app can be deposited together with the paper, and it will be available as long as the paper is. For example, check the supplement files for our paper [111].

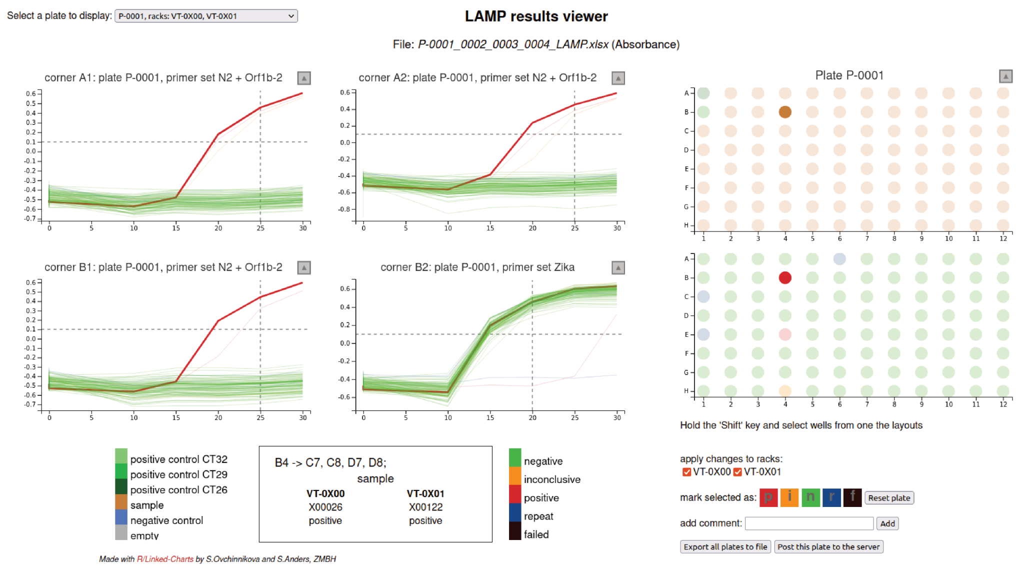
- an HTML file can be opened locally with any web browser.

  A stand-alone app is contained within a single HTML file (or several files for the sake of interpretability of the HTML code or to include figures). This file can be opened on any computer with a modern browser, and no other software installation is required. Thus, the app can be easily shared with collaborators from different areas of research who may be utterly unfamiliar with R.

GUI apps

Broad possibilities for customisation make it possible to use R/LinkedCharts for tasks beyond its primary goal (which is, as it follows from the name, linking several charts together for intuitive exploration). Since callback functions of the "rlc" package are not restricted to any predefined list of tasks, one can access the full spectrum of R functionality. Interaction with a chart cannot only update the app state but also store information in variables or external files, read new input from a file or ask the user for input, trigger some complicated calculations, send requests and data to a server. With all this, an R/LinkedCharts app can work as a graphical user interface for a custom R task.

To this end, the "rlc" package (`lc_input` function, see Figure 3.9G) offers a collection of elements to gather user input. The function provides a LinkedCharts interface to HTML "input" tag to add buttons, checkboxes, radio buttons, scrolls and text fields to the app. As any chart of the "rlc"



Select a plate to display: P-0001, racks: VT-0X00, VT-0X01
LAMP results viewer
File: P-0001_0002_0003_0004_LAMP.xlsx (Absorbance)
corner A1: plate P-0001, primer set N2 + Orf1b-2
corner A2: plate P-0001, primer set N2 + Orf1b-2
corner B1: plate P-0001, primer set N2 + Orf1b-2
corner B2: plate P-0001, primer set Zika
Plate P-0001
Hold the 'Shift' key and select wells from one the layouts
Made with R/Linked-Charts by S.Ovchinnikova and S.Anders, ZMBH

🟩	positive control CT32
🟩	positive control CT29
🟩	positive control CT26
🟫	sample
🟦	negative control
⬜	empty

B4 -> C7, C8, D7, D8;
sample

VT-0X00	**VT-0X01**
X00026	X00122
positive	positive

🟩	negative
🟧	inconclusive
🟥	positive
🟦	repeat
⬛	failed

apply changes to racks:
[✓ VT-0X00] [✓ VT-0X01]

mark selected as: p i n r f Reset plate

add comment: Add

Export all plates to file | Post this plate to the server

Figure 3.13: An example of an app that was used as a GUI to perform manual inspection and classification of LAMP testing for SARS-CoV-19 viral RNA [152]. The app was used in the lab of Prof. Dr. Michael Knop (ZMBH) during the SARS-CoV-2 surveillance study [153] and for voluntary testing for Covid-19 infection on campus (University of Heidelberg) offered since June 2020. To the right, the app shows a 96-well plate layout coloured either by content type (sample, empty, positive or negative control) or by the assigned result. To the left, it shows the results of three tests and one control for each sample. Accumulation of the product is indicated by the change of colour from red to yellow and is measured as a difference in absorbance on two wave lengths. This difference is plotted as a function of time. Besides exploration (highlighting the corresponding lines for each sample), the app allows to manually reassign status, store results locally as a *.csv* file or send them to the server, where they can be requested by people who provided the sample. The app is provided as an R script. It was used in-house and not published. However, the code and some example data are available on GitHub at https://github.com/anders-biostat/lamp_plate_analysis
.

package, the `lc_input` can get an R callback that is triggered every time the user changes the state of an input element (clicks a button or enters new text).

A screenshot of such a GUI app is shown in Figure 3.13. The app is a part of the project dedicated to applying loop-mediated isothermal amplification (LAMP, [154]) for detecting SARS-CoV-2 virus [152].

At the beginning of 2020, the advance of Covid-19 infection quickly led to the overloading of available testing capacities and, in turn, hindered prompt detection of the disease spread, especially by asymptomatic carriers. To address the issue, LAMP tests were offered at the University of Heidelberg as a cheaper and simpler alternative to commonly performed qPCR tests. Like PCR, LAMP allows detecting the presence of a specified DNA sequence in a saliva or swab sample, amplifying it with the help of pre-made primers. However, LAMP reaction does not require cyclic temperature changes and, thus, is much easier to perform. The disadvantage of this technique is that it is more stochastic than qPCR, and one may want to test each sample multiple times to increase accuracy.

Within a few months, a pipeline for RT-LAMP tests for the presence of SARS-CoV-2 viral RNA in saliva samples was successfully established. In this pipeline, after RNA extraction, each sample was split to fill four wells of a microplate. Three wells were used for testing and one for positive control (with other primers that will lead to product accumulation in any sample). LAMP tests can be run as colourimetric or as fluorescent assays. In both cases, the plates were heated up to 65°C and changes in absorbance or fluorescence were measured periodically for the next 50 minutes. If the sample is positive, in about 20 minutes, it should change colour from red to yellow (which can be measured as a difference of absorbance on 560 nm and 437 nm wavelengths) or become fluorescent. Due to the stochastic nature of the LAMP reaction, the resulting curve of absorbance or fluorescence changes over time should be observed manually for reliable conclusions. And that is where interactive visualisation is most helpful.

After establishing the pipeline, the lab offered its testing capacity for students and employees on the campus of the University of Heidelberg who wanted to get tested for Covid-19 infection and, later, for randomly chosen participants of the SARS-CoV-2 surveillance study [153]. For their convenience, a website was created where people could register their samples and query the result. The interactive app from Figure 3.13 was designed as a mediator between the output of the plate reader and the web platform.

It served multiple purposes. First of all, it was used by the involved lab members to inspect the results of colourimetric or fluorescence RT-LAMP scans. This is important due to the stochastic nature of the LAMP reaction. The results of all four performed tests are displayed in the app as four sets of curves showing changes in absorbance or fluorescence over time. To the right, there is a 96-well plate layout used for RNA extraction, coloured by either the well content (sample, control, empty) or by the assigned status (positive, negative, inconclusive, failed). If the user hovers the mouse over a particular well, all four corresponding curves are highlighted (as it is shown in Figure 3.13). And vice versa: hovering the mouse over any of the curves, highlights all other curves and the well for the corresponding sample. Thus, it was easy for the lab members to conclude the final status of the sample.

Another purpose of the app is manual classification. An automated classification is also performed as an initial step. However, in some cases, manual adjustments are required to account for some randomness in the reaction. In addition, a "failed" status, which means that the sample can not be tested at all, can only be set manually. Any sample can be selected to assign a new status by pressing a corresponding button in the right bottom corner. One can also add a comment to any sample. The results can be then stored as a *.csv* file.

Finally, the third purpose of the app was to push all the results to the server, where they could be queried by people who have provided the samples. A new dialogue window then appears for the user to log in to the server, and then a report is generated both as a log file and as an information table.

The app was used during the SARS-CoV-2 surveillance study [153] and for free voluntary testing for Covid-19 infection offered on the campus of the Heidelberg University during the pandemic of 2020/2021. This app is made for in-house usage and tailored for the pipeline of the specific lab and, thus, is not published. However, the source code is available on GitHub at `https://github.com/anders-biostat/lamp_plate_analysis`. It is provided in the form of an R script that reads in and processes the output of the microplate reader and then uses it to add interactive charts to the pre-made HTML page. The page contains an empty layout and functions to populate and ensure the functionality of non-LinkedCharts interactive elements. The R script is also responsible for storing any changes that are made in the app, saving them as an external file and pushing the results to the server. All this functionality is defined either as charts' callbacks or independent functions called by the "jrc" package.

Overall, the app is an example of not only a GUI made with the “rlc” package but also how one can utilise the power of JavaScript and R for interactivity in combination with R/LinkedCharts to ensure the behaviour that perfectly fits the needs of the given project.

3.3.4 Further customization

Since LinkedCharts is JavaScript-based, it can be combined with many existing web solutions without changing the source code of the package. One can customise the charts’ appearance with CSS, add additional scripts, specify an HTML layout. R/LinkedCharts can add interactive charts to an existing HTML page supplied as `startPage` argument of the `openPage`. Additional images, files or scripts can be loaded from a directory specified by the `rootDirectory` argument. In addition, more experienced users can take a step back and utilise the “jrc” package (which is described in Section 3.1) to employ the full power of JavaScript for reacting to user’s actions. The “rlc” package is based on “jrc” and inherits its main classes and, therefore, full “jrc” functionality is available for any R/LinkedCharts app by default. All this gives the user full control of what the app looks like and how it functions. Therefore a LinkedCharts app can be fitted to the specific needs of the particular project.

3.3.5 Implementation

All visualisation and interactivity handling in the LinkedCharts is implemented in JavaScript based on the *D3.js* library. (See Section 2.3.6 for more details.) The JavaScript basis of the “rlc” package is by itself a fully functional tool for interactive data visualisation that can be used to create standalone apps (Section 3.3.3). In *linked-charts.js* every chart is represented by an object and with a collection of properties to set any aspect of the chart. In addition to the set of properties, the charts have a collection of initialising and update functions, each responsible for maintaining a specific area: updating styles, adding or removing elements, positioning them, updating axes, etc. These functions are modularised, and therefore it is easy to customise behaviour even of a predefined chart. It is also possible to specify an entirely new type of a chart simply by providing a collection of update functions. The interactivity is ensured by providing the charts instructions where to look for property values in the form of callback functions rather than the values themselves. Therefore, any update causes *linked-charts.js* to request the most recent value of the property and display it accordingly. The complete information on the *linked-charts.js* principles and syntax, interactive examples, tutorials and download links can be found on our website `https://anders-biostat.github.io/linked-charts/js/`. The source code is available on GitHub at `https://github.com/anders-biostat/linked-charts`.

The "rlc" package is an R wrapper around *linked-charts.js* with additional functionality for maintaining multiple connections to the same app. It extends the `App` class of the "jrc" package (Section 3.1). Unlike `App` in "jrc", the `LCApp` also stores all current charts and automatically places them on the opened web page. It also handles client events and transforms R commands into the form that can be processed by *linked-charts.js*. It consists of an R and a JavaScript part, and the latter is loaded automatically into any served HTML page. The same way as the "jrc" package, "rlc" offers a collection of wrapper functions for every public method of the `LCApp` class and thus can be used by people without any knowledge of the inner structure of the package.

The "rlc" package is available on CRAN `https://CRAN.R-project.org/package=rlc` or GitHub `https://github.com/anders-biostat/rlc`.

Chapter 4

Discussion

4.1 Role of interactivity in visualisation

Interactivity has, on many occasions, proved itself to be useful for data visualisation. Static plots can accommodate only a limited amount of information, while data size and complexity are growing bigger with new technology advances. To make a static plot, one has to make a decision on what data to put aside, and it is not always evident that these data are, in fact, not significant. In big data exploration and presentation, a static plot is generally only a piece of a puzzle. It is crucial to put all such pieces together to see the entire picture, and interactivity is what can glue them to each other in an intuitive manner.

A static plot is also fixed. Any change, no matter how small, requires to redo the plot. This can be not very easy, for example, for a paper reader, since it takes to download the data and run the code if it is at all provided by the author. Even during the data exploration phase, when a researcher already has all the data and knows the code better than anyone else, it can still be annoying to keep changing colours, scales, sizes or opacity and rerunning the same lines over and over. Thus, one can be more inclined to believe a conclusion rather than thoroughly check it from all possible sides. Interactivity becomes handy in both cases.

It helps a reader to believe the results providing a fast and easy way to check that presented visualisations are not cherry-picked and do not contain hidden patterns. Interactivity is engaging. In the same way, as images attract more attention than plain text, interactive visualisation is often more catchy than static graphics. Some online news portals now employ this phenomenon by interactively presenting their graphs even if their amount of data can easily be put in a static plot. Interactivity helps others to browse the presented results at their own pace rather than following someone else's mind flow. The readers familiarise themselves with the data by trying out their ideas or just playing around, which helps to get on the same page with the author. Overall, interactivity is a great way to attract the reader's attention and quickly give him or her an overview of the data.

It is also a powerful tool for the researcher. A possibility to dive into the data with just a couple of mouse clicks and moves inspires exploration and, therefore, leads to a more thorough inspection of the data. There is no switching between coding and observing a result, no need to think about how exactly to specify the desired effect on the plot. Even an experienced user working with static plots constantly has to keep in mind two problems: how to plot something and how to examine the result. Interactivity allows one to start with the first task and then completely immerse in the latter without any distractions. It is only more important for the researchers who are not that confident with their coding skills since presented with a new problem (such as how to make a similar plot, but with points coloured somehow differently), they can easier lose track of some minor issues in the back of their mind.

Interactivity can even add an extra dimension to the visualisations, which is time. Not only is it possible to observe several various states and snapshots of the data, but also to trace the transition from one state to another, which can be in some cases more helpful than just comparing multiple static images.

The field of interactive visualisation is an actively developing one with some already well known and established tools such as "shiny" [29] or "plotly" [30]. However, possible benefits of interactivity are far from being exhausted, and this work contributes to some less explored areas. Here, I have presented three packages to facilitate the use of interactivity for visualising biological data.

4.2 "sleepwalk"

This section is copied from [111] originally written by myself.

Dimension-reduced embeddings such as those provided by t-SNE and UMAP have become a core tool in single-cell transcriptomics. They provide an overview of a study, help to check for expected and unexpected features in the data, allow researchers to form new hypotheses and to plan and organise the subsequent analysis. As they generally contain artefacts, a common concern is that these plots may be over-interpreted.

Dimension reduction is a research area with a rich history, long predating the use of these techniques for single-cell biology. The issue with distortions has been long discussed, with the possible distortions being classified [125] and quantified [126], and advice on careful interpretation derived from these [155]. To visually alert the viewer to distortions, some authors have suggested colouring each point by its so-called stress, i.e., the deviation of the point's on-screen distance to the other points from the distances in feature space [156]. Others proposed to colour the area around the points according to the amount of compression or stretching that the manifold underwent locally due to projection [157].

Such visualisations are valuable tools for developing and improving dimension reducing methods. Our approach, however, offers a novel aspect that is crucial: rather than merely alerting the user to distortions, Sleepwalk allows the user to directly see the underlying "truth" for the selected cell. It is possible due to our use of interactivity: by allowing the user to rapidly move the focus from cell to cell and the app instantly following in redrawing the colours, we are effectively escaping the confines of a two-dimensional representation (or, three-dimensional, if we also count static colouring as a dimension).

I have shown how this novel approach gives insights into dimension-reduced embeddings that would otherwise stay hidden and thus solves a core problem in the practical use of dimension reduction methods. I envision that Sleepwalk will be used in two manners: first, as a tool of exploratory data analysis, helping researchers to better understand their data, but also second as a reporting and communication tool, allowing researchers to present their results more transparently. For this latter application, Sleepwalk's ability to produce stand-alone HTML pages is crucial, as these pages can then be used, e.g., as supplements to publications, where they allow readers to check embeddings themselves, without the need to install any software.

I should be clear that a visual, interactive data exploration with Sleepwalk does not replace formal inference but complements or typically precedes it. Once one has formed a hypothesis about one's data using Sleepwalk, one should employ suitable formal analysis methods, such as statistical hypotheses tests, to confirm them. That analysis will then typically be done on the full, high-dimensional data. Dimension reduction methods are data reduction methods: this sacrifice of data is done to allow for visual inspection but is a hindrance for any numerical analysis.

The principle of Sleepwalk is beneficial not only for inspection of a single dataset but also lends itself for generalisation to comparative tasks. I have shown several possible modes of comparison: between different embeddings of the same data, between embeddings from several samples, and between different ways of preprocessing data and obtaining distances. The comparison between samples will find direct application in any study working with multiple samples; the other two are helpful in method selection and method development, as they allow for the comparison of data processing pipelines.

I, therefore, expect that Sleepwalk will find broad use not only in single-cell transcriptomics but essentially all instances of big data where experimental units (cells, samples, or the like) are described in a high-dimensional feature space.

4.3 Linked Charts

Unlike Sleepwalk, which solves an important but quite specific problem, LinkedCharts is a general purpose library. By now, interactive data presentations have become very common. However, researchers are still not so likely to employ interactivity for their everyday routine and prefer to generate numerous static plots instead of one app. Interactivity is generally something used only on special occasion: to share information with colleagues or to make an overview of an essential stage of the project. These apps are often not even done by the same people who performed most of the analysis and are instead delegated to collaborators with more coding experience. As a result, interactivity is still underused during routine data exploration.

As I have shown, R/LinkedCharts addresses this issue. It is designed to be used spontaneously for on-the-fly testing of any current ideas. To this end, it relies on two pillars: code simplicity and extensive possibilities for customisation to fit any given task.

With all this, I believe easy exploratory analysis to be the leading niche for R/Linked charts. As I have shown, it allows users to generate visualisations with the same effort as one generally puts into routine data digging and exploration. However, interactive apps are much more engaging than the static plots commonly used in the early stages of any project. When checking an idea or concern takes just a click, the researcher is more likely to go through the data thoroughly and, with this, hopefully, save time on the further steps of the analysis.

Once the need to present final or intermediate results to colleagues arises, the same essential apps that were previously used as “quick and dirty” solutions can be prettified and shared by deploying the app on a server. The required changes for an R/LinkedCharts app to work on a server are minimal, and therefore there is no need to start from scratch. One can utilise the same scripts as personal drafts for exploration and as a basis for result presentation.

The JavaScript basis of R/LinkedCharts offers an interface of its own that is also very simple and similar to the “rlc” syntax. Therefore, a user familiar with JavaScript or willing to learn its essentials gets a way to convert an R app into a stand-alone one fully contained within an HTML page. Such an app does not require any side resources, can be shared by email between collaborators and opened in any browser. It does not need to have a constantly running R session and can be published on any hosting, including the most simple ones that do not allow to run other software in the background.

The structure of the JS/LinkedCharts library and even principles of JavaScript as a language to manipulate DOM elements can allow an experienced user to customise not just the ecosystem in which LinkedCharts will be embedded but the charts themselves without a need to dive into the source code. One can go as far as defining custom types of charts (see `https://anders-biostat.github.io/linked-charts/js/tutorials/layers.html` for more details on that).

Overall, R/LinkedCharts serves two primary purposes: to facilitate data exploration and presentation. It offers an easy way to utilise interactivity for everyday research tasks. And it also provides the user with a possibility to fully employ the power of JavaScript for presenting the data. The latter aspect addresses more experienced users and thus has no limits for possible customisation of a LinkedCharts app.

4.3.1 Code simplicity

In order to encourage people to employ visualisation routinely, it is not enough to make it just simple. It should also be familiar to potential users. This motivation is what has driven the design of R/LinkedCharts syntax. As a result, R/LinkedCharts consists of two major parts, both of which are set as arguments to various plotting functions.

First of all, we have a set of static properties that define all required and optional aspects of a chart, such as styling, coordinates, axes settings, labels and titles, etc. They are not different from commonly used plotting libraries and, thus, can be easily grasped by any user who produces any kind of plots in R. The list of all available properties can be found on the man page for any plotting function of the "rlc" package or, with examples, on our website at `https://anders-biostat.github.io/linked-charts/rlc/tutorials/props.html`.

Linking is done by means of callback functions that correspond to specific user actions (mouse click, double click or hovering, selecting elements with pressed *Shift* key). These functions are called any time the corresponding event occurs, and they describe the app's reaction to this event. Though it may sound complicated for someone who is not very confident in his or her coding skills, the content of such a function is simply the code that is repeatedly copied and pasted while working with static plots. For example, if one keeps making the same plot for different genes to browse through the data, he or she can wrap the required code in a function and use it as `on_click` argument of some summary plot in a R/LinkedCharts app.

Therefore, as I have shown in Section 3.3.3, making an interactive app with the "rlc" package is more similar to restructuring the existing code rather than writing something new. In addition, there is an `updateCharts` function that can be used instead of replacing an existing plot with a new one. The only concept in R/LinkedCharts apps that may be new for less advanced users is utilising custom functions. However, it is in any case considered good practice to arrange pieces of code that perform a specific task as a function since it makes code more robust and interpretable. Therefore, adopting such a habit can be beneficial for a user regardless of whether he or she then decides to use R/LinkedCharts.

4.3.2 Customisation

A tool for exploration should also possess a high degree of flexibility. We do not want a researcher to formulate questions to fit the app. Instead, we want an app to be capable of facilitating the search for answers for a wide range of possible problems. To this end, R/LinkedCharts works with custom callback functions instead of providing a predefined list of possible reactions to user's interactions with the app.

In Section 3.3 I have provided code for several interactive apps and their linking functions. Those were the simplest examples of `on_click` callbacks which include only changing one of the state variables (the ones that store currently selected gene, sample, etc.) and a call to the `updateCharts` function. It is, in fact, a ubiquitous and yet powerful form of a callback for "rlc". However, it does not have to be that simple. A callback can include any additional calculations, results storing, or any other kind of analysis one may need. One of our tutorials, for instance, `https://anders-biostat.github.io/linked-charts/rlc/tutorials/citeseq1.html` uses a callback that looks for the genes with higher expression levels in the selected cluster than in the other cells and prints the top ten of them as a table. Therefore, the R/LinkedCharts app can be customised to the point when it does not even use chart linking and instead works as a graphical user interface for a given task.

The linking mechanism itself is also not fixed to any particular scheme. Every chart can be linked to any other or multiple ones. Any kind of backwards or partial linking is also possible. As a result, there are no requirements for the data structure. For any given chart, there are, of course, predefined input formats. However, the overall data arrangement is up to the user. All the data may be stored as a single list, or there can be several variables. Each chart of an app may use its own data variable, or it can take parts of multiple variables. One may have to convert the data into some format for complicated apps, but even then, it would serve only for code simplification. Generally, no data conversion or reordering is required.

4.4 JavaScript for enhanced interactivity

Both visualisation solutions, "sleepwalk" and "rlc", are R packages but rely heavily on JavaScript. It emphasises a more general idea about interactive visualisation that I would like to promote with this work.

JavaScript, as was mentioned in Section 2.3.2, was developed to add interactive elements to previously static web pages. And till now, interactivity remains its main goal. Therefore it seems only natural to look at JavaScript when talking about interactive visualisations. Moreover, the infrastructure that is built around the R programing language makes it easy to combine the two languages in a user-friendly manner. The idea itself is not new, and JavaScript has already been used by various R packages such as "plotly" [30], “htmlwidgets” [158] and many others. However, I think that the connection between R and JavaScript should become more straightforward to let users apply their JavaScript knowledge directly and provide a way to utilise any of the available JS libraries.

As a way of such a direct interaction, I have proposed the “jrc” package. Its core idea can be summarised just in one sentence: It allows users to run any JavaScript command from R and any piece of R code from JavaScript. Section 3.1 goes into more detail through its full functionality. However, all the features are there only to facilitate this main purpose: exchanging messages between an R session and a web page. “jrc” is not a package for visualisation but rather a tool to make such packages. And any package, built on top of “jrc”, inherits this capability of exchanging messages, and hence access to any custom JavaScript features.

Both “sleepwalk” and “rlc” a based on “jrc”, but in two distinct manners.

“sleepwalk” is written mainly in JavaScript. R only performs input checks and then passes the data to the web page, where they are processed and displayed. The same principle is used in the “htmlwidgets” package, and it is useful when one wants to later save the generated app as a fully functional *.html* file. Nevertheless, “sleepwalk” retains the ability to maintain an open connection, which is used, for instance, by its lasso selection function (when indices of the selected points are sent back to the R session). It also can be used to influence the content of the web page from the R session, if needed.

“rlc”, on the other hand, relies on the web socket connection all the time. JavaScript performs no calculations, and the data are passed only by request; most of the reactions to mouse events trigger a call to an R function. Such an active connection makes it easy to control the app from an R session and, thus, ensures possibilities for customisation even for those unfamiliar with JavaScript. Moreover, users who know web applications or are willing to learn some basics can also make changes directly in JavaScript, either by loading custom scripts or using the inherited methods of the “jrc” package. The app described in Section 3.3.3 and in Figure 3.13 is an example where both approaches are used. The *.html* file utilised by the app contains a custom layout and additional JavaScript functionality. Also, some of the default LinkedCharts elements are modified from the R session after the charts are added.

The difference between the presented visualisation packages is defined by the way how they utilise the underlying “jrc” package. “sleepwalk” uses it to run a specified JavaScript interface, while “rlc” extends it by adding possibilities to store and display charts. Both approaches explore possible benefits of the direct communication between R and JavaScript provided by the “jrc” package. I believe that not only package developers can benefit from it, but also end-users, who now get an opportunity to customise their apps to the extent that the package developer may have not even imagined, since direct access to JavaScript functionality allows one to do anything with the visualisation app.

Bibliography

1. Marx, V. The big challenges of big data. *Nature* **498,** 255–260 (2013).
2. Demchenko, Y., Zhao, Z., Grosso, P., Wibisono, A. & De Laat, C. *Addressing big data challenges for scientific data infrastructure* in *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings* (2012), 614–617.
3. Rieber, L. P. A historical review of visualization in human cognition. *Educational technology research and development* **43,** 45–56 (1995).
4. Friendly, M. & Denis, D. J. Milestones in the history of thematic cartography, statistical graphics, and data visualization. *URL http://www. datavis. ca/milestones* **32,** 13 (2001).
5. Bertin, J. *Semiology of Graphics: Diagrams, Networks, Maps* ISBN: 9781589482616. https://books.google.de/books?id=X5caQwAACAAJ (ESRI Press, 2011).
6. Wilkinson, L. *The grammar of graphics* (Springer Science & Business Media, 2013).
7. O'Donoghue, S. I. *et al.* Visualization of biomedical data. *Annual Review of Biomedical Data Science* **1,** 275–304 (2018).
8. Deng, L. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine* **29,** 141–142 (2012).
9. Heer, J., Kong, N. & Agrawala, M. *Sizing the horizon: the effects of chart size and layering on the graphical perception of time series visualizations* in *Proceedings of the SIGCHI conference on human factors in computing systems* (2009), 1303–1312.

10. Fisher, D. *Big data exploration requires collaboration between visualization and data infrastructures* in *Proceedings of the Workshop on Human-In-the-Loop Data Analytics* (2016), 1–5.

11. Keahey, T. A. Using visualization to understand big data. *IBM Business Analytics Advanced Visualisation* (2013).

12. Bresciani, S. & Eppler, M. J. The risks of visualization. *Identität und Vielfalt der Kommunikations-wissenschaft (2009),* 165–178 (2009).

13. Newman, W. M. *Principles of interactive computer graphics* tech. rep. (1979).

14. Becker, R. A. & Cleveland, W. S. Brushing scatterplots. *Technometrics* **29,** 127–142 (1987).

15. Shander, B. *5 Reasons to Visualize Your Data and Make it Interactive* 2016. https://medium.com/@billshander/5-reasons-to-visualize-your-data-and-make-it-interactive-65442d8612f6.

16. Yuk, M. & Diamond, S. *Data visualization for dummies* (John Wiley & Sons, 2014).

17. Caldarola, E. G. & Rinaldi, A. M. *Big Data Visualization Tools: A Survey* in *Proceedings of the 6th International Conference on Data Science, Technology and Applications* (2017), 296–305.

18. Noronha, A. *et al.* ReconMap: an interactive visualization of human metabolism. *Bioinformatics* **33,** 605–607 (2017).

19. Wick, R. R., Schultz, M. B., Zobel, J. & Holt, K. E. Bandage: interactive visualization of de novo genome assemblies. *Bioinformatics* **31,** 3350–3352 (2015).

20. Hillje, R., Pelicci, P. G. & Luzi, L. Cerebro: interactive visualization of scRNA-seq data. *Bioinformatics* **36,** 2311–2313 (2020).

21. Broman, K. W. R/qtlcharts: interactive graphics for quantitative trait locus mapping. *Genetics* **199,** 359–361 (2015).

22. Zhao, J., Chevalier, F., Collins, C. & Balakrishnan, R. Facilitating discourse analysis with interactive visualization. *IEEE Transactions on Visualization and Computer Graphics* **18,** 2639–2648 (2012).

23. Wu, Y. *et al.* OpinionSeer: interactive visualization of hotel customer feedback. *IEEE transactions on visualization and computer graphics* **16,** 1109–1118 (2010).

24. Bostock, M., Ogievetsky, V. & Heer, J. D^3 data-driven documents. *IEEE transactions on visualization and computer graphics* **17,** 2301–2309 (2011).

25. Satyanarayan, A., Russell, R., Hoffswell, J. & Heer, J. Reactive vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE transactions on visualization and computer graphics* **22,** 659–668 (2015).

26. P'ng, C. *et al.* BPG: Seamless, automated and interactive visualization of scientific data. *BMC bioinformatics* **20,** 1–5 (2019).

27. Sievert, C. *et al.* Extending ggplot2 for Linked and Animated Web Graphics. *Journal of Computational and Graphical Statistics* **28,** 299–308 (2019).

28. Satyanarayan, A., Moritz, D., Wongsuphasawat, K. & Heer, J. Vega-lite: A grammar of interactive graphics. *IEEE transactions on visualization and computer graphics* **23,** 341–350 (2016).

29. RStudio, Inc. *Easy web applications in R.* URL: http://www.rstudio.com/shiny/ (2013).

30. Sievert, C. *Interactive Web-Based Data Visualization with R, plotly, and shiny* ISBN: 9781138331457. https://plotly-r.com (Chapman and Hall/CRC, 2020).

31. Travaglini, K. J. *et al.* A molecular cell atlas of the human lung from single-cell RNA sequencing. *Nature* **587,** 619–625 (2020).

32. Roider, T. *et al.* Dissecting intratumour heterogeneity of nodal B-cell lymphomas at the transcriptional, genetic and drug-response levels. *Nature Cell Biology* **22,** 896–906 (2020).

33. Kalucka, J. *et al.* Single-cell transcriptome atlas of murine endothelial cells. *Cell* **180,** 764–779 (2020).

34. Batch, A. & Elmqvist, N. The interactive visualization gap in initial exploratory data analysis. *IEEE transactions on visualization and computer graphics* **24,** 278–287 (2017).

35. Bishop, C. M. *Pattern recognition and machine learning* (Springer, 2006).

36. *The American Heritage Medical Dictionary* 2007.

37. Omenn, G. S., Nass, S. J., Micheel, C. M., *et al.* Evolution of translational omics: lessons learned and the path forward (2012).

38. Conway, C. *et al.* Elucidating drivers of oral epithelial dysplasia formation and malignant transformation to cancer using RNAseq. *Oncotarget* **6,** 40186–40201. ISSN: 1949-2553. https://www.oncotarget.com/article/5529/ (2015).

39. Stoeckius, M. *et al.* Simultaneous epitope and transcriptome measurement in single cells. *Nature Methods* **14,** 865 (2017).

40. Schaum, N. *et al.* Single-cell transcriptomics of 20 mouse organs creates a Tabula Muris: The Tabula Muris Consortium. *Nature* **562,** 367 (2018).

41. Wang, Z., Gerstein, M. & Snyder, M. RNA-Seq: a revolutionary tool for transcriptomics. *Nature reviews genetics* **10,** 57–63 (2009).

42. Lockhart, D. J. *et al.* Expression monitoring by hybridization to high-density oligonucleotide arrays. *Nature biotechnology* **14,** 1675–1680 (1996).

43. Callow, M. J., Dudoit, S., Gong, E. L., Speed, T. P. & Rubin, E. M. Microarray expression profiling identifies genes with altered expression in HDL-deficient mice. *Genome research* **10,** 2022–2029 (2000).

44. Abe, T. *et al.* Time-course microarray transcriptome data of in vitro cultured testes and age-matched in vivo testes. *Data in Brief* **33,** 106482 (2020).

45. Jassal, B. *et al.* The reactome pathway knowledgebase. *Nucleic acids research* **48,** D498–D503 (2020).

46. Grebe, S. K. & Singh, R. J. LC-MS/MS in the clinical laboratory–where to from here? *The Clinical biochemist reviews* **32,** 5 (2011).

47. Bassani-Sternberg, M. *et al.* Direct identification of clinically relevant neoepitopes presented on native human melanoma tissue by mass spectrometry. *Nature communications* **7,** 1–16 (2016).

48. Bojkova, D. *et al.* Proteomics of SARS-CoV-2-infected host cells reveals therapy targets. *Nature* **583,** 469–472 (2020).

49. Mertins, P. *et al.* Proteogenomics connects somatic mutations to signalling in breast cancer. *Nature* **534,** 55–62 (2016).

50. Zhao, J., Qin, B., Nikolay, R., Spahn, C. M. & Zhang, G. Translatomics: the global view of translation. *International journal of molecular sciences* **20,** 212 (2019).

51. Ingolia, N. T., Ghaemmaghami, S., Newman, J. R. & Weissman, J. S. Genome-wide analysis in vivo of translation with nucleotide resolution using ribosome profiling. *science* **324,** 218–223 (2009).

52. Wang, T. *et al.* Translating mRNAs strongly correlate to proteins in a multivariate manner and their translation ratios are phenotype specific. *Nucleic acids research* **41,** 4743–4754 (2013).

53. Wang, Z.-Y. *et al.* Transcriptome and translatome co-evolution in mammals. *Nature* **588,** 642–647 (2020).

54. Zhang, M. *et al.* A peptide encoded by circular form of LINC-PINT suppresses oncogenic transcriptional elongation in glioblastoma. *Nature communications* **9,** 1–17 (2018).

55. Chen, X., Wei, S., Ji, Y., Guo, X. & Yang, F. Quantitative proteomics using SILAC: principles, applications, and developments. *Proteomics* **15,** 3175–3192 (2015).

56. Aviner, R., Geiger, T. & Elroy-Stein, O. Genome-wide identification and quantification of protein synthesis in cultured cells and whole tissues by puromycin-associated nascent chain proteomics (PUNCH-P). *Nature protocols* **9,** 751 (2014).

57. Moore, L. D., Le, T. & Fan, G. DNA methylation and its basic function. *Neuropsychopharmacology* **38,** 23–38 (2013).

58. Bibikova, M. *et al.* High density DNA methylation array with single CpG site resolution. *Genomics* **98,** 288–295 (2011).

59. Frommer, M. *et al.* A genomic sequencing protocol that yields a positive display of 5-methylcytosine residues in individual DNA strands. *Proceedings of the National Academy of Sciences* **89,** 1827–1831 (1992).

60. Oakes, C. C. *et al.* DNA methylation dynamics during B cell maturation underlie a continuum of disease phenotypes in chronic lymphocytic leukemia. *Nature genetics* **48,** 253–264 (2016).

61. Smallwood, S. A. *et al.* Single-cell genome-wide bisulfite sequencing for assessing epigenetic heterogeneity. *Nature methods* **11,** 817–820 (2014).

62. Wexler, P. *et al. Encyclopedia of toxicology* (Academic Press, 2005).

63. Pereira, D. & Williams, J. Origin and evolution of high throughput screening. *British journal of pharmacology* **152,** 53–61 (2007).

64. He, L. *et al.* in *Cancer systems biology* 351–398 (Springer, 2018).

65. Argelaguet, R. *et al.* Multi-Omics Factor Analysis—a framework for unsupervised integration of multi-omics data sets. *Molecular systems biology* **14,** e8124 (2018).

66. Berg, E. L., Kunkel, E. J., Hytopoulos, E. & Plavec, I. Characterization of compound mechanisms and secondary activities by BioMAP analysis. *Journal of pharmacological and toxicological methods* **53,** 67–74 (2006).

67. Fliri, A. F., Loging, W. T., Thadeio, P. F. & Volkmann, R. A. Analysis of drug-induced effect patterns to link structure and side effects of medicines. *Nature chemical biology* **1,** 389–397 (2005).

68. White, E. L. *et al.* A novel inhibitor of Mycobacterium tuberculosis pantothenate synthetase. *Journal of biomolecular screening* **12,** 100–105 (2007).

69. Klaeger, S. *et al.* The target landscape of clinical kinase drugs. *Science* **358** (2017).

70. Gunter, T. D. & Terry, N. P. The emergence of national electronic health record architectures in the United States and Australia: models, costs, and questions. *Journal of medical Internet research* **7,** e3 (2005).

71. Hoerbst, A & Ammenwerth, E. Electronic health records. *Methods Inf Med* **49,** 320–336 (2010).

72. Layman, E. J. Ethical issues and the electronic health record. *The health care manager* **39,** 150–161 (2020).

73. Yi, M. Major Issues in Adoption of Electronic Health Records. *Journal of Digital Information Management* **16** (2018).

74. Cowie, M. R. *et al.* Electronic health records to facilitate clinical research. *Clinical Research in Cardiology* **106,** 1–9 (2017).

75. Hicks, J. L. *et al.* Best practices for analyzing large-scale health data from wearables and smartphone apps. *NPJ digital medicine* **2,** 1–12 (2019).

76. Clarke, A. & Steele, R. Smartphone-based public health information systems: Anonymity, privacy and intervention. *Journal of the Association for Information Science and Technology* **66,** 2596–2608 (2015).

77. Huang, H., Zhang, R. & Lu, X. *A Recommendation Model for Medical Data Visualization Based on Information Entropy and Decision Tree Optimized by Two Correlation Coefficients* in *Proceedings of the 9th International Conference on Information Communication and Management* (2019), 52–56.

78. Moradian, N. *et al.* The urgent need for integrated science to fight COVID-19 pandemic and beyond. *Journal of translational medicine* **18,** 1–7 (2020).

79. Liu, Q. *et al.* A web visualization tool using T cell subsets as the predictor to evaluate COVID-19 patient's severity. *Plos one* **15,** e0239695 (2020).

80. Carter, R. A. *et al.* A Single-Cell Transcriptional Atlas of the Developing Murine Cerebellum. *Current Biology* **28,** 2910–2920.e2. ISSN: 09609822. https://linkinghub.elsevier.com/retrieve/pii/S0960982218309928 (2019) (Sept. 2018).

81. TM431. *RealTime-Glo^{TM} MT Cell Viability Assay Technical Manual* Promega Corporation (). https://www.promega.de/resources/protocols/technical-manuals/101/realtimeglo-mt-cell-viability-assay-protocol/.

82. TM375. *CellTox^{TM} Green Cytotoxicity Assay Technical Manual* Promega Corporation (). https://www.promega.de/resources/protocols/technical-manuals/101/celltox-green-cytotoxicity-assay-protocol/.

83. Yadav, B. *et al.* Quantitative scoring of differential drug sensitivity for individually optimized anticancer therapies. *Scientific reports* **4,** 1–10 (2014).

84. R Core Team. *R: A Language and Environment for Statistical Computing* R Foundation for Statistical Computing (Vienna, Austria, 2020). https://www.R-project.org/.

85. RStudio Team. *RStudio: Integrated Development Environment for R* RStudio, PBC. (Boston, MA, 2020). http://www.rstudio.com/.

86. Ihaka, R. & Gentleman, R. R: a language for data analysis and graphics. *Journal of computational and graphical statistics* **5,** 299–314 (1996).

87. Van Rossum, G. & Drake, F. L. *Python 3 Reference Manual* ISBN: 1441412697 (CreateSpace, Scotts Valley, CA, 2009).

88. Bezanson, J., Edelman, A., Karpinski, S. & Shah, V. B. Julia: A fresh approach to numerical computing. *SIAM review* **59,** 65–98 (2017).

89. Gentleman, R. C. *et al.* Bioconductor: open software development for computational biology and bioinformatics. *Genome biology* **5,** 1–16 (2004).

90. Ushey, K., Allaire, J. & Tang, Y. *reticulate: Interface to 'Python'* R package version 1.18 (2020). https://CRAN.R-project.org/package=reticulate.

91. Urbanek, S., Urbanek, M. S. & JDK, S. J. Package 'rJava' (2020).

92. Eddelbuettel, D. & François, R. Rcpp: Seamless R and C++ Integration. *Journal of Statistical Software* **40,** 1–18. https://www.jstatsoft.org/v40/i08/ (2011).

93. Eddelbuettel, D. *Seamless R and C++ Integration with Rcpp* ISBN 978-1-4614-6867-7 (Springer, New York, 2013).

94. Eddelbuettel, D. & Balamuta, J. J. Extending extitR with extitC++: A Brief Introduction to extitRcpp. *The American Statistician* **72,** 28–36. https://doi.org/10.1080/00031305.2017.1375990 (2018).

95. Wickham, H. *et al.* Welcome to the tidyverse. *Journal of Open Source Software* **4,** 1686 (2019).

96. Bengtsson, H. *A Unifying Framework for Parallel and Distributed Processing in R using Futures* 2020. arXiv: 2008.00553 [cs.DC]. https://arxiv.org/abs/2008.00553.

97. Hoefling, H. & Annau, M. *hdf5r: Interface to the 'HDF5' Binary Data Format* R package version 1.3.3 (2020). https://CRAN.R-project.org/package=hdf5r.

98. Wickham, H. *ggplot2: Elegant Graphics for Data Analysis* ISBN: 978-3-319-24277-4. https://ggplot2.tidyverse.org (Springer-Verlag New York, 2016).

99. Kolde, R. *pheatmap: Pretty Heatmaps* R package version 1.0.12 (2019). https://CRAN.R-project.org/package=pheatmap.

100. Postel, J. *et al.* Internet protocol (1981).

101. Deering, S., Hinden, R., *et al. Internet protocol, version 6 (IPv6) specification* 1998.

102. Postel, J. *et al.* Transmission control protocol (1981).

103. Postel, J. *et al.* User datagram protocol (1980).

104. Zimmermann, H. OSI reference model-the ISO model of architecture for open systems interconnection. *IEEE Transactions on communications* **28,** 425–432 (1980).

105. Fielding, R. *et al. Hypertext transfer protocol–HTTP/1.1* 1999.

106. Fette, I. & Melnikov, A. *The WebSocket protocol* RFC 6455 (Internet Engineering Task Force, 2011). https://tools.ietf.org/html/rfc6455.

107. Cheng, J. & Chang, W. *httpuv: HTTP and WebSocket Server Library* R package version 1.5.4 (2020). https://CRAN.R-project.org/package=httpuv.

108. Chang, W., Cheng, J., Dipert, A. & Borges, B. *websocket: 'WebSocket' Client Library* R package version 1.3.2 (2021). https://CRAN.R-project.org/package=websocket.

109. Chang, W. *R6: Encapsulated Classes with Reference Semantics* R package version 2.5.0 (2020). https://CRAN.R-project.org/package=R6.

110. Wickham, H. *Advanced r* (CRC press, 2019).

111. Ovchinnikova, S. & Anders, S. Exploring dimension-reduced embeddings with Sleepwalk. *Genome research* **30,** 749–756 (2020).

112. Ringnér, M. What is principal component analysis? en. *Nature Biotechnology* **26,** 303–304. ISSN: 1087-0156, 1546-1696. http://www.nature.com/articles/nbt0308-303 (2019) (Mar. 2008).

113. Kruskal, J. B. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. en. *Psychometrika* **29,** 1–27. ISSN: 1860-0980. https://doi.org/10.1007/BF02289565 (1964).

114. Kohonen, T. Self-organized formation of topologically correct feature maps. *Biological Cybernetics* **43,** 59–69. ISSN: 1432-0770. https://doi.org/10.1007/BF00337288 (Jan. 1982).

115. Butler, A., Hoffman, P., Smibert, P., Papalexi, E. & Satija, R. Integrating single-cell transcriptomic data across different conditions, technologies, and species. *Nature Biotechnology* **36,** 411–420. ISSN: 1087-0156, 1546-1696. http://www.nature.com/doifinder/10.1038/nbt.4096 (2019) (Apr. 2, 2018).

116. Satija Lab. *Using Seurat with multi-modal data* Mar. 24, 2018. https://satijalab.org/seurat/multimodal_vignette.html.

117. Van der Maaten, L. & Hinton, G. Visualizing High-Dimensional Data Using t-SNE. *Journal of Machine Learning Research* **9,** 2579–2605. http://www.jmlr.org/papers/v9/vandermaaten08a.html (2008).

118. McInnes, L., Healy, J. & Melville, J. *UMAP: Uniform Manifold Approximation and Projection for dimension Reduction* arXiv:1802.03426 [cs, stat]. Feb. 2018. http://arxiv.org/abs/1802.03426 (2019).

119. Angerer, P. *et al.* destiny: diffusion maps for large-scale single-cell data in R. *Bioinformatics* **32,** 1241–1243. ISSN: 1367-4803, 1460-2059. https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/btv715 (2019) (Apr. 15, 2016).

120. Coifman, R. R. & Lafon, S. Diffusion maps. *Applied and Computational Harmonic Analysis* **21,** 5–30 (2006).

121. Trapnell, C. *et al.* The dynamics and regulators of cell fate decisions are revealed by pseudotemporal ordering of single cells. *Nature Biotechnology* **32,** 381–386. ISSN: 1087-0156, 1546-1696. http://www.nature.com/articles/nbt.2859 (2019) (Apr. 2014).

122. Qiu, X. *et al.* Reversed graph embedding resolves complex single-cell trajectories. *Nature Methods* **14,** 979 (2017).

123. Mao, Q., Wang, L., Goodison, S. & Sun, Y. *Dimensionality Reduction Via Graph Structure Learning* in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (ACM, Sydney, NSW, Australia, 2015), 765–774. ISBN: 978-1-4503-3664-2. http://doi.acm.org/10.1145/2783258.2783309.

124. Nguyen, L. H. & Holmes, S. Ten quick tips for effective dimensionality reduction. *PLOS Computational Biology* **15,** 1–19. https://doi.org/10.1371/journal.pcbi.1006907 (June 2019).

125. Aupetit, M. Visualizing distortions and recovering topology in continuous projection techniques. *Neurocomputing* **70,** 1304–1330. ISSN: 09252312. https://linkinghub.elsevier.com/retrieve/pii/S0925231206004814 (2019) (Mar. 2007).

126. Kaski, S. *et al.* Trustworthiness and metrics in visualizing similarity of gene expression. *BMC Bioinformatics* **4,** 48 (2003).

127. Becht, E. *et al.* Dimensionality reduction for visualizing single-cell data using UMAP. *Nature Biotechnology* **37,** 38 (2019).

128. Tung, P.-Y. *et al.* Batch effects and the effective design of single-cell gene expression studies. *Scientific Reports* **7,** 39921. https://doi.org/10.1038/srep39921 (Jan. 3, 2017).

129. Haghverdi, L., Lun, A. T. L., Morgan, M. D. & Marioni, J. C. Batch effects in single-cell RNA-sequencing data are corrected by matching mutual nearest neighbors. *Nature Biotechnology* **36,** 421. https://doi.org/10.1038/nbt.4091 (Apr. 2, 2018).

130. Phillips, J. M. & Venkatasubramanian, S. *A Gentle Introduction to the Kernel Distance* arXiv:1103.1625. 2011. http://arxiv.org/abs/1103.1625.

131. Wang, B., Zhu, J., Pierson, E., Ramazzotti, D. & Batzoglou, S. Visualization and analysis of single-cell RNA-seq data by kernel-based similarity learning. *Nature Methods* **14,** 414 (2017).

132. Yang, L. & Jin, R. *Distance metric learning: A comprehensive survey* tech. rep. (Michigan State University, 2006). http://www.cs.cmu.edu/~liuy/frame_survey_v2.pdf.

133. Cayton, L. *Algorithms for manifold learning* tech. rep. CS2008-0923 (University of California at San Diego, 2005). http://www.lcayton.com/resexam.pdf.

134. Moon, K. R. *et al.* Manifold learning-based methods for analyzing single-cell RNA-sequencing data. *Current Opinion in Systems Biology* **7,** 36–46 (2018).

135. Buettner, F. *et al.* Computational analysis of cell-to-cell heterogeneity in single-cell RNA-sequencing data reveals hidden subpopulations of cells. *Nature Biotechnology* **33,** 155–160. https://doi.org/10.1038/nbt.3102 (Jan. 2015).

136. Brawand, D. *et al.* The evolution of gene expression levels in mammalian organs. *Nature* **478,** 343–348. ISSN: 0028-0836, 1476-4687. http://www.nature.com/articles/nature10532 (2019) (Oct. 2011).

137. Dietrich, S. *et al.* Drug-perturbation-based stratification of blood cancer. *Journal of Clinical Investigation* **128,** 427–445. https://www.jci.org/articles/view/93801 (Jan. 2018).

138. Green, D. A. A colour scheme for the display of astronomical intensity images. *Bulletin of the Astromical Society of India* **39,** 289–295 (2011).

139. 10x Genomics. *What is Cell Ranger?* 2019. https://support.10xgenomics.com/single-cell-gene-expression/software/pipelines/latest/what-is-cell-ranger.

140. Baglama, J., Reichel, L. & Lewis, B. W. *irlba: Fast Truncated Singular Value Decomposition and Principal Components Analysis for Large Dense and Sparse Matrices* R package version 2.3.3 (2019). https://CRAN.R-project.org/package=irlba.

141. Baglama, J. & Reichel, L. Augmented Implicitly Restarted Lanczos Bidiagonalization Methods. *SIAM Journal on Scientific Computing* **27,** 19–42 (2005).

142. Krijthe, J. H. *Rtsne: T-Distributed Stochastic Neighbor Embedding using Barnes-Hut Implementation* R package version 0.15 (2015). https://github.com/jkrijthe/Rtsne.

143. Van der Maaten, L. Accelerating t-SNE using Tree-Based Algorithms. *Journal of Machine Learning Research* **15,** 3221–3245. http://jmlr.org/papers/v15/vandermaaten14a.html (2014).

144. Melville, J. *uwot: The Uniform Manifold Approximation and Projection (UMAP) Method for Dimensionality Reduction* R package version 0.0.0.9010 (2019). https://github.com/jlmelville/uwot.

145. Pedersen, T. L. *ggforce: Accelerating ggplot2* R package version 0.2.0 (2019). https://CRAN.R-project.org/package=ggforce.

146. Ritchie, M. E. *et al.* limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Research* **43,** e47 (2015).

147. Buja, A., McDonald, J. A., Michalak, J. & Stuetzle, W. *Interactive data visualization using focusing and linking* in *Proceedings of the 2nd conference on Visualization'91* (1991), 156–163.

148. Love, M. I., Huber, W. & Anders, S. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome biology* **15,** 1–21 (2014).

149. Dudoit, S., Yang, Y. H., Callow, M. J. & Speed, T. P. Statistical methods for identifying differentially expressed genes in replicated cDNA microarray experiments. *Statistica sinica,* 111–139 (2002).

150. Lebeau, F. *d3-beeswarm plugin* 2017. https://bl.ocks.org/Kcnarf/5c989173d0e0c74ab4b62161b33bb0a8.

151. Hattab, H. E. *reveal.js: The HTML Presentation Framework* 2021. https://revealjs.com/.

152. Herbst, K. *et al.* Colorimetric RT-LAMP and LAMP-sequencing for detecting SARS-CoV-2 RNA in clinical samples. *Bio-protocol* **11,** e3964–e3964 (2021).

153. Deckert, A. *et al.* Effectiveness and cost-effectiveness of four different strategies for SARS-CoV-2 surveillance in the general population (CoV-Surv Study): a structured summary of a study protocol for a cluster-randomised, two-factorial controlled trial. *Trials* **22,** 1–4 (2021).

154. Notomi, T. *et al.* Loop-mediated isothermal amplification of DNA. *Nucleic acids research* **28,** e63–e63 (2000).

155. Wattenberg, M., Viégas, F. & Johnson, I. How to Use t-SNE Effectively. *Distill.* http://doi.org/10.23915/distill.00002 (2016).

156. Seifert, C., Sabol, V. & Kienreich, W. *Stress Maps: Analysing Local Phenomena in Dimensionality Reduction Based Visualisations* in *EuroVAST 2010: International Symposium on Visual Analytics Science and Technology* (eds Kohlhammer, J. & Keim, D.) (The Eurographics Association, 2010). ISBN: 978-3-905673-74-6.

157. Lespinats, S. & Aupetit, M. CheckViz: Sanity Check and Topological Clues for Linear and Non-Linear Mappings. *Computer Graphics Forum* **30,** 113–125. ISSN: 01677055. http://doi.wiley.com/10.1111/j.1467-8659.2010.01835.x (2019) (Mar. 2011).

158. Vaidyanathan, R. *et al. htmlwidgets: HTML Widgets for R* R package version 1.5.3 (2020). https://CRAN.R-project.org/package=htmlwidgets.