MONASH University

Fluent statistical computing interfaces for biological data analysis

Stuart Andrew Lee

B.MatComSci., University of Adelaide; MSc.Bioinformatics.

University of Melbourne

A thesis submitted for the degree of Doctor of Philosophy at

Monash University in 2020

Department of Econometrics and Business Statistics

Contents

Copyright notice

Abstract

Exploratory data analysis is vital to modern science workflows; it allows scientists to grasp problems with their data and generate new hypotheses. This work explores three facets of exploratory data analysis workflows as applied to biological data science: data wrangling, integration and visualisation. It contributes new statistical computing interfaces and frameworks with the explicit aim of enabling scientists to understand their data and models in their biological context. In chapter 2 we show that genomics data can be represented using tidy data semantics, and consequently the process of wrangling it can be simplified via our grammar of genomic data transformation. The next contribution is exploring the implications of our grammar on the integration and representation of genomics data. In chapter 3, we provide a framework for integrating genomics data from multiple assays, via combining model estimates over their genomic regions. Next we extend our grammar to represent single variable measurements along the genome in multiple ways; in chapter 4 we present a software tool that allows coverage scores to be aggregated and visualised over an experimental design and genomic features and use this tool to uncover intron signal in RNA-seq data. Finally, in chapter 5 we contribute a new visualisation interface that provides scientists with a toolkit for discovering structure in their high dimensional data, and assist them in understanding when non-linear dimension reduction has worked appropriately.

Declaration

I hereby declare that this thesis contains no material which has been accepted for the award of any other degree or diploma in any university or equivalent institution, and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

This thesis includes 4 publications, two of which have been published and two which have not been submitted yet (Lee, Cook, and Lawrence, 2019; Lee, Lawrence, and Love, 2020). As the core theme of my thesis is the development of software interfaces for biological data analysis, and given the collaborative nature of statistical computing and bioinformatics research; all of the included papers in this thesis reflect and acknowledge the contributions of my co-authors. Of special note is chapter 4, which reflects my contribution (in the form of a software package) to the submitted manuscript entitled *Covering all your bases: incorporating intron signal from RNA-seq data* (Lee et al., 2020), and includes new data analyses not found in the paper. The following table details the publications, including my and my fellow co-authors contributions:

Thesis Chapter	Publication Title	Status (published, in press, accepted or returned for revision)	Nature and % of student contribution	Co-author name(s) Nature and % of Co-authors contribution	Co-author(s), Monash student Y/N
2	**plyranges**: a grammar of genomic data transformation	Published	70%. Concept, software development, data analysis, and manuscript writing	(1) Dianne Cook, Concept and manuscript revision 10% (2) Michael Lawrence, Concept and software development 20%	(1) No (2) No
3	Fluent genomics with **plyranges** and **tximeta**	Published	60%. Concept, software development, data analysis and manuscript writing	(1) Michael Lawrence, manuscript feedback and editing 5% (2) Michael I Love, Concept, data analysis and manuscript writing 35%	(1) No (2) No

I have renumbered sections of submitted or published papers in order to generate a consistent presentation within the thesis.

Student name: Stuart Andrew Lee

Student signature:

Date: 05/08/2020

I hereby certify that the above declaration correctly reflects the nature and extent of the student's and co-authors' contributions to this work. In instances where I am not the responsible author I have consulted with the responsible author to agree on the respective contributions of the authors.

Supervisor name: Dianne Helen Cook

Supervisor signature:

Date: 05/08/2020

Acknowledgements

I have had the great privilege to work with and be inspired by many people during my PhD candidature. It has been a challenging and rewarding experience and I would not have been able to get through it without the support of my advisers, colleagues, friends and family.

I am deeply grateful to my supervisors: Dianne Cook, Matthew Ritchie, and Paul Harrison. Di has been instrumental in helping me grow as a researcher and teacher; she has been patient and always full of interesting insights and enthusiasm for the practice of doing data analysis. Thank you Di for your support and guidance throughout this process. I look forward to keep working with you in the future. Matt is one of the kindest and most generous scientists I know, thank you for allowing me to be a part of your lab and for all of the lab lunches! Paul has been a great advocate for me and my work, thank you for getting down in the trenches and always being available to help when I nag you about technical difficulties I'm having.

My PhD would have been very different if I did not have the chance to collaborate with and learn from Michael Lawrence. Thank you for hosting me at Genentech in San Francisco and pushing me to grow as a programmer and researcher. Likewise, I am thankful to have had the opportunity to work with Michael Love who has been an enthusiastic collaborator and supporter. Hopefully I can visit North Carolina soon!

Thank you to my PhD committee: David Frazier, David Powell, and Farshid Vahid. I am lucky to be in such a supportive environment at Monash EBS, a big thank you to Gael Martin and Rob Hyndman for their advice and guidance during the PhD.

I am thankful to be part of the broader R and Bioconductor communities. Special thanks to Ian Lyttle and Alicia Schep for answering all of my annoying questions and letting me contribute to the **vegawidget** package. I thank Earo Wang, Ursula Laa, Shian Su, and Nick Tierney for always being available for chats (but really most importantly coffee or snacks) about software development and visualisation.

I would like to thank my colleagues Charity Law, Peter Hickey, Alexandra Garnham, Hannah Coughlan, and Saskia Freytag for their friendship and guidance on all things statistical during my thesis. I look forward to having a drink with you in person.

I would not have been able to make it through this PhD if not for the support of friends and family. Thank you to the members of FCC, Casey, Clare, Cyrus, Joy and Margs, for always being a delight and making my sides hurt from laughter and fried chicken. Thank you Michael, Wai Keen, Rachel, Robbie, Anna, LP, Laura, Saskia, Kathy, Lan and Josh for making life more fun. A big thanks to my mum, dad and sister for always being supportive of my choices and listening to my complaints. Finally, thank you to my partner Joy for encouraging me to pursue a PhD and always believing in me, even when I don't believe in myself. I am a better person because of you and I am so happy I get to spend my life with you.

Preface

This thesis has been written using R Markdown with the **bookdown** package (Xie, 2016, 2017). The **renv** package has been used to create a reproducible environment to build the thesis from source (Ushey, 2019). All materials required to compile the thesis are available at `https://github.com/sa-lee/thesis`. An online version of the thesis is available to read at `https://sa-lee.github.io/thesis`.

Chapter 2 has been published in *Genome Biology*. It won the ACEMS Business Analytics Prize for Best Paper in 2019. Chapter 3 has been published in *F1000 Research*. Chapter 4 is based on my software contributions to the submitted article Lee et al. (2020). Chapter 5 has not been submitted.

Chapter 1

Introduction

Exploratory data analysis (EDA) is a vital element of the modern statistical workflow - it is an analyst's first pass at understanding their data; revealing all its messes and uncovering hidden insights (Tukey, 1977; Grolemund and Wickham, 2017). It is an iterative process involving computation and visualization, leading to new hypotheses that can be tested and formalised using statistical modelling (Figure 1.1). As datasets grow in complexity and become increasingly heterogeneous and multidimensional, the use of EDA becomes vital to ensure the integrity and quality of analysis outputs. This is certainly true in high-throughput biological data science, where constraints on computation time and memory, in addition to the analyst's time, makes EDA difficult and neglected, which impacts the robustness and reliability of any downstream analysis.

This thesis focuses on core aspects of EDA as part of a biological data science workflow: wrangling, integration and visualisation, with a focus on applications to genomics and transcriptomics. To begin we discuss wrangling biological data using a coherent representation and programming interface (Figure 1.2). Section 1.1 introduces a grammar-based framework for transforming genomics data that is described in Chapter 2. We then look at integrating data and model outputs over genomic regions to gain biological insight (1.3). Section 1.2 introduces a framework for incorporating genomic regions over multiple assays, described in Chapter 3, while 1.3 discusses finding 'interesting' genomic regions via combining multiple summaries of a single assay, described in 4. Next, we consider

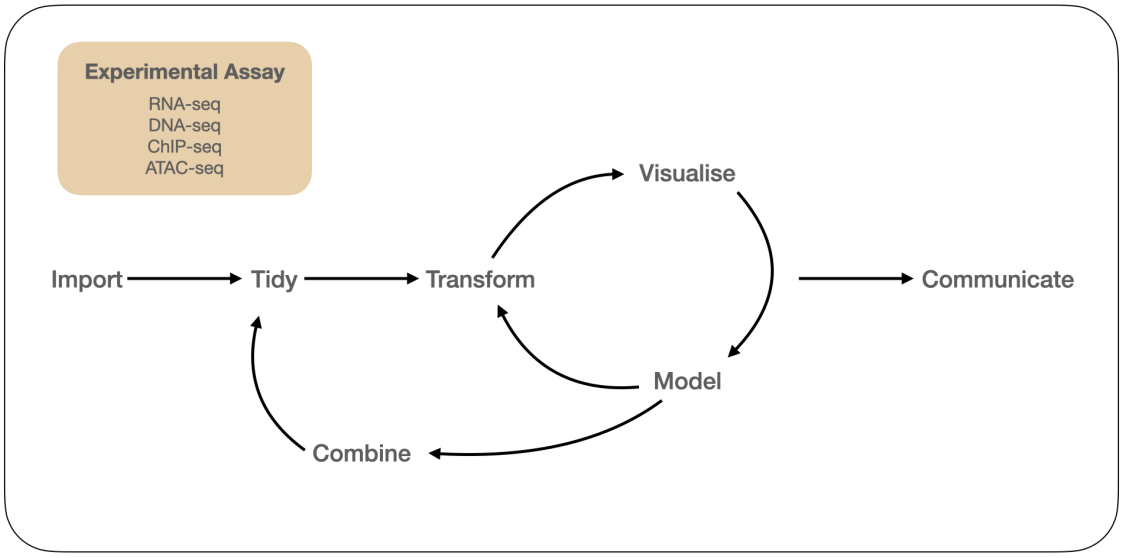


Figure 1.1: *An idealised model of the biological data science workflow (adapted from Grolemund and Wickham (2017)). We begin with data generated from one or more biological assay(s) corresponding to a research hypothesis or question. Our primary focus is on data generated from bulk assays that measure gene expression (RNA-seq), genetic variation (DNA-seq), and gene regulation (ChIP-seq, ATAC-seq). Throughout this process, we need computational tools to gain insight into the biology under study and communicate our analysis in a reproducible manner.*

the challenges in visualising high dimensional data (Figure 1.4). Section 1.4 introduces an interactive visualisation approach for understanding non-linear dimension reduction techniques described in Chapter 5. Lastly, in Chapter 6 describes the outputs of the thesis and plans for future developments.

1.1 A grammar for genomic data analysis

The approach taken by the suite of software packages collectively known as the **tidyverse** is an attempt to formalise aspects of the EDA process in the R programming language under a single semantic known as *tidy data* (R Core Team, 2019; Wickham et al., 2019; Wickham, 2014). Simply put, a *tidy data* set is a rectangular table where each row of the table corresponds to an observation, each column corresponds to a variable and each cell a value. There is a surprisingly large amount of utility that can be achieved with this definition. By having each column representing a variable, variables in the data can be mapped to graphical aesthetics of plots. This paradigm enables the grammar of

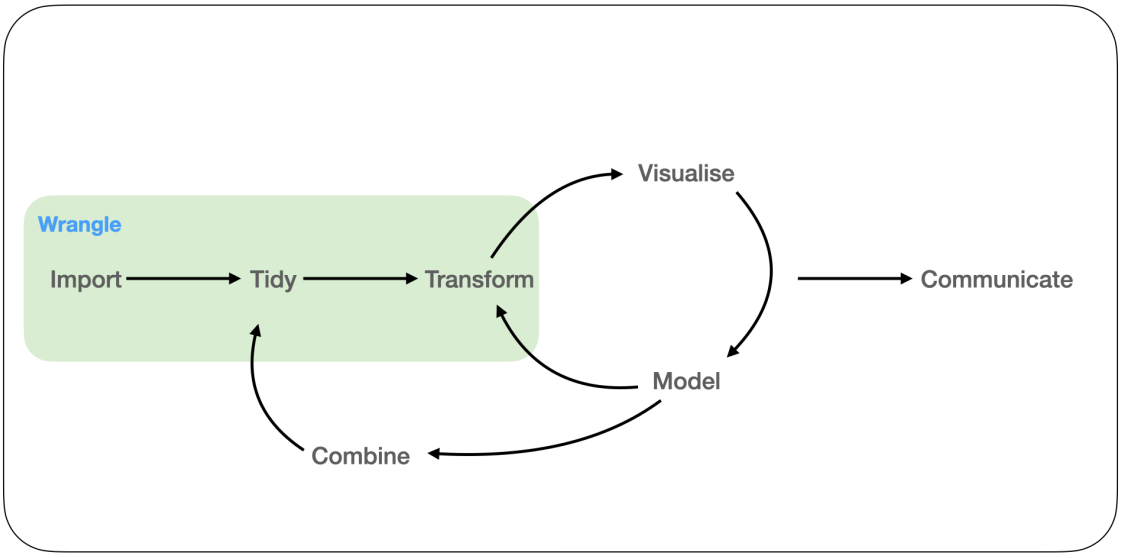


Figure 1.2: *In the wrangle phase of the workflow, data from an assay is imported into a programming language. It is then tidied into a new representation that should capture the biological semantics of the measurements. Following this, the representation can be transformed to generate new summaries.*

graphics as implemented by **ggplot2** (Wickham, 2016; Wilkinson, 2005). User interfaces as implemented by **tidyverse**, and in particular the **dplyr** package, are *fluent*; they form a domain specific language (DSL) that gives users a mental model for performing and composing common data transformation tasks (Wickham et al., 2017; Fowler, n.d.).

It is unclear whether the *fluent* interfaces as implemented using the *tidy data* framework can be more generally applied and useful in fields such as high-throughput biology where domain specific semantics are required (Figure 1.2). This is particularly true in the Bioconductor ecosystem, where much thought has gone into the design of data structures that enable interoperability between different tools, biological assays and analysis goals (Huber et al., 2015a).

Chapter 2 shows that the *tidy data* semantic is applicable to in memory data measured along the genome and develops a *fluent* interface to transforming it called **plyranges**. The software provides a framework to an assist an analyst to compose queries on genomics datasets. Our software is agnostic to how counts from bulk assays have been obtained.

Indeed, we have used data obtained from both alignment and quantification based approaches throughout the thesis to perform useful analyses.

This chapter has been published as Lee, Cook, and Lawrence (2019).

1.2 Integration of genomic data structures

It is rare that a biological data analysis will involve a single measurement assay or that only one aspect of a measurement assay will be of interest to the biological question under study (Figure 1.3). While there are many approaches to integrating data sets from multiple assays using multivariate statistical techniques (Meng et al., 2016; Stein-O'Brien et al., 2018) and data structures to represent them (Ramos et al., 2017), there has been little thought given to the interoperability between these approaches and the **tidyverse**. In Chapter 3 we describe a simple end-to-end workflow for integrating results along the genome using **plyranges**. This workflow shows that our grammar based approach does not impair interoperability between the **tidyverse** and Bioconductor approaches, and in

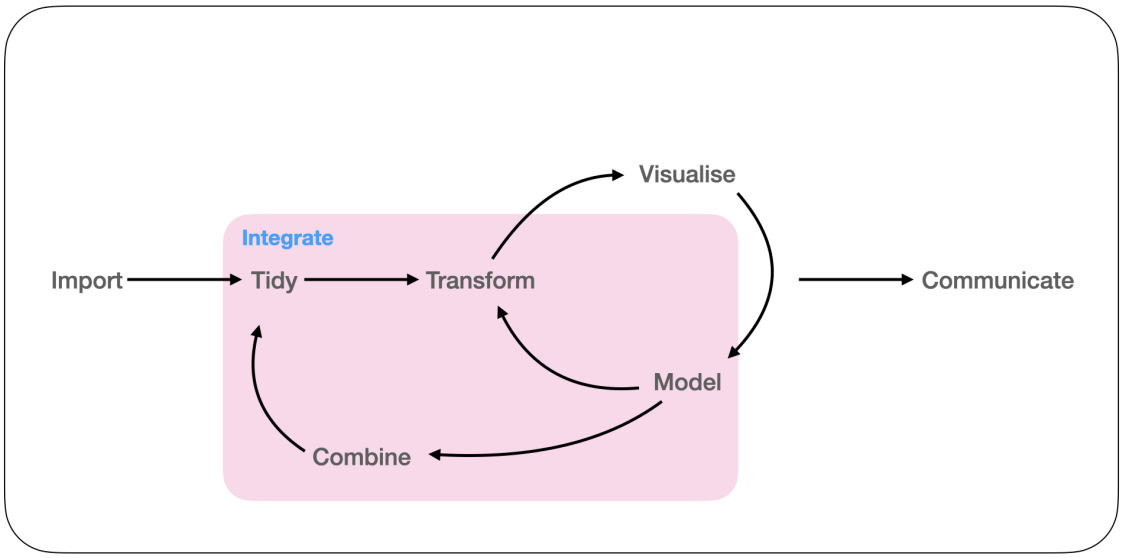


Figure 1.3: *Data from multiple assays can be combined together using modelling and transformations to gain biological insight. This is achieved either via a statistical technique or more simply via joining data and model results so they have a common representation across different granularities of the genome.*

fact they work seamlessly together. This chapter has been published as Lee, Lawrence, and Love (2020).

1.3 Representation of genomic data structures

In Chapter 4 we explore the limits of the *tidy data* semantic by extending **plyranges** to analyse coverage estimated on RNA-seq data by developing a new software tool called **superintronic**. We show that the long-form tidy representation is an effective way of combining the experimental design and reference annotations into a single genomic data structure for exploration. We use **superintronic** to develop a framework for discovering interesting regions of coverage and apply our approach to integrating intron signal from RNA-seq data. This chapter is based on my software and analysis contributions to the Lee et al. (2020).

1.4 Interactive visualisation for high-dimensional data

Finally, we move away from data wrangling and towards the integration of visualisation with model-based summaries of high-dimensional data sets (Figure 1.4). We focus on a common tool for EDA (especially applied to single-cell transcriptomics): non-linear dimension reduction (NLDR). We consider the incorporation of interactive and dynamic graphics to assist analysts in using NLDR techniques for cluster orientation tasks. In

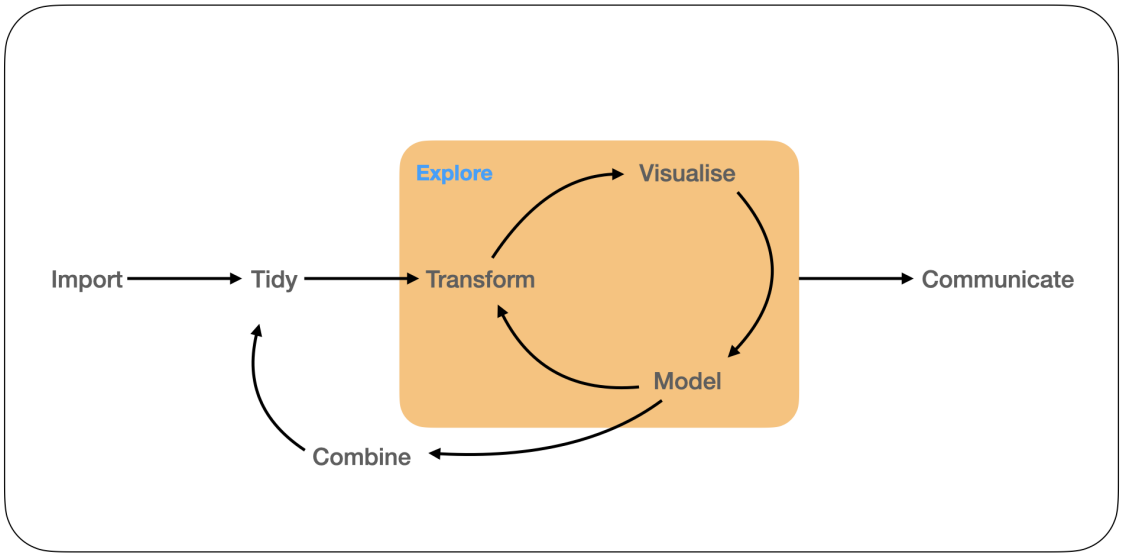


Figure 1.4: *Visualisation is a tool for making sense of models. Here we explore, how well a model has captured the structure within the data using interactive graphics.*

particular, we advocate for the use of tours (Cook et al., 1995) alongside an NLDR visualisation to highlight potential pitfalls and distortions obtained from an NLDR method. This approach acknowledges that there is no 'one' best visualisation or dimension reduction for a high-dimensional dataset, and we often want to have an understanding of both the global and local structure within our data.

Chapter 5 introduces a software package called **liminal** for constructing these views and a user interaction framework for identifying distortions. We present several case studies using data that capture aspects of single cell transcriptomics workflows, and use our approach to diagnose the quality of results obtained via popular NLDR methods like t-distributed stochastic neighbour embeddings (t-SNE) (Maaten and Hinton, 2008).

Chapter 2

plyranges: a grammar of data transformation for genomics

There is a cognitive load placed on users in learning a data abstraction from the Bioconductor project and understanding its appropriate use. Users must navigate these abstractions to perform a genomic analysis task, when a single data abstraction, a *GRanges* object will suffice. By recognizing that the *GRanges* class follows 'tidy' data principles, we create a grammar of genomic data transformation, defining verbs for performing actions on and between genomic interval data and providing a way of performing common data analysis tasks through a coherent interface to existing Bioconductor infrastructure. We implement this grammar as a Bioconductor/R package called **plyranges**.

2.1 Background

High-throughput genomics promises to unlock new disease therapies, and strengthen our knowledge of basic biology. To deliver on those promises, scientists must derive a stream of knowledge from a deluge of data. Genomic data is challenging in both scale and complexity. Innovations in sequencing technology often outstrip our capacity to process the output. Beyond their common association with genomic coordinates, genomic data are heterogeneous, consisting of raw sequence read alignments, genomic feature annotations like genes and exons, and summaries like coverage vectors, ChIP-seq peak calls, variant

calls, and per-feature read counts. Genomic scientists need software tools to wrangle the different types of data, process the data at scale, test hypotheses, and generate new ones, all while focusing on the biology, not the computation. For the tool developer, the challenge is to define ways to model and operate on the data that align with the mental model of scientists, and to provide an implementation that scales with their ambition.

Several domain specific languages (DSLs) enable scientists to process and reason about heterogeneous genomics data by expressing common operations, such as range manipulation and overlap-based joins, using the vocabulary of genomics. Their implementations either delegate computations to a database, or operate over collections of files in standard formats like BED. An example of the former is the **Genome Query Language (GQL)** and its distributed implementation **GenAp** which use a SQL-like syntax for fast retrieval of information of unprocessed sequencing data (Kozanitis, Christos et al., 2014; Kozanitis and Patterson, 2016). Similarly, the **Genometric Query Language (GMQL)** implements a DSL for combining genomic datasets (Kaitoua, A et al., 2017). The command line application **BEDtools** develops an extensive algebra for performing arithmetic between two or more sets of genomic regions (Quinlan and Hall, 2010). All of the aforementioned DSLs are designed to be evaluated either at the command line or embedded in scripts for batch processing. They exist in a sparse ecosystem, mostly consisting of UNIX and database tools that lack biological semantics and operate at the level of files and database tables.

The Bioconductor/R packages **IRanges** and **GenomicRanges** (R Core Team, 2019; Lawrence et al., 2013a; Huber et al., 2015a) define a DSL for analyzing genomics data with R, an interactive data analysis environment that encourages reproducibility and provides high-level abstractions for manipulating, modelling and plotting data, through state of the art methods in statistical computing. The packages define object-oriented (OO) abstractions for representing genomic data and enable interoperability by allowing users and developers to use these abstractions in their own code and packages. Other genomic DSLs that are embedded in programming languages include **pybedtools** and **valr** (Dale, Pedersen, and Quinlan, 2011; Riemondy et al., 2017), however these packages lack the interoperability provided by the aforementioned Bioconductor packages and are not easily extended.

The Bioconductor infrastructure models the genomic data and operations from the perspective of the power user, one who understands and wants to take advantage of the subtle differences in data types. This design has enabled the development of sophisticated tools, as evidenced by the hundreds of packages depending on the framework. Unfortunately, the myriad of data structures have overlapping purposes and important but obscure differences in behavior that often confuse the typical end user.

Recently, there has been a concerted, community effort to standardize R data structures and workflows around the notion of tidy data (Wickham, 2014). A tidy dataset is defined as a tabular data structure that has observations as rows and columns as variables, and all measurements pertain to a single observational unit. The tidy data pattern is useful because it allows us to see how the data relate to the design of an experiment and the variables measured. The **dplyr** package (Wickham et al., 2017) defines an application programming interface (API) that maps notions from the general relational algebra to verbs that act on tidy data. These verbs can be composed together on one or more tidy datasets with the pipe operator from the **magrittr** package (Bache and Wickham, 2014). Taken together these features enable a user to write human readable analysis workflows.

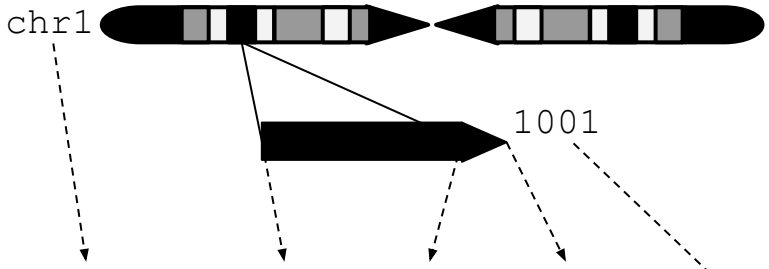
We have created a genomic DSL called **plyranges** that reformulates notions from existing genomic algebras and embeds them in R as a genomic extension of **dplyr**. By analogy, **plyranges** is to the genomic algebra, as **dplyr** is to the relational algebra. The **plyranges** Bioconductor package implements the language on top of a key subset of Bioconductor data structures and thus fully integrates with the Bioconductor framework, gaining access to its scalable data representations and sophisticated statistical methods.

2.2 Results

2.2.1 Genomic Relational Algebra

Data Model

The **plyranges** DSL is built on the core Bioconductor data structure *GRanges*, which is a constrained table, with fixed columns for the chromosome, start and end coordinates, and the strand, along with an arbitrary set of additional columns, consisting of measurements



seqname	start	end	strand	gene_id	score
chr1	102012	120303	+	1001	10
chr1	520211	526211	-	2151	25
...	...	...	...	...	...

Figure 2.1: *An illustration of the* GRanges *data model for a sample from an RNA-seq experiment. The core components of the data model include a seqname column (representing the chromosome), a ranges column which consists of start and end coordinates for a genomic region, and a strand identifier (either positive, negative, or unstranded). Metadata are included as columns to the right of the dotted line as annotations (gene_id) or range level covariates (score).*

or metadata specific to the data type or experiment (Figure 2.1). *GRanges* balances flexibility with formal constraints, so that it is applicable to virtually any genomic workflow, while also being semantically rich enough to support high-level operations on genomic ranges. As a core data structure, *GRanges* enables interoperability between **plyranges** and the rest of Bioconductor. Adhering to a single data structure simplifies the API and makes it easier to learn and understand, in part because operations become endomorphic, i.e., they return the same type as their input.

GRanges follow the intuitive tidy data pattern: it is a rectangular table corresponding to a single biological context. Each row contains a single observation and each column is a variable describing the observations. *GRanges* specializes the tidy pattern in that the observations always pertain to some genomic feature, but it largely remains compatible with the general relational operations defined by **dplyr**. Thus, we define our algebra as an extension of the **dplyr** algebra, and borrow its syntax conventions and design principles.

	Verb	Description
Aggregate	**`summarize()`**	aggregate over column(s)
	`disjoin_ranges()`	aggregate column(s) over the union of end coordinates
	`reduce_ranges()`	aggregate column(s) by merging overlapping and neighboring ranges
Modify (Unary)	**`mutate()`**	modifies any column
	`select()`	select columns
	`arrange()`	sort by columns
	`stretch()`	extend range by fixed amount
	`shift_(direction)`	shift coordinates
	`flank_(direction)`	generate flanking regions
Modify (Binary)	`%intersection%`	row-wise intersection
	`%union%`	row-wise union
	`compute_coverage`	coverage over all ranges
	`%setdiff%`	row-wise set difference
	`between()`	row-wise gap range
	`span()`	row-wise spanning range
Merge	`join_overlap_*()`	merge by overlapping ranges
	`join_nearest`	merge by nearest neighbor ranges
	`join_follow`	merge by following ranges
	`join_precedes`	merge by preceding ranges
	`union_ranges`	range-wise union
	`intersect_ranges`	range-wise intersect
	`setdiff_ranges`	range-wise set difference
	`complement_ranges`	range-wise set complement
Operate	`anchor_direction()`	fix coordinates at direction
	`group_by()`	partition by column(s)
	`group_by_overlaps()`	partition by overlaps
Restrict	**`filter()`**	subset rows
	`filter_by_overlaps()`	subset by overlap
	`filter_by_non_overlaps()`	subset by no overlap

Table 2.1: *Overview of the **plyranges** grammar. The core verbs are briefly described and categorized into one of the following higher level categories: aggregate, modify, merge, operate, or restrict. A verb is given bold text if its origin is from the **dplyr** grammar.*

Algebraic operations

The **plyranges** DSL defines an expressive algebra for performing genomic operations with and between *GRanges* objects (see table 2.1). The grammar includes several classes of operation that cover most use cases in genomics data analysis. There are range arithmetic operators, such as for resizing ranges or finding their intersection, and operators for merging, filtering and aggregating by range-specific notions like overlap and proximity.

Arithmetic operations transform range coordinates, as defined by their *start*, *end* and *width*. The three dimensions are mutually dependent and partially redundant, so direct manipulation of them is problematic. For example, changing the *width* column needs to change either the *start*, *end* or both to preserve integrity of the object. We introduce the *anchor* modifier to disambiguate these adjustments. Supported anchor points include the start, end and midpoint, as well as the 3′ and 5′ ends for strand-directed ranges. For example, if we anchor the start, then setting the width will adjust the end while leaving the start stationary.

The algebra also defines conveniences for relative coordinate adjustments: *shift* (unanchored adjustment to both start and end) and *stretch* (anchored adjustment of width). We can perform any relative adjustment by some combination of those two operations. The *stretch* operation requires an anchor and assumes the midpoint by default. Since *shift* is unanchored, the user specifies a suffix for indicating the direction: left/right or, for stranded features, upstream/downstream. For example, `shift_right()` shifts a range to the right.

The *flank* operation generates new ranges that are adjacent to existing ones. This is useful, for example, when generating upstream promoter regions for genes. Analogous to *shift*, a suffix indicates the side of the input range to flank.

As with other genomic grammars, we define set operations that treat ranges as sets of integers, including *intersect*, *union*, *difference*, and *complement*. There are two sets of these: parallel and merging. For example, the parallel intersection (`x %intersect% y`) finds the intersecting range between xi and yi for i in $1 \ldots n$, where n is the length of both x and y. In contrast, the merging intersection (`intersect_ranges(x, y)`) returns a new

set of disjoint ranges representing wherever there was overlap between a range in x and a range in y. Finding the parallel union will fail when two ranges have a gap, so we introduce a `span()` operator that takes the union while filling any gap. The `complement()` operation is unique in that it is unary. It finds the regions not covered by any of the ranges in a single set. Closely related is the `between()` parallel operation, which finds the gap separating *xi* and *yi*. The binary operations are callable from within arithmetic, restriction and aggregation expressions.

To support merging, our algebra recasts finding overlaps or nearest neighbors between two genomic regions as variants of the relational join operator. A join acts on two *GRanges* objects: x and y. The join operator is relational in the sense that metadata from the x and y ranges are retained in the joined range. All join operators in the **plyranges** DSL generate a set of hits based on overlap or proximity of ranges and use those hits to merge the two datasets in different ways. There are four supported matching algorithms: *overlap*, *nearest*, *precede*, and *follow* (Figure 2.2). We can further restrict the matching by whether the query is completely *within* the subject, and adding the *directed* suffix ensures that matching ranges have the same direction (strand).

For merging based on the hits, we have three modes: *inner*, *intersect* and *left*. The *inner* overlap join is similar to the conventional inner join in that there is a row in the result for every match. A major difference is that the matching is not by identity, so we have to choose one of the ranges from each pair. We always choose the left range. The *intersect* join uses the intersection instead of the left range. Finally, the overlap *left* join is akin to left outer join in Codd's relational algebra: it performs an overlap inner join but also returns all x ranges that are not hit by the y ranges.

Since the *GRanges* object is a tabular data structure, our grammar includes operators to filter, sort and aggregate by columns in a *GRanges*. These operations can be performed over partitions formed using the `group_by()` modifier. Together with our algebra for arithmetic and merging, these operations conform to the semantics and syntax of the **dplyr** grammar. Consequently, **plyranges** code is generally more compact than the equivalent **GenomicRanges** code (Figure 2.3).

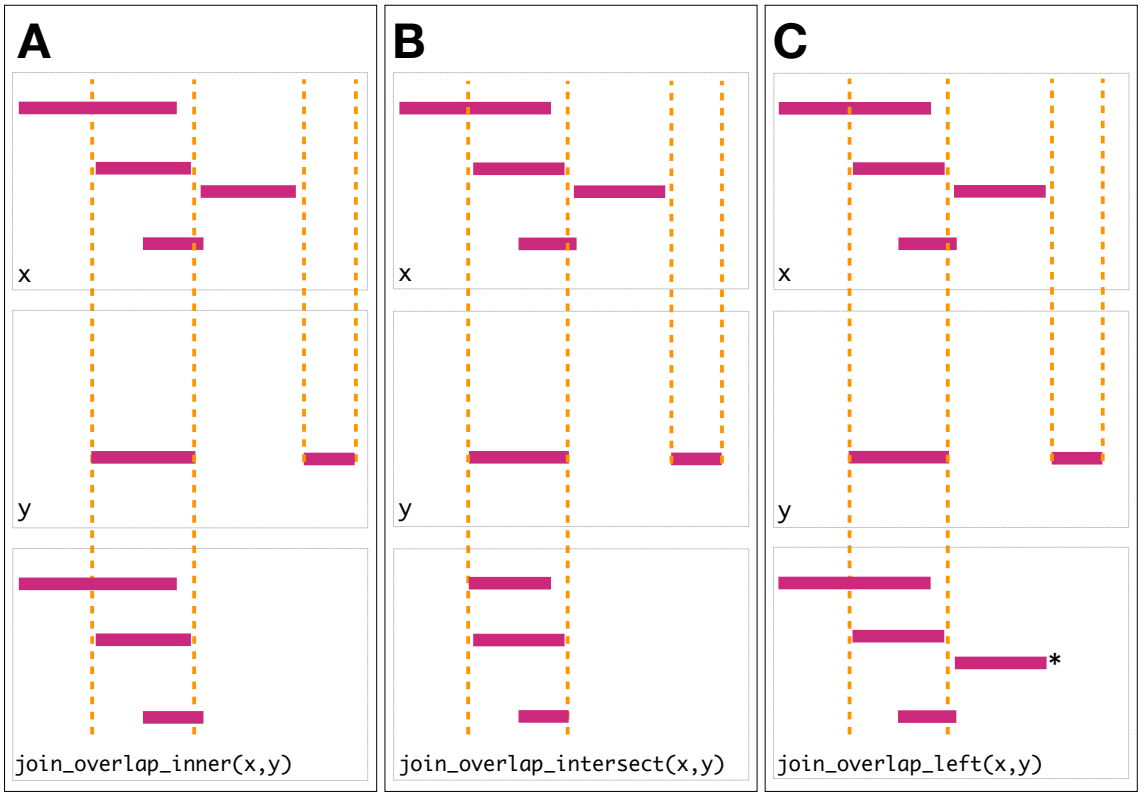


Figure 2.2: *Illustration of the three overlap join operators. Each join takes two* GRanges *objects, x and y as input. A 'Hits' object for the join is computed which consists of two components. The first component contains the indices of the ranges in x that have been overlapped (the rectangles of x that cross the orange lines). The second component consists of the indices of the ranges in y that overlap the ranges in x. In this case a range in y overlaps the ranges in x three times, so the index is repeated three times. The resulting 'Hits' object is used to modify x by where it was 'hit' by y and merge all metadata columns from x and y based on the indices contained in the 'Hits' object. This procedure is applied generally in the* ***plyranges*** *DSL for both overlap and nearest neighbor operations. The join semantics alter what is returned:* ***A****: for an* ***inner*** *join the x ranges that are overlapped by y are returned. The returned ranges also include the metadata from the y range that overlapped the three x ranges.* ***B*** *An* ***intersect*** *join is identical to an inner join except that the intersection is taken between the overlapped x ranges and the y ranges.* ***C*** *For the* ***left*** *join all x ranges are returned regardless of whether they are overlapped by y. In this case the third range (rectangle with the asterisk next to it) of the join would have missing values on metadata columns that came from y.*

A
```
library(plyranges)
gwas <- read_bed('snps.bed')
exons <- read_bed('exons.bed')
res <- exons %>%
          join_overlap_inner(snps) %>%
          group_by(rsID) %>%
          summarise(n = n_distinct(exonID))
```

B
```
library(GenomicRanges)
library(rtracklayer)
gwas  <- import('snps.bed')
exons <- import('exons.bed')
hits <- findOverlaps(exons, gwas,
                     ignore.strand = FALSE)
olap <- splitAsList(exons$name[queryHits(hits)],
                    gwas$name[subjectHits(hits)])
n <- lengths(unique(olap))
res <- DataFrame(rsID = names(n),
                 n = as.integer(n))
```

Figure 2.3: *Idiomatic code examples for **plyranges** (**A**) and **GenomicRanges** (**B**) illustrating an overlap and aggregate operation that returns the same result. In each example, we have two BED files consisting of SNPs that are genome-wide association study (GWAS) hits and reference exons. Each code block counts for each SNP the number of distinct exons it overlaps. The **plyranges** code achieves this with an overlap join followed by partitioning and aggregation. Strand is ignored by default here. The **GenomicRanges** code achieves this using the* Hits *and* List *classes and their methods.*

2.2.2 Developing workflows with plyranges

Here we provide illustrative examples of using the **plyranges** DSL to show how our grammar could be integrated into genomic data workflows. As we construct the workflows we show the data output intermittently to assist the reader in understanding the pipeline steps. The workflows highlight how interoperability with existing Bioconductor infrastructure, enables easy access to public datasets and methods for analysis and visualization.

Peak Finding

In the workflow of ChIP-seq data analysis, we are interested in finding peaks from islands of coverage over chromosome. Here we will use **plyranges** to call peaks from islands of coverage above 8 then plot the region surrounding the tallest peak.

Using **plyranges** and the Bioconductor package **AnnotationHub** (Morgan, 2017) we can download and read BigWig files from ChIP-Seq experiments from the Human Epigenome Roadmap project (Roadmap Epigenomics Consortium et al., 2015). Here we analyse a BigWig file corresponding to H3 lysine 27 trimethylation (H3K27Me3) of primary T CD8+ memory cells from peripheral blood, focussing on coverage islands over chromosome 10.

First, we extract the genome information from the BigWig file and filter to get the range for chromosome 10. This range will be used as a filter when reading the file.

```
library(plyranges)
chr10_ranges <- bw_file %>%
  get_genome_info() %>%
  filter(seqnames == "chr10")
```

Then we read the BigWig file only extracting scores if they overlap chromosome 10. We also add the genome build information to the resulting ranges. This book-keeping is good practice as it ensures the integrity of any downstream operations such as finding overlaps.

```
chr10_scores <- bw_file %>%
  read_bigwig(overlap_ranges = chr10_ranges) %>%
  set_genome_info(genome = "hg19")
chr10_scores
```

```
#> GRanges object with 5789841 ranges and 1 metadata column:
#>             seqnames              ranges strand |     score
#>                <Rle>           <IRanges>  <Rle> | <numeric>
#>         [1]    chr10             1-60602      * |   0.04228
#>         [2]    chr10         60603-60781      * |   0.16324
```

```
#>         [3]    chr10         60782-60816      * |   0.37214
#>         [4]    chr10         60817-60995      * |   0.16324
#>         [5]    chr10         60996-61625      * |   0.04228
#>         ...      ...                 ...    ... .       ...
#>   [5789837]    chr10 135524723-135524734      * |   0.14432
#>   [5789838]    chr10 135524735-135524775      * |   0.25023
#>   [5789839]    chr10 135524776-135524784      * |   0.42779
#>   [5789840]    chr10 135524785-135524806      * |   0.73002
#>   [5789841]    chr10 135524807-135524837      * |   1.03103
#>   -------
#>   seqinfo: 25 sequences from hg19 genome
```

We then filter for regions with a coverage score greater than 8, and following this reduce individual runs to ranges representing the islands of coverage. This is achieved with the `reduce_ranges()` function, which allows a summary to be computed over each island: in this case we take the maximum of the scores to find the coverage peaks over chromosome 10.

```
all_peaks <- chr10_scores %>%
  filter(score > 8) %>%
  reduce_ranges(score = max(score))
all_peaks
```

```
#> GRanges object with 1085 ranges and 1 metadata column:
#>          seqnames              ranges strand |     score
#>             <Rle>           <IRanges>  <Rle> | <numeric>
#>      [1]    chr10     1299144-1299370      * |  13.22640
#>      [2]    chr10     1778600-1778616      * |   8.20512
#>      [3]    chr10     4613068-4613078      * |   8.76027
#>      [4]    chr10     4613081-4613084      * |   8.43660
#>      [5]    chr10             4613086      * |   8.11508
#>      ...      ...                 ...    ... .       ...
```

```
#>   [1081]    chr10 135344482-135344488      * |   9.23238
#>   [1082]    chr10 135344558-135344661      * |  11.84341
#>   [1083]    chr10 135344663-135344665      * |   8.26966
#>   [1084]    chr10 135344670-135344674      * |   8.26966
#>   [1085]    chr10 135345440-135345441      * |   8.26966
#>   -------
#>   seqinfo: 25 sequences from hg19 genome
```

Returning to the *GRanges* object containing normalized coverage scores, we filter to find the coordinates of the peak containing the maximum coverage score. We can then find a 5000 nt region centered around the maximum position by anchoring and modifying the width.

Finally, the overlap inner join is used to restrict the chromosome 10 coverage islands, to the islands that are contained in the 5000nt region that surrounds the max peak (Figure 2.4).

```
#> GRanges object with 890 ranges and 2 metadata columns:
#>         seqnames            ranges strand |   score.x   score.y
#>            <Rle>         <IRanges>  <Rle> | <numeric> <numeric>
#>     [1]    chr10 21805891-21805988      * |   0.02066   29.9573
#>     [2]    chr10 21805989-21806000      * |   0.02112   29.9573
#>     [3]    chr10 21806001-21806044      * |   0.02207   29.9573
#>     [4]    chr10 21806045-21806049      * |   0.02159   29.9573
#>     [5]    chr10 21806050-21806081      * |   0.02112   29.9573
#>     ...      ...               ...    ... .       ...       ...
#>   [886]    chr10          21810878      * |   5.24952   29.9573
#>   [887]    chr10          21810879      * |   5.83534   29.9573
#>   [888]    chr10 21810880-21810884      * |   6.44268   29.9573
#>   [889]    chr10 21810885-21810895      * |   7.07055   29.9573
#>   [890]    chr10 21810896-21810911      * |   6.44268   29.9573
```

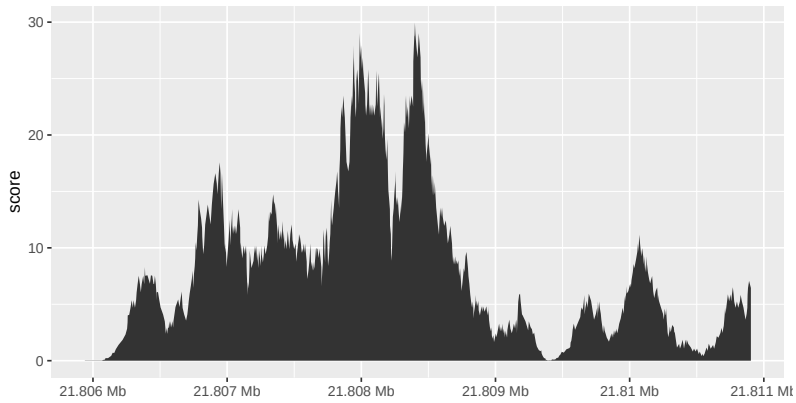


Figure 2.4: *The final result of the **plyranges** operations to find a 5000nt region surrounding the peak of normalised coverage scores over chromosome 10, displayed as a density plot.*

```
#>   -------
#>   seqinfo: 25 sequences from hg19 genome
```

Computing Windowed Statistics

Another common operation in genomics data analysis is to compute data summaries over genomic windows. In **plyranges** this can be achieved via the `group_by_overlaps()` operator. We bin and count and find the average GC content of reads from a H3K27Me3 ChIP-seq experiment by the Human Epigenome Roadmap Consortium.

We can directly obtain the genome information from the header of the BAM file: in this case the reads were aligned to the hg19 genome build and there are no reads overlapping the mitochondrial genome.

```
bam <- read_bam(h1_bam_sorted, index = h1_bam_sorted_index)
locations <- bam %>%
  get_genome_info()
```

Next we only read in alignments that overlap the genomic locations we are interested in and select the query sequence. Note that the reading of the BAM file is deferred: only alignments that pass the filter are loaded into memory. We can add another column representing the GC proportion for each alignment using the `letterFrequency()` function

from the **Biostrings** package (Pagès et al., 2018). After computing the GC proportion as the score column, we drop all other columns in the *GRanges* object.

```
alignments <- bam %>%
  filter_by_overlaps(locations) %>%
  select(seq) %>%
  mutate(
    score = as.numeric(letterFrequency(seq, "GC", as.prob = TRUE))
  ) %>%
  select(score)
alignments
```

```
#> GRanges object with 8275595 ranges and 1 metadata column:
#>             seqnames            ranges strand |     score
#>                <Rle>         <IRanges>  <Rle> | <numeric>
#>         [1]    chr10       50044-50119      - |  0.276316
#>         [2]    chr10       50050-50119      + |  0.250000
#>         [3]    chr10       50141-50213      - |  0.447368
#>         [4]    chr10       50203-50278      + |  0.263158
#>         [5]    chr10       50616-50690      + |  0.276316
#>         ...      ...               ...    ... .       ...
#>   [8275591]     chrY 57772745-57772805      - |  0.513158
#>   [8275592]     chrY 57772751-57772800      + |  0.526316
#>   [8275593]     chrY 57772767-57772820      + |  0.565789
#>   [8275594]     chrY 57772812-57772845      + |  0.250000
#>   [8275595]     chrY 57772858-57772912      + |  0.592105
#>   -------
#>   seqinfo: 24 sequences from an unspecified genome
```

Finally, we create 10000nt tiles over the genome and compute the number of reads and average GC content over all reads that fall within each tile using an overlap join and merging endpoints.

```
bins <- locations %>%
  tile_ranges(width = 10000L)


alignments_summary <- bins %>%
  join_overlap_inner(alignments) %>%
  disjoin_ranges(n = n(), avg_gc = mean(score))
alignments_summary
```

```
#> GRanges object with 286030 ranges and 2 metadata columns:
#>            seqnames            ranges strand |         n    avg_gc
#>               <Rle>         <IRanges>  <Rle> | <integer> <numeric>
#>        [1]    chr10       49999-59997      * |        88  0.369019
#>        [2]    chr10       59998-69997      * |        65  0.434211
#>        [3]    chr10       69998-79996      * |        56  0.386513
#>        [4]    chr10       79997-89996      * |        71  0.512973
#>        [5]    chr10       89997-99996      * |        64  0.387747
#>        ...      ...               ...    ... .       ...       ...
#>   [286026]     chrY 57722961-57732958      * |        36  0.468202
#>   [286027]     chrY 57732959-57742957      * |        38  0.469529
#>   [286028]     chrY 57742958-57752956      * |        38  0.542936
#>   [286029]     chrY 57752957-57762955      * |        42  0.510652
#>   [286030]     chrY 57762956-57772954      * |       504  0.526942
#>   -------
#>   seqinfo: 24 sequences from an unspecified genome; no seqlengths
```

Quality Control Metrics

We have created a *GRanges* object from genotyping performed on the H1 cell line, consisting of approximately two million single nucleotide polymorphisms (SNPs) and short insertion/deletions (indels). The *GRanges* object consists of 7 columns, relating to the alleles of a SNP or indel, the B-allele frequency, log relative intensity of the probes, GC content score over a probe, and the name of the probe. We can use this information to compute the

transition-transversion ratio, a quality control metric, within each chromosome in *GRanges* object.

First we filter out the indels and mitochondrial variants. Then we create a logical vector corresponding to whether there is a transition event.

```
h1_snp_array <- h1_snp_array %>%
  filter(!(ref %in% c("I", "D")), seqnames != "M") %>%
  mutate(transition = (ref %in% c("A", "G") & alt %in% c("G","A"))|
                      (ref %in% c("C","T") & alt %in% c("T", "C")))
```

We then compute the transition-transversion ratio over each chromosome using `group_by()` in combination with `summarize()` (Figure 2.5).

```
ti_tv_results <- h1_snp_array %>%
  group_by(seqnames) %>%
  summarize(n_snps = n(),
            ti_tv = sum(transition) / sum(!transition))
ti_tv_results
```

```
#> DataFrame with 24 rows and 3 columns
#>     seqnames    n_snps     ti_tv
#>        <Rle> <integer> <numeric>
#> 1          Y      2226   1.43812
#> 2          6    154246   3.32013
#> 3         13     83736   3.40669
#> 4         10    120035   3.49401
#> 5          4    153243   3.29529
#> ...      ...       ...       ...
#> 20        16     77538   3.19828
#> 21        12    113208   3.47852
#> 22        20     57073   3.71210
```

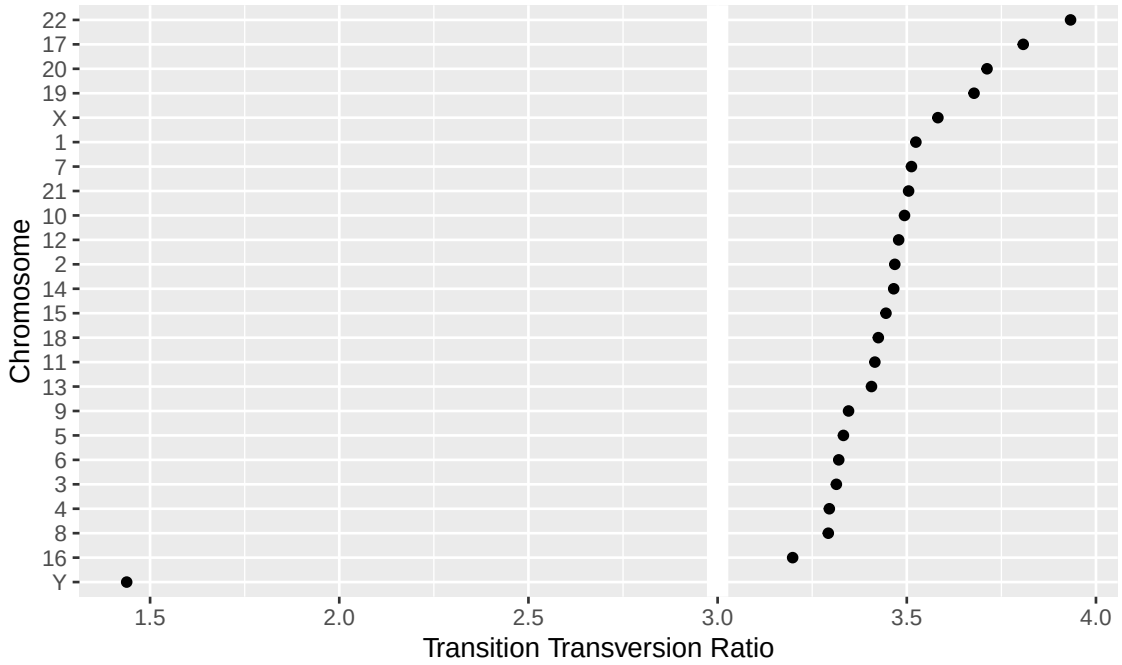


Figure 2.5: *The final result of computing quality control metrics over the SNP array data with* ***plyranges****, displayed as a dot plot. Chromosomes are ordered by their estimated transition-transversion ratio. A white reference line is drawn at the expected ratio for a human exome."*

```
#> 23        21     32349   3.50480
#> 24         X     55495   3.58220
```

2.3 Discussion

The design of **plyranges** adheres to well understood principles of language and API design: cognitive consistency, cohesion, endomorphism and expressiveness (Green and Petre, 1996). To varying degrees, these principles also underlie the design of **dplyr** and the Bioconductor infrastructure.

We have aimed for **plyranges** to have a simple and direct mapping to the user's cognitive model, i.e., how the user thinks about the data. This requires careful selection of the level of abstraction so that the user can express workflows in the language of genomics. This motivates the adoption of the tidy *GRanges* object as our central data structure. The basic *data.frame* and **dplyr** *tibble* lack any notion of genomic ranges and so could not easily support our genomic grammar, with its specific verbs for range-oriented data

manipulation. Another example of cognitive consistency is how **plyranges** is insensitive to direction/strand by default when, e.g., detecting overlaps. **GenomicRanges** has the opposite behavior. We believe that defaulting to purely spatial overlap is most intuitive to most users.

To further enable cognitive consistency, **plyranges** functions are cohesive. A function is defined to be cohesive if it performs a singular task without producing any side-effects. Singular tasks can always be broken down further at lower levels of abstraction. For example, to resize a range, the user needs to specify which position (start, end, midpoint) should be invariant over the transformation. The `resize()` function from the **GenomicRanges** package has a `fix` argument that sets the anchor, so calling `resize()` coalesces anchoring and width modification. The coupling at the function call level is justified since the effect of setting the width depends on the anchor. However, **plyranges** increases cohesion and decouples the anchoring into its own function call.

Increasing cohesion simplifies the interface to each operation, makes the meaning of arguments more intuitive, and relies on function names as the primary means of expression, instead of a more complex mixture of function and argument names. This results in the user being able to conceptualize the **plyranges** DSL as a flat catalog of functions, without having to descend further into documentation to understand a function's arguments. A flat function catalog also enhances API discoverability, particularly through auto-completion in integrated developer environments (IDEs). One downside of pushing cohesion to this extreme is that function calls become coupled, and care is necessary to treat them as a group when modifying code.

Like **dplyr**, **plyranges** verbs are functional: they are free of side effects and are generally endomorphic, meaning that when the input is a *GRanges* object they return a *GRanges* object. This enables chaining of verbs through syntax like the forward pipe operator from the **magrittr** package. This syntax has a direct cognitive mapping to natural language and the intuitive notion of pipelines. The low-level object-oriented APIs of Bioconductor tend to manipulate data via sub-replacement functions, like `start(gr) <- x`. These ultimately produce the side effect of replacing a symbol mapping in the current environment and thus are not amenable to so-called fluent syntax.

Expressiveness relates to the information content in code: the programmer should be able to clarify intent without unnecessary verbosity. For example, our overlap-based join operations are more concise than the multiple steps necessary to achieve the same effect in the original **GenomicRanges** API. In other cases, the **plyranges** API increases verbosity for the sake of clarity and cohesion. Explicitly calling `anchor()` can require more typing, but the code is easier to comprehend. Another example is the set of routines for importing genomic annotations, including `read_gff()`, `read_bed()`, and `read_bam()`. Compared to the generic `import()` in **rtracklayer**, the explicit format-based naming in **plyranges** clarifies intent and the type of data being returned. Similarly, every **plyranges** function that computes with strand information indicates its intentions by including suffixes such as *directed*, *upstream* or *downstream* in its name, otherwise strand is ignored. The **GenomicRanges** API does not make this distinction explicit in its function naming, instead relying on a parameter that defaults to strand sensitivity, an arguably confusing behavior.

The implementation of **plyranges** is built on top of Bioconductor infrastructure, meaning most functions are constructed by composing generic functions from core Bioconductor packages. As a result, any Bioconductor packages that uses data structures that inherit from *GRanges* will be able to use **plyranges** for free. Another consequence of building on top of Bioconductor generics is that the speed and memory usage of **plyranges** functions are similar to the highly optimized methods implemented in Bioconductor for *GRanges* objects.

A caveat to constructing a compatible interface with **dplyr** is that **plyranges** makes extensive use of non-standard evaluation in R via the **rlang** package (Henry and Wickham, 2017). Simply, this means that computations are evaluated in the context of the *GRanges* objects. Both **dplyr** and **plyranges** are based on the **rlang** language, because it allows for more expressive code that is free of repeated references to the container. Implicitly referencing the container is particularly convenient when programming interactively. Consequently, when programming with **plyranges**, a user needs to generally understand the **rlang** language and how to adapt their code accordingly. Users familiar with the **tidyverse** should already have such knowledge.

2.4 Conclusion

We have shown how to create expressive and reproducible genomic workflows using the **plyranges** DSL. By realising that the *GRanges* data model is tidy we have highlighted how to implement a grammar for performing genomic arithmetic, aggregation, restriction and merging. Our examples show that **plyranges** code is succinct, human readable and can take advantage of the interoperability provided by the Bioconductor ecosystem and the R language.

We also note that the grammar elements and design principles we have described are programming language agnostic and could be easily be implemented in another language where genomic information could be represented as a tabular data structure. We chose R because it is what we are familiar with and because the aforementioned Bioconductor packages have implemented the *GRanges* data structure.

We aim to continue developing the **plyranges** package and to extend it for use with more complex data structures, such as the *SummarizedExperiment* class, the core Bioconductor data structure for representing experimental results (e.g., counts) from multiple sample experiments in conjunction with feature and sample metadata. Although, the *SummarizedExperiment* is not strictly tidy, it does consist of three tidy data structures that are related by feature and sample identifiers. Therefore, the grammar and design of the **plyranges** DSL is naturally extensible to the *SummarizedExperiment*.

As the **plyranges** interface encourages tidy data practices, it integrates well with the grammar of graphics (Wickham, 2016). To achieve responsive performance, interactive graphics rely on lazy data access and computing patterns, so the deferred mechanisms within **plyranges** should help support interactive genomics applications.

2.5 Availability of Data and Materials

The BigWig file for the H3K27Me3 primary T CD8+ memory cells from peripheral blood ChIP-seq data from the Human Roadmap Epigenomics project
was downloaded from the **AnnotationHub** package (2.13.1) under accession AH33458

(Morgan, 2017; Roadmap Epigenomics Consortium et al., 2015). The BAM file corresponding to the H1 cell line ChIP-seq data is available at NCBI GEO under accession GSM433167 (Barrett et al., 2013; Roadmap Epigenomics Consortium et al., 2015). The SNP array data for the H1 cell line data is available at NCBI GEO under accession GPL18952 (Roadmap Epigenomics Consortium et al., 2015).

The **plyranges** package is open source under an Artistic 2.0 license (Lee, Lawrence, and Cook, 2018). The software can be obtained via the Bioconductor project website `https://bioconductor.org` or accessed via Github `https://github.com/sa-lee/plyranges`.

Acknowledgements

We would like to thank Dr Matthew Ritchie at the Walter and Eliza Hall Institute and Dr Paul Harrison for their feedback on earlier drafts of this work. We would also like to thank Lori Shepherd and Hèrve Pages for the code review they performed and users who have submitted feedback and pull requests.

Chapter 3

Fluent genomics with plyranges and tximeta

We construct a simple workflow for fluent genomics data analysis using the R/Bioconductor ecosystem. This involves three core steps: import the data into an appropriate abstraction, model the data with respect to the biological questions of interest, and integrate the results with respect to their underlying genomic coordinates. Here we show how to implement these steps to integrate published RNA-seq and ATAC-seq experiments on macrophage cell lines. Using **tximeta**, we import RNA-seq transcript quantifications into an analysis-ready data structure, called the *SummarizedExperiment*, that contains the ranges of the reference transcripts and metadata on their provenance. Using *SummarizedExperiments* to represent the ATAC-seq and RNA-seq data, we model differentially accessible (DA) chromatin peaks and differentially expressed (DE) genes with existing Bioconductor packages. Using **plyranges** we then integrate the results to see if there is an enrichment of DA peaks near DE genes by finding overlaps and aggregating over log-fold change thresholds. The combination of these packages and their integration with the Bioconductor ecosystem provide a coherent framework for analysts to iteratively and reproducibly explore their biological data.

3.1 Introduction

In this workflow, we examine a subset of the RNA-seq and ATAC-seq data from Alasoo et al. (2018), a study that involved treatment of macrophage cell lines from a number of human donors with interferon gamma (IFNg), Salmonella infection, or both treatments combined. Alasoo et al. (2018) examined gene expression and chromatin accessibility in a subset of 86 successfully differentiated induced pluripotent stem cells (iPSC) lines, and compared baseline and response with respect to chromatin accessibility and gene expression at specific quantitative trait loci (QTL). The authors found that many of the stimulus-specific expression QTL were already detectable as chromatin QTL in naive cells, and further hypothesize about the nature and role of transcription factors implicated in the response to stimulus.

We will perform a much simpler analysis than the one found in Alasoo et al. (2018), using their publicly available RNA-seq and ATAC-seq data (ignoring the genotypes). We will examine the effect of IFNg stimulation on gene expression and chromatin accessibility, and look to see if there is an enrichment of differentially accessible (DA) ATAC-seq peaks in the vicinity of differentially expressed (DE) genes. This is plausible, as the transcriptomic response to IFNg stimulation may be mediated through binding of regulatory proteins to accessible regions, and this binding may increase the accessibility of those regions such that it can be detected by ATAC-seq.

Throughout the workflow (Figure 3.1), we will use existing Bioconductor infrastructure to understand these datasets. In particular, we will emphasize the use of the Bioconductor packages **plyranges** and **tximeta** and the *SummarizedExperiment* class (Figure 3.2. The **plyranges** package fluently transforms data tied to genomic ranges using operations like shifting, window construction, overlap detection, etc. It is described by Lee, Cook, and Lawrence (2019) and leverages underlying core Bioconductor infrastructure (Lawrence et al., 2013b; Huber et al., 2015b) and the **tidyverse** design principles Wickham et al. (2019).

(ref:se): A *SummarizedExperiment* orients bulk assay measurements as matrices where rows correspond to features, and columns correspond to samples. The rows have their own accessor functions which contains additional data about the feature of interest, in

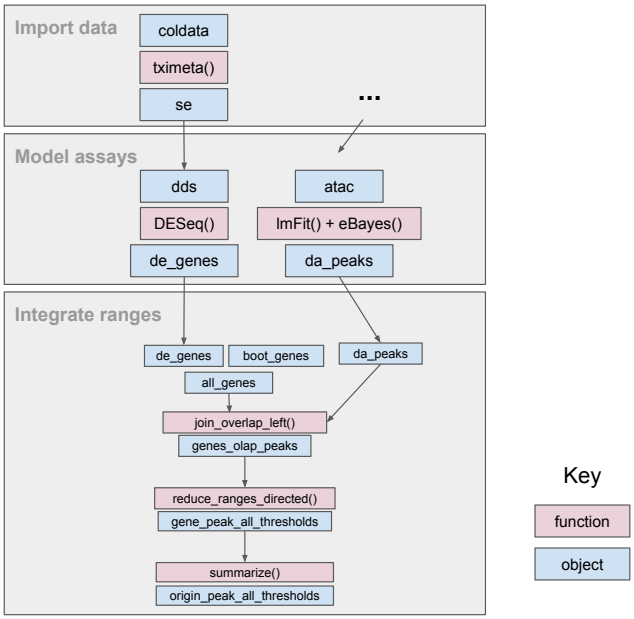


Figure 3.1: *An overview of the fluent genomics workflow. First, we* import *data as a* SummarizedExperiment *object, which enables interoperability with downstream analysis packages. Then we* model *our assay data, using the existing Bioconductor packages* ***DESeq2*** *and* ***limma****. We take the results of our models for each assay with respect to their genomic coordinates, and* integrate *them. First, we compute the overlap between the results of each assay, then aggregate over the combined genomic regions, and finally summarize to compare enrichment for differentially expressed genes to non differentially expressed genes. The final output can be used for downstream visualization or further transformation.*

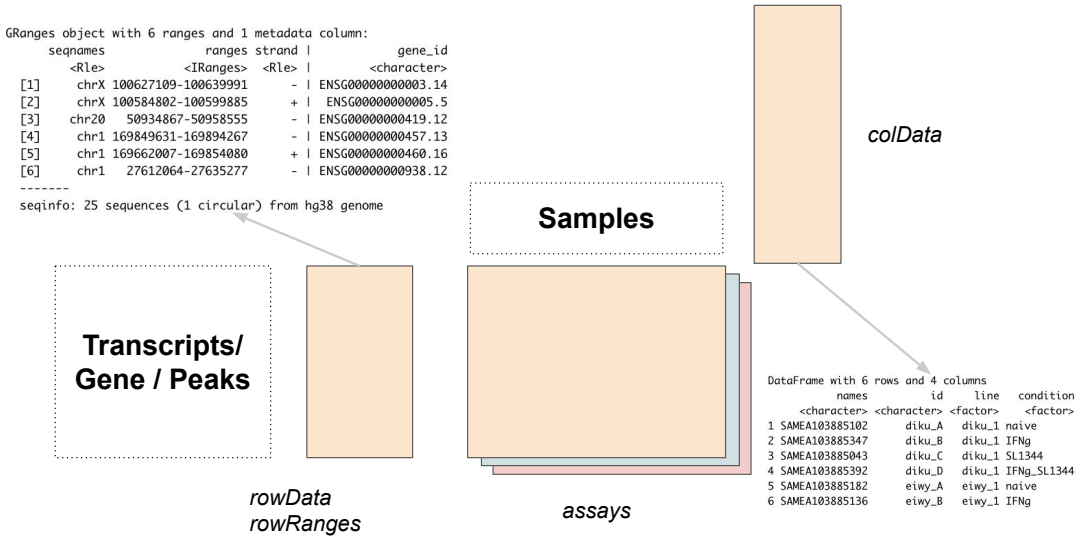


Figure 3.2: *(ref:se)*

this `rowRanges()` returns a *GRanges* where rows are the genomic coordinates for each gene measured in the assay, while `colData()` returns a *DataFrame* where rows contain information about the samples.

The **tximeta** package described by Love et al. (2019) is used to read RNA-seq quantification data into R/Bioconductor, such that the transcript ranges and their provenance are automatically attached to the object containing expression values and differential expression results.

3.1.1 Experimental Data

The data used in this workflow is available from two packages: the **macrophage** Bioconductor **ExperimentData** package and from the workflow package **fluentGenomics** (Lee and Love, n.d.).

The **macrophage** package contains RNA-seq quantification from 24 RNA-seq samples, a subset of the RNA-seq samples generated and analyzed by Alasoo et al. (2018). The paired-end reads were quantified using **Salmon** (Patro et al., 2017), using the Gencode 29 human reference transcripts (Frankish, GENCODE-consoritum, and Flicek, 2018). For more details on quantification, and the exact code used, consult the vignette of the **macrophage** package. The package also contains the **Snakemake** file that was used to distribute the **Salmon** quantification jobs on a cluster (Köster and Rahmann, 2012).

The **fluentGenomics** package contains functionality to download and generate a cached *SummarizedExperiment* object from the normalized ATAC-seq data provided by Alasoo and Gaffney (2017). This object contains all 145 ATAC-seq samples across all experimental conditions as analyzed by Alasoo et al. (2018). The data can be also be downloaded directly from the Zenodo deposition.

The following code loads the path to the cached data file, or if it is not present, will create the cache and generate a *SummarizedExperiment* using the the **BiocFileCache** package (Shepherd and Morgan, 2019).

```
library(fluentGenomics)
path_to_se <- cache_atac_se()
```

We can then read the cached file and assign it to an object called `atac`.

```
atac <- readRDS(path_to_se)
```

A precise description of how we obtained this *SummarizedExperiment* object can be found in section 3.2.2.

3.2 Import Data as a *SummarizedExperiment*

3.2.1 Using tximeta to import RNA-seq quantification data

First, we specify a directory `path`, where the quantification files are stored. You could simply specify this directory with:

```
path <- "/path/to/quant/files"
```

where the path is relative to your current R session. However, in this case we have distributed the files in the **macrophage** package. The relevant directory and associated files can be located using `system.file()`.

```
dir <- system.file("extdata", package="macrophage")
```

Information about the experiment is contained in the `coldata.csv` file. We leverage the **dplyr** and **readr** packages (as part of the **tidyverse**) to read this file into R (Wickham et al., 2019). We will see later that **plyranges** extends these packages to accommodate genomic ranges.

```
library(readr)
library(dplyr)
colfile <- file.path(dir, "coldata.csv")
```

```
coldata <- read_csv(colfile) %>%
  select(
    names,
    id = sample_id,
    line = line_id,
    condition = condition_name
  ) %>%
  mutate(
    files = file.path(dir, "quants", names, "quant.sf.gz"),
    line = factor(line),
    condition = relevel(factor(condition), "naive")
  )
glimpse(coldata)
```

```
#> Rows: 24
#> Columns: 5
#> $ names     <chr> "SAMEA103885102", "SAMEA103885347", "SAMEA103885043", "SA...
#> $ id        <chr> "diku_A", "diku_B", "diku_C", "diku_D", "eiwy_A", "eiwy_B...
#> $ line      <fct> diku_1, diku_1, diku_1, diku_1, eiwy_1, eiwy_1, eiwy_1, e...
#> $ condition <fct> naive, IFNg, SL1344, IFNg_SL1344, naive, IFNg, SL1344, IF...
#> $ files     <chr> "/Users/slee0046/thesis/renv/library/R-4.0/x86_64-apple-d...
```

After we have read the `coldata.csv` file, we select relevant columns from this table, create a new column called `files`, and transform the existing `line` and `condition` columns into factors. In the case of `condition`, we specify the "naive" cell line as the reference level. The `files` column points to the quantifications for each observation – these files have been gzipped, but would typically not have the 'gz' ending if used from **Salmon** directly. One other thing to note is the use of the pipe operator, `%>%`, which can be read as "then", i.e. first read the data, *then* select columns, *then* mutate them.

Now we have a table summarizing the experimental design and the locations of the quantifications. The following lines of code do a lot of work for the analyst: importing the

RNA-seq quantification (dropping *inferential replicates* in this case), locating the relevant reference transcriptome, attaching the transcript ranges to the data, and fetching genome information. Inferential replicates are especially useful for performing transcript-level analysis, but here we will use a point estimate for the per-gene counts and perform gene-level analysis. The result is a *SummarizedExperiment* object.

```
library(SummarizedExperiment)
library(tximeta)
se <- tximeta(coldata, dropInfReps=TRUE)
se
```

```
#> class: RangedSummarizedExperiment
#> dim: 205870 24
#> metadata(6): tximetaInfo quantInfo ... txomeInfo txdbInfo
#> assays(3): counts abundance length
#> rownames(205870): ENST00000456328.2 ENST00000450305.2 ...
#>   ENST00000387460.2 ENST00000387461.2
#> rowData names(3): tx_id gene_id tx_name
#> colnames(24): SAMEA103885102 SAMEA103885347 ... SAMEA103885308
#>   SAMEA103884949
#> colData names(4): names id line condition
```

On a machine with a working internet connection, the above command works without any extra steps, as the **tximeta** function obtains any necessary metadata via FTP, unless it is already cached locally. The **tximeta** package can also be used without an internet connection, in this case the linked transcriptome can be created directly from a **Salmon** index and gtf.

```
makeLinkedTxome(
  indexDir=file.path(path, "gencode.v29_salmon_0.12.0"),
  source="Gencode",
  organism="Homo sapiens",
```

```
  release="29",
  genome="GRCh38",
  fasta="gencode.v29.transcripts.fa.gz", # ftp link to fasta file
  gtf=file.path(path, "gencode.v29.annotation.gtf.gz"), # local version
  write=FALSE
)
```

Because **tximeta** knows the correct reference transcriptome, we can ask **tximeta** to summarize the transcript-level data to the gene level using the methods of Soneson, Love, and Robinson (2015).

```
gse <- summarizeToGene(se)
```

One final note is that the *start* of positive strand genes and the *end* of negative strand genes is now dictated by the genomic extent of the isoforms of the gene (so the *start* and *end* of the reduced *GRanges*). Another alternative would be to either operate on transcript abundance, and perform differential analysis on transcript, and so avoid defining the transcription start site (TSS) of a set of isoforms, or to use gene-level summarized expression but to pick the most representative TSS based on isoform expression.

3.2.2 Importing ATAC-seq data as a *SummarizedExperiment* object

The *SummarizedExperiment* object containing ATAC-seq peaks can be created from the following tab-delimited files from Alasoo and Gaffney (2017):

- The sample metadata: `ATAC_sample_metadata.txt.gz` (<1M)
- The matrix of normalized read counts: `ATAC_cqn_matrix.txt.gz` (109M)
- The annotated peaks: `ATAC_peak_metadata.txt.gz` (5.6M)

To begin, we read in the sample metadata, following similar steps to those we used to generate the `coldata` table for the RNA-seq experiment:

```
atac_coldata <- read_tsv("ATAC_sample_metadata.txt.gz") %>%
  select(
    sample_id,
    donor,
    condition = condition_name
  ) %>%
  mutate(condition = relevel(factor(condition), "naive"))
```

The ATAC-seq counts have already been normalized with **cqn** (Hansen, Irizarry, and Wu, 2012) and $\log_2$ transformed. Loading the **cqn**-normalized matrix of $\log_2$ transformed read counts takes ~30 seconds and loads an object of ~370 Mb. We set the column names so that the first column contains the rownames of the matrix, and the remaining columns are the sample identities from the `atac_coldata` object.

```
atac_mat <- read_tsv(
  "ATAC_cqn_matrix.txt.gz",
  skip = 1,
  col_names = c("rownames", atac_coldata[["sample_id"]])
 )
rownames <- atac_mat[["rownames"]]
atac_mat <- as.matrix(atac_mat[,-1])
rownames(atac_mat) <- rownames
```

We read in the peak metadata (locations in the genome), and convert it to a *GRanges* object. The `as_granges()` function automatically converts the *data.frame* into a *GRanges* object. From that result, we extract the peak_id column and set the genome information to the build "GRCh38". We know this from the Zenodo entry.

```
library(plyranges)
peaks_df <- read_tsv(
  "ATAC_peak_metadata.txt.gz",
  col_types = c("cidciicdc")
```

```
  )


peaks_gr <- peaks_df %>%
  as_granges(seqnames = chr) %>%
  select(peak_id=gene_id) %>%
  set_genome_info(genome = "GRCh38")
```

Finally, we construct a *SummarizedExperiment* object. We place the matrix into the assays slot as a named list, the annotated peaks into the row-wise ranges slot, and the sample metadata into the column-wise data slot:

```
atac <- SummarizedExperiment(
  assays = list(cqndata=atac_mat),
  rowRanges = peaks_gr,
  colData = atac_coldata
)
```

3.3 Model assays

3.3.1 RNA-seq differential gene expression analysis

We can easily run a differential expression analysis with **DESeq2** using the following code chunks (Love, Huber, and Anders, 2014). The design formula indicates that we want to control for the donor baselines (*line*) and test for differences in gene expression on the condition. For a more comprehensive discussion of DE workflows in Bioconductor see Love et al. (2016) and Law et al. (2018).

```
library(DESeq2)
dds <- DESeqDataSet(gse, ~line + condition)
# filter out lowly expressed genes
# at least 10 counts in at least 6 samples
```

```
keep <- rowSums(counts(dds) >= 10) >= 6
dds <- dds[keep,]
```

The model is fit with the following line of code:

```
dds <- DESeq(dds)
```

Below we set the contrast on the condition variable, indicating we are estimating the $\log_2$ fold change (LFC) of IFNg stimulated cell lines against naive cell lines. We are interested in LFC greater than 1 at a nominal false discovery rate (FDR) of 1%.

```
res <- results(dds,
               contrast=c("condition","IFNg","naive"),
               lfcThreshold=1, alpha=0.01)
```

The `results()` function extracts a summary of the DE analysis: in this case for each gene we have the LFC comparing the two cell lines, the Wald test statistic of LFC from the fitted negative binomial GLM, and associated p-value and corrected p-value accounting for the FDR. To see the results of the expression analysis, we can generate a summary table and a mean-abundance (MA) plot (Dudoit et al., 2002):

```
summary(res)
```

```
#>
#> out of 17806 with nonzero total read count
#> adjusted p-value < 0.01
#> LFC > 1.00 (up)    : 502, 2.8%
#> LFC < -1.00 (down) : 247, 1.4%
#> outliers [1]       : 0, 0%
#> low counts [2]     : 0, 0%
#> (mean count < 3)
#> [1] see 'cooksCutoff' argument of ?results
#> [2] see 'independentFiltering' argument of ?results
```

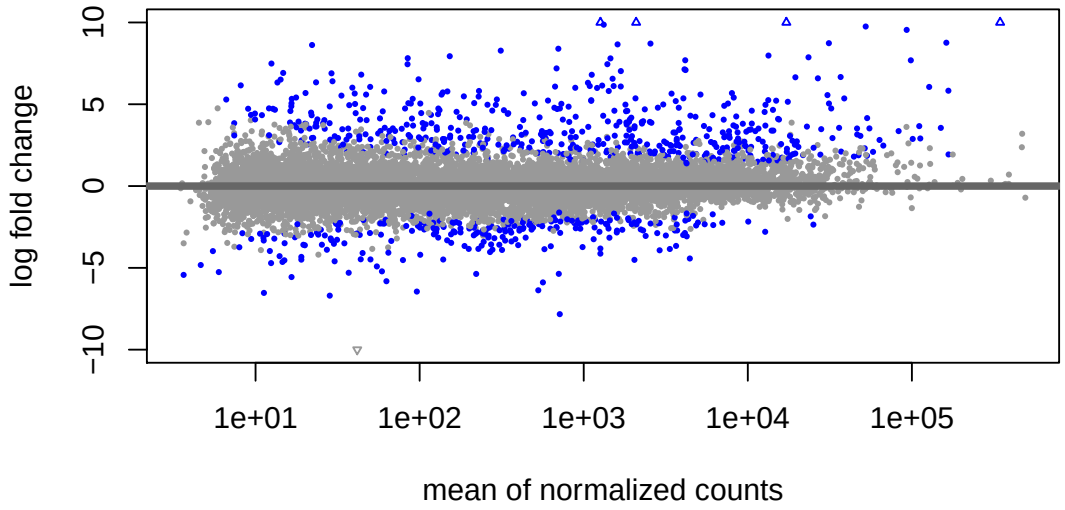


Figure 3.3: *Visualization of* `DESeq2` *results as an "MA plot". Genes that have an adjusted* p-value *below 0.01 are colored red.In this case the LFC between conditions is shown on the y-axis, while the average normalised gene counts across all conditions are shown on the x-axis. The assumption of an RNA-seq analysis that most genes are not DE, so most genes are scattered about zero on the y-axis while genes that have evidence of DE are far from the zero baseline. The x-axis gives a sense of the total expression of the gene.*

```
DESeq2::plotMA(res, ylim=c(-10,10))
```

We now output the results as a *GRanges* object, and due to the conventions of **plyranges**, we construct a new column called *gene_id* from the row names of the results. Each row now contains the genomic region (*seqnames*, *start*, *end*, *strand*) along with corresponding metadata columns (the *gene_id* and the results of the test). Note that **tximeta** has correctly identified the reference genome as "hg38", and this has also been added to the **GRanges** along the results columns. This kind of book-keeping is vital once overlap operations are performed to ensure that **plyranges** is not comparing across incompatible genomes.

```
suppressPackageStartupMessages(library(plyranges))
de_genes <- results(dds,
                    contrast=c("condition","IFNg","naive"),
                    lfcThreshold=1,
```

```
                    format="GRanges") %>%
  names_to_column("gene_id")
de_genes
```

```
#> GRanges object with 17806 ranges and 7 metadata columns:
#>           seqnames              ranges strand |            gene_id   baseMean
#>              <Rle>           <IRanges>  <Rle> |        <character>  <numeric>
#>       [1]     chrX 100627109-100639991      - | ENSG00000000003.14    171.571
#>       [2]    chr20   50934867-50958555      - | ENSG00000000419.12    967.751
#>       [3]     chr1 169849631-169894267      - | ENSG00000000457.13    682.433
#>       [4]     chr1 169662007-169854080      + | ENSG00000000460.16    262.963
#>       [5]     chr1   27612064-27635277      - | ENSG00000000938.12   2660.102
#>       ...      ...                 ...    ... .                ...        ...
#>   [17802]    chr10   84167228-84172093      - |  ENSG00000285972.1   10.04746
#>   [17803]     chr6   63572012-63583587      + |  ENSG00000285976.1 4586.34617
#>   [17804]    chr16   57177349-57181390      + |  ENSG00000285979.1   14.29653
#>   [17805]     chr8 103398658-103501895      - |  ENSG00000285982.1   27.76296
#>   [17806]    chr10   12563151-12567351      + |  ENSG00000285994.1    6.60409
#>           log2FoldChange     lfcSE      stat    pvalue      padj
#>                <numeric> <numeric> <numeric> <numeric> <numeric>
#>       [1]     -0.2822450 0.3005710   0.00000 1.0000000  1.000000
#>       [2]      0.0391223 0.0859708   0.00000 1.0000000  1.000000
#>       [3]      1.2846179 0.1969067   1.44545 0.1483329  1.000000
#>       [4]     -1.4718762 0.2186916  -2.15772 0.0309493  0.409728
#>       [5]      0.6754781 0.2360530   0.00000 1.0000000  1.000000
#>       ...            ...       ...       ...       ...       ...
#>   [17802]      0.5484518  0.444319         0         1         1
#>   [17803]     -0.0339296  0.188005         0         1         1
#>   [17804]      0.3123477  0.522700         0         1         1
#>   [17805]      0.9945187  1.582373         0         1         1
#>   [17806]      0.2539975  0.595751         0         1         1
```

```
#>   -------
#>   seqinfo: 25 sequences (1 circular) from hg38 genome
```

From this, we can restrict the results to those that meet our FDR threshold and select (and rename) the metadata columns we are interested in:

```
de_genes <- de_genes %>%
  filter(padj < 0.01) %>%
  select(
    gene_id,
    de_log2FC = log2FoldChange,
    de_padj = padj
  )
```

We now wish to extract genes for which there is evidence that the LFC is *not* large. We perform this test by specifying an LFC threshold and an alternative hypothesis (`altHypothesis`) that the LFC is less than the threshold in absolute value. In this case, the p-values are taken as maximum of the upper and lower Wald tests under the hypothesis absolute value of the estimated LFC is lower than the threshold. To visualize the result of this test, you can run `results` without `format="GRanges"`, and pass this object to `plotMA()` as before. We label these genes as `other_genes` and later as "non-DE genes", for comparison with our `de_genes` set.

```
other_genes <- results(dds,
                       contrast=c("condition","IFNg","naive"),
                       lfcThreshold=1,
                       altHypothesis="lessAbs",
                       format="GRanges") %>%
  filter(padj < 0.01) %>%
  names_to_column("gene_id") %>%
  select(
    gene_id,
```

```
    de_log2FC = log2FoldChange,
    de_padj = padj
  )
```

3.3.2 ATAC-seq peak differential abundance analysis

The following section describes the process we have used for generating a *GRanges* object of differential peaks from the ATAC-seq data in Alasoo et al. (2018). The code chunks for the remainder of this section are optional.

For assessing differential accessibility, we run **limma** (Smyth, 2004), and generate the a summary of LFCs and adjusted p-values for the peaks:

```
library(limma)
design <- model.matrix(~donor + condition, colData(atac))
fit <- lmFit(assay(atac), design)
fit <- eBayes(fit)
idx <- which(colnames(fit$coefficients) == "conditionIFNg")
tt <- topTable(fit, coef=idx, sort.by="none", n=nrow(atac))
```

We now take the `rowRanges()` of the *SummarizedExperiment* and attach the LFCs and adjusted p-values from **limma**, so that we can consider the overlap with differential expression. Note that we set the genome build to "hg38" and restyle the chromosome information to use the "UCSC" style (e.g. "chr1", "chr2", etc.). Again, we know the genome build from the Zenodo entry for the ATAC-seq data.

```
atac_peaks <- rowRanges(atac) %>%
  remove_names() %>%
  mutate(
    da_log2FC = tt$logFC,
    da_padj = tt$adj.P.Val
  ) %>%
  set_genome_info(genome = "hg38")
```

```
seqlevelsStyle(atac_peaks) <- "UCSC"
```

The final *GRanges* object containing the DA peaks is included in the **fluentGenomics** and can be loaded as follows:

```
library(fluentGenomics)
peaks
```

```
#> GRanges object with 296220 ranges and 3 metadata columns:
#>             seqnames              ranges strand |          peak_id da_log2FC
#>                <Rle>           <IRanges>  <Rle> |      <character> <numeric>
#>        [1]     chr1          9979-10668      * |      ATAC_peak_1  0.266185
#>        [2]     chr1         10939-11473      * |      ATAC_peak_2  0.322177
#>        [3]     chr1         15505-15729      * |      ATAC_peak_3 -0.574160
#>        [4]     chr1         21148-21481      * |      ATAC_peak_4 -1.147066
#>        [5]     chr1         21864-22067      * |      ATAC_peak_5 -0.896143
#>        ...      ...                 ...    ... .              ...       ...
#>   [296216]     chrX 155896572-155896835      * | ATAC_peak_296216 -0.834629
#>   [296217]     chrX 155958507-155958646      * | ATAC_peak_296217 -0.147537
#>   [296218]     chrX 156016760-156016975      * | ATAC_peak_296218 -0.609732
#>   [296219]     chrX 156028551-156029422      * | ATAC_peak_296219 -0.347678
#>   [296220]     chrX 156030135-156030785      * | ATAC_peak_296220  0.492442
#>                da_padj
#>              <numeric>
#>        [1] 9.10673e-05
#>        [2] 2.03435e-05
#>        [3] 3.41708e-08
#>        [4] 8.22299e-26
#>        [5] 4.79453e-11
#>        ...         ...
#>   [296216] 1.33546e-11
```

```
#>   [296217] 3.13015e-01
#>   [296218] 3.62339e-09
#>   [296219] 6.94823e-06
#>   [296220] 7.07664e-13
#>   -------
#>   seqinfo: 23 sequences from hg38 genome; no seqlengths
```

3.4 Integrate ranges

3.4.1 Finding overlaps with plyranges

We have already used **plyranges** a number of times above, to `filter()`, `mutate()`, and `select()` on *GRanges* objects, as well as ensuring the correct genome annotation and style has been used. The **plyranges** package provides a grammar for performing transformations of genomic data (Lee, Cook, and Lawrence, 2019). Computations resulting from compositions of **plyranges** "verbs" are performed using underlying, highly optimized range operations in the **GenomicRanges** package (Lawrence et al., 2013b).

For the overlap analysis, we filter the annotated peaks to have a nominal FDR bound of 1%.

```
da_peaks <- peaks %>%
  filter(da_padj < 0.01)
```

We now have *GRanges* objects that contain DE genes, genes without strong signal of DE, and DA peaks. We are ready to answer the question: is there an enrichment of DA ATAC-seq peaks in the vicinity of DE genes compared to genes without sufficient DE signal?

3.4.2 Down sampling non-differentially expressed genes

As **plyranges** is built on top of **dplyr**, it implements methods for many of its verbs for *GRanges* objects. Here we can use `slice()` to randomly sample the rows of the

`other_genes`. The `sample.int()` function will generate random samples of size equal to the number of DE-genes from the number of rows in `other_genes`:

```
size <- length(de_genes)
slice(other_genes, sample.int(n(), size))
```

```
#> GRanges object with 749 ranges and 3 metadata columns:
#>         seqnames              ranges strand |            gene_id  de_log2FC
#>            <Rle>           <IRanges>  <Rle> |        <character>  <numeric>
#>     [1]     chr4   77047158-77076005      - |  ENSG00000118816.9 -0.2235346
#>     [2]     chr4       932387-958656      + | ENSG00000127419.16 -0.0868589
#>     [3]     chr4      986997-1004506      + | ENSG00000127415.12 -0.1856082
#>     [4]    chr18   74250847-74292016      - | ENSG00000166347.18  0.3571972
#>     [5]     chr7   32916815-32943176      - | ENSG00000205763.13  0.0781232
#>     ...      ...                 ...    ... .                ...        ...
#>   [745]     chr5 157731197-157741448      + | ENSG00000113272.13   0.203247
#>   [746]     chr4       124480-202303      + |  ENSG00000250312.7  -0.195769
#>   [747]    chr17   59952240-59964761      - | ENSG00000189050.15   0.420200
#>   [748]     chr2 178480468-178516462      + | ENSG00000116095.10   0.209888
#>   [749]     chr8   38030341-38060365      + |  ENSG00000187840.4  -0.499126
#>             de_padj
#>           <numeric>
#>     [1] 1.01599e-28
#>     [2] 9.58269e-15
#>     [3] 1.66307e-09
#>     [4] 1.53096e-03
#>     [5] 8.05479e-07
#>     ...         ...
#>   [745] 3.11825e-06
#>   [746] 3.19089e-07
#>   [747] 1.74215e-04
```

```
#>   [748] 7.99140e-11
#>   [749] 8.53669e-03
#>   -------
#>   seqinfo: 25 sequences (1 circular) from hg38 genome
```

We can repeat this many times to create many samples via `replicate()`. We are subsampling gene sets without evidence of DE to have the same size as DE gene set, because for each set we want to how different the ATAC peaks around these different sets assuming the number of DE genes is fixed. The sampling of the non-DE genes is done without replacement so each replication produces a different set to compare. By replicating the subsampling multiple times, we minimize the variance on the enrichment statistics induced by the sampling process.

```
# set a seed for the results
set.seed(2019-08-02)
subsamp_genes <- replicate(10,
                           slice(other_genes, sample.int(n(), size)),
                           simplify = FALSE)
```

This creates a list of *GRanges* objects as a list, and we can bind these together using `bind_ranges()`. This function creates a new column called *resample* on the result that identifies each of the input *GRanges* objects:

```
subsamp_genes <- bind_ranges(subsamp_genes, .id = "resample")
```

Similarly, we can then combine the `subsamp_genes` *GRanges*, with the DE *GRanges* object. As the resample column was not present on the DE *GRanges* object, this is given a missing value which we recode to a 0 using `mutate()`

```
all_genes <- bind_ranges(
  de=de_genes,
  not_de = subsamp_genes,
```

```
  .id="origin"
) %>%
  mutate(
    origin = factor(origin, c("not_de", "de")),
    resample = ifelse(is.na(resample), 0L, as.integer(resample))
  )
all_genes
```

```
#> GRanges object with 8239 ranges and 5 metadata columns:
#>          seqnames              ranges strand |            gene_id de_log2FC
#>             <Rle>           <IRanges>  <Rle> |        <character> <numeric>
#>      [1]     chr1 196651878-196747504      + | ENSG00000000971.15   4.98711
#>      [2]     chr6   46129993-46146699      + |  ENSG00000001561.6   1.92722
#>      [3]     chr4   17577192-17607972      + | ENSG00000002549.12   2.93373
#>      [4]     chr7 150800403-150805120      + |  ENSG00000002933.8   3.16722
#>      [5]     chr4   15778275-15853230      + | ENSG00000004468.12   5.40894
#>      ...      ...                 ...    ... .                ...       ...
#>   [8235]    chr17   43527844-43579620      - | ENSG00000175832.12 -0.240918
#>   [8236]    chr17   18260534-18266552      + | ENSG00000177427.12 -0.166059
#>   [8237]    chr20   63895182-63936031      + | ENSG00000101152.10  0.250539
#>   [8238]     chr1   39081316-39487177      + | ENSG00000127603.25 -0.385054
#>   [8239]     chr8   41577187-41625001      + | ENSG00000158669.11  0.155922
#>              de_padj  resample   origin
#>            <numeric> <integer> <factor>
#>      [1] 1.37057e-13         0       de
#>      [2] 3.17478e-05         0       de
#>      [3] 2.01310e-11         0       de
#>      [4] 1.07360e-08         0       de
#>      [5] 4.82905e-18         0       de
#>      ...         ...       ...      ...
#>   [8235] 9.91611e-03        10   not_de
```

```
#>   [8236] 9.12051e-05        10   not_de
#>   [8237] 1.74085e-09        10   not_de
#>   [8238] 2.65539e-03        10   not_de
#>   [8239] 2.96375e-17        10   not_de
#>   -------
#>   seqinfo: 25 sequences (1 circular) from hg38 genome
```

3.4.3 Expanding genomic coordinates around the transcription start site

Now we would like to modify our gene ranges so they contain the 10 kilobases on either side of their TSS. There are many ways one could do this, but we prefer an approach via the anchoring methods in **plyranges**. Because there is a mutual dependence between the start, end, width, and strand of a *GRanges* object, we define anchors to fix one of *start* and *end*, while modifying the *width*. As an example, to extract just the TSS, we can anchor by the 5′ end of the range and modify the width of the range to equal 1.

```
all_genes <- all_genes %>%
  anchor_5p() %>%
  mutate(width = 1)
```

Anchoring by the 5′ end of a range will fix the *end* of negatively stranded ranges, and fix the *start* of positively stranded ranges.

We can then repeat the same pattern but this time using `anchor_center()` to tell **plyranges** that we are making the TSS the midpoint of a range that has total width of 20kb, or 10kb both upstream and downstream of the TSS.

```
all_genes <- all_genes %>%
  anchor_center() %>%
  mutate(width=2*1e4)
```

3.4.4 Use overlap joins to find relative enrichment

We are now ready to compute overlaps between RNA-seq genes (our DE set and resampled sets) and the ATAC-seq peaks. In **plyranges**, overlaps are defined as joins between two *GRanges* objects: a *left* and a *right GRanges* object. In an overlap join, a match is any range on the *left GRanges* that is overlapped by the *right GRanges*. One powerful aspect of the overlap joins is that the result maintains all (metadata) columns from each of the *left* and *right* ranges which makes downstream summaries easy to compute.

To combine the DE genes with the DA peaks, we perform a left overlap join. This returns to us the `all_genes` ranges (potentially with duplication), but with the metadata columns from those overlapping DA peaks. For any gene that has no overlaps, the DA peak columns will have `NA` values.

```
genes_olap_peaks <- all_genes %>%
  join_overlap_left(da_peaks)
genes_olap_peaks
```

```
#> GRanges object with 27766 ranges and 8 metadata columns:
#>           seqnames              ranges strand |            gene_id de_log2FC
#>              <Rle>           <IRanges>  <Rle> |        <character> <numeric>
#>       [1]     chr1 196641878-196661877      + | ENSG00000000971.15   4.98711
#>       [2]     chr6   46119993-46139992      + |  ENSG00000001561.6   1.92722
#>       [3]     chr4   17567192-17587191      + | ENSG00000002549.12   2.93373
#>       [4]     chr4   17567192-17587191      + | ENSG00000002549.12   2.93373
#>       [5]     chr4   17567192-17587191      + | ENSG00000002549.12   2.93373
#>       ...      ...                 ...    ... .                ...       ...
#>   [27762]     chr1   39071316-39091315      + | ENSG00000127603.25 -0.385054
#>   [27763]     chr1   39071316-39091315      + | ENSG00000127603.25 -0.385054
#>   [27764]     chr8   41567187-41587186      + | ENSG00000158669.11  0.155922
#>   [27765]     chr8   41567187-41587186      + | ENSG00000158669.11  0.155922
#>   [27766]     chr8   41567187-41587186      + | ENSG00000158669.11  0.155922
```

```
#>              de_padj  resample   origin          peak_id da_log2FC     da_padj
#>            <numeric> <integer> <factor>      <character> <numeric>   <numeric>
#>       [1] 1.37057e-13         0       de  ATAC_peak_21236 -0.546582 1.15274e-04
#>       [2] 3.17478e-05         0       de ATAC_peak_231183  1.453297 9.73225e-17
#>       [3] 2.01310e-11         0       de ATAC_peak_193578  0.222371 3.00939e-11
#>       [4] 2.01310e-11         0       de ATAC_peak_193579 -0.281615 7.99889e-05
#>       [5] 2.01310e-11         0       de ATAC_peak_193580  0.673705 7.60043e-15
#>       ...         ...       ...      ...              ...       ...         ...
#>   [27762] 2.65539e-03        10   not_de   ATAC_peak_5357 -1.058236 3.69052e-16
#>   [27763] 2.65539e-03        10   not_de   ATAC_peak_5358 -1.314112 6.44280e-26
#>   [27764] 2.96375e-17        10   not_de ATAC_peak_263396 -0.904080 8.19577e-13
#>   [27765] 2.96375e-17        10   not_de ATAC_peak_263397  0.364738 2.08835e-08
#>   [27766] 2.96375e-17        10   not_de ATAC_peak_263399  0.317387 1.20088e-08
#>   -------
#>   seqinfo: 25 sequences (1 circular) from hg38 genome
```

Now we can ask, how many DA peaks are near DE genes relative to "other" non-DE genes? A gene may appear more than once in `genes_olap_peaks`, because multiple peaks may overlap a single gene, or because we have re-sampled the same gene more than once, or a combination of these two cases.

For each gene (that is the combination of chromosome, the start, end, and strand), and the "origin" (DE vs not-DE) we can compute the distinct number of peaks for each gene and the maximum peak based on LFC. This is achieved via `reduce_ranges_directed()`, which allows an aggregation to result in a *GRanges* object via merging neighboring genomic regions. The use of the directed suffix indicates we are maintaining strand information. In this case, we are simply merging ranges (genes) via the groups we mentioned above. We also have to account for the number of resamples we have performed when counting if there are any peaks, to ensure we do not double count the same peak:

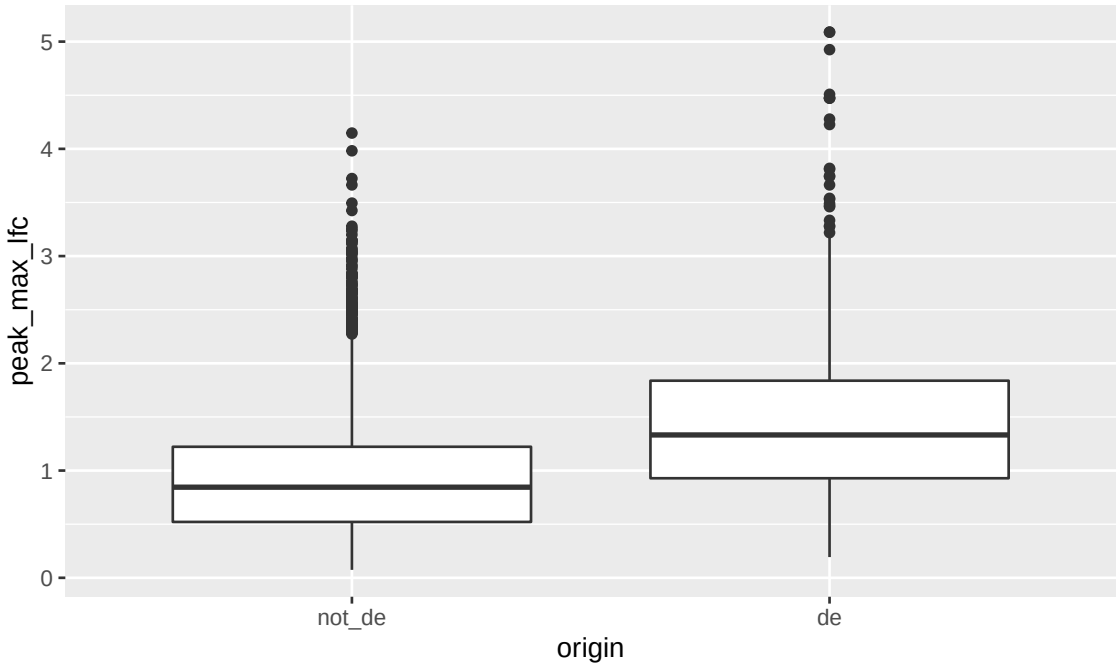


Figure 3.4: *A boxplot of maximum LFCs for DA peaks for DE genes compared to non-DE genes where genes have at least one DA peak.*

```
gene_peak_max_lfc <- genes_olap_peaks %>%
  group_by(gene_id, origin)  %>%
  reduce_ranges_directed(
    peak_count = sum(!is.na(da_padj)) / n_distinct(resample),
    peak_max_lfc = max(abs(da_log2FC))
  )
```

We can then filter genes if they have any peaks and compare the peak fold changes between non-DE and DE genes using a boxplot:

```
library(ggplot2)
gene_peak_max_lfc %>%
  filter(peak_count > 0) %>%
  as.data.frame() %>%
  ggplot(aes(origin, peak_max_lfc)) +
  geom_boxplot()
```

In general, the DE genes have larger maximum DA fold changes relative to the non-DE genes.

Next we examine how thresholds on the DA LFC modify the enrichment we observe of DA peaks near DE or non-DE genes. First, we want to know how the number of peaks within DE genes and non-DE genes change as we change threshold values on the peak LFC. As an example, we could compute this by arbitrarily chosen LFC thresholds of 1 or 2 as follows:

```
origin_peak_lfc <- genes_olap_peaks %>%
  group_by(origin) %>%
  summarize(
    peak_count = sum(!is.na(da_padj)) / n_distinct(resample),
    lfc1_peak_count =
      sum(abs(da_log2FC) > 1, na.rm=TRUE) / n_distinct(resample),
    lfc2_peak_count =
      sum(abs(da_log2FC) > 2, na.rm=TRUE) / n_distinct(resample)
  )
origin_peak_lfc
```

```
#> DataFrame with 2 rows and 4 columns
#>     origin peak_count lfc1_peak_count lfc2_peak_count
#>   <factor>  <numeric>       <numeric>       <numeric>
#> 1   not_de     2391.8           369.5            32.5
#> 2   de         3416.0          1097.0           234.0
```

Here we see that DE genes tend to have more DA peaks near them, and that the number of DA peaks decreases as we increase the DA LFC threshold (as expected). We now show how to compute the ratio of peak counts from DE compared to non-DE genes, so we can see how this ratio changes for various DA LFC thresholds.

For all variables except for the *origin* column we divide the first rows values by the second row, which will be the enrichment of peaks in DE genes compared to other genes. This

requires us to reshape the summary table from long form back to wide form using the **tidyr** package. First we pivot the results of the *peak_count* columns into name-value pairs, then pivot again to place values into the *origin* column. Then we create a new column with the relative enrichment:

```
library(tidyr)
origin_peak_lfc %>%
  as.data.frame() %>%
  pivot_longer(cols = -origin) %>%
  pivot_wider(names_from = origin, values_from = value) %>%
  mutate(enrichment = de / not_de)
```

```
#> # A tibble: 3 x 4
#>   name            not_de    de enrichment
#>   <chr>            <dbl> <dbl>      <dbl>
#> 1 peak_count      2392.   3416       1.43
#> 2 lfc1_peak_count  370.   1097       2.97
#> 3 lfc2_peak_count   32.5   234       7.2
```

The above table shows that relative enrichment increases for a larger LFC threshold.

Due to the one-to-many mappings of genes to peaks, it is unknown if we have the same number of DE genes participating or less, as we increase the threshold on the DA LFC. We can examine the number of genes with overlapping DA peaks at various thresholds by grouping and aggregating twice. First, the number of peaks that meet the thresholds are computed within each gene, origin, and resample group. Second, within the origin column, we compute the total number of peaks that meet the DA LFC threshold and the number of genes that have more than zero peaks (again averaging over the number of resamples).

```
genes_olap_peaks %>%
  group_by(gene_id, origin, resample) %>%
  reduce_ranges_directed(
```

```
    lfc1 = sum(abs(da_log2FC) > 1, na.rm=TRUE),
    lfc2 = sum(abs(da_log2FC) > 2, na.rm=TRUE)
  ) %>%
  group_by(origin) %>%
  summarize(
    lfc1_gene_count = sum(lfc1 > 0) / n_distinct(resample),
    lfc1_peak_count = sum(lfc1) / n_distinct(resample),
    lfc2_gene_count = sum(lfc2 > 0) / n_distinct(resample),
    lfc2_peak_count = sum(lfc2) / n_distinct(resample)
  )
```

```
#> DataFrame with 2 rows and 5 columns
#>     origin lfc1_gene_count lfc1_peak_count lfc2_gene_count lfc2_peak_count
#>   <factor>       <numeric>       <numeric>       <numeric>       <numeric>
#> 1   not_de           271.2           369.5            30.3            32.5
#> 2   de               515.0          1097.0           151.0           234.0
```

To do this for many thresholds is cumbersome and would create a lot of duplicate code. Instead we create a single function called `count_above_threshold()` that accepts a variable and a vector of thresholds, and computes the sum of the absolute value of the variable for each element in the `thresholds` vector.

```
count_if_above_threshold <- function(var, thresholds) {
  lapply(thresholds, function(.) sum(abs(var) > ., na.rm = TRUE))
}
```

The above function will compute the counts for any arbitrary threshold, so we can apply it over possible LFC thresholds of interest. We choose a grid of one hundred thresholds based on the range of absolute LFC values in the `da_peaks` *GRanges* object:

```
thresholds <- da_peaks %>%
  mutate(abs_lfc = abs(da_log2FC)) %>%
  with(
    seq(min(abs_lfc), max(abs_lfc), length.out = 100)
  )
```

The peak counts for each threshold are computed as a new list-column called *value*. First, the *GRanges* object has been grouped by the gene, origin, and the number of resamples columns. Then we aggregate over those columns, so each row will contain the peak counts for all of the thresholds for a gene, origin, and resample. We also maintain another list-column that contains the threshold values.

```
genes_peak_all_thresholds <- genes_olap_peaks %>%
  group_by(gene_id, origin, resample) %>%
  reduce_ranges_directed(
    value = count_if_above_threshold(da_log2FC, thresholds),
    threshold = list(thresholds)
  )
genes_peak_all_thresholds
```

```
#> GRanges object with 8239 ranges and 5 metadata columns:
#>          seqnames              ranges strand |            gene_id   origin
#>             <Rle>           <IRanges>  <Rle> |        <character> <factor>
#>      [1]     chr1 196641878-196661877      + | ENSG00000000971.15       de
#>      [2]     chr6   46119993-46139992      + |  ENSG00000001561.6       de
#>      [3]     chr4   17567192-17587191      + | ENSG00000002549.12       de
#>      [4]     chr7 150790403-150810402      + |  ENSG00000002933.8       de
#>      [5]     chr4   15768275-15788274      + | ENSG00000004468.12       de
#>      ...      ...                 ...    ... .                ...      ...
#>   [8235]    chr17   43569620-43589619      - | ENSG00000175832.12   not_de
#>   [8236]    chr17   18250534-18270533      + | ENSG00000177427.12   not_de
```

```
#>   [8237]    chr20   63885182-63905181      + | ENSG00000101152.10   not_de
#>   [8238]     chr1   39071316-39091315      + | ENSG00000127603.25   not_de
#>   [8239]     chr8   41567187-41587186      + | ENSG00000158669.11   not_de
#>           resample         value                          threshold
#>          <integer> <IntegerList>                      <NumericList>
#>      [1]         0     1,1,1,... 0.0658243,0.1184840,0.1711436,...
#>      [2]         0     1,1,1,... 0.0658243,0.1184840,0.1711436,...
#>      [3]         0     6,6,6,... 0.0658243,0.1184840,0.1711436,...
#>      [4]         0     4,4,4,... 0.0658243,0.1184840,0.1711436,...
#>      [5]         0  11,11,11,... 0.0658243,0.1184840,0.1711436,...
#>      ...       ...           ...                                ...
#>   [8235]        10     1,1,1,... 0.0658243,0.1184840,0.1711436,...
#>   [8236]        10     3,3,2,... 0.0658243,0.1184840,0.1711436,...
#>   [8237]        10     5,5,5,... 0.0658243,0.1184840,0.1711436,...
#>   [8238]        10     3,3,3,... 0.0658243,0.1184840,0.1711436,...
#>   [8239]        10     3,3,3,... 0.0658243,0.1184840,0.1711436,...
#>   -------
#>   seqinfo: 25 sequences (1 circular) from hg38 genome
```

Now we can expand these list-columns into a long *GRanges* object using `expand_ranges()`. This function will unlist the *value* and *threshold* columns and lengthen the resulting *GRanges* object. To compute the peak and gene counts for each threshold, we apply the same summarization as before:

```
origin_peak_all_thresholds <- genes_peak_all_thresholds %>%
  expand_ranges() %>%
  group_by(origin, threshold) %>%
  summarize(
    gene_count = sum(value > 0) / n_distinct(resample),
    peak_count = sum(value) / n_distinct(resample)
```

```
  )
origin_peak_all_thresholds
```

```
#> DataFrame with 200 rows and 4 columns
#>       origin threshold gene_count peak_count
#>     <factor> <numeric>  <numeric>  <numeric>
#> 1     not_de 0.0658243      708.0     2391.4
#> 2     not_de 0.1184840      698.8     2320.6
#> 3     not_de 0.1711436      686.2     2178.6
#> 4     not_de 0.2238033      672.4     1989.4
#> 5     not_de 0.2764629      650.4     1785.8
#> ...      ...       ...        ...        ...
#> 196       de   5.06849          2          2
#> 197       de   5.12115          0          0
#> 198       de   5.17381          0          0
#> 199       de   5.22647          0          0
#> 200       de   5.27913          0          0
```

Again we can compute the relative enrichment in LFCs in the same manner as before, by pivoting the results to long form then back to wide form to compute the enrichment.

```
origin_threshold_counts <- origin_peak_all_thresholds %>%
  as.data.frame() %>%
  pivot_longer(cols = -c(origin, threshold),
               names_to = c("type", "var"),
               names_sep = "_",
               values_to = "count") %>%
  select(-var)
```

We visualize the peak enrichment changes of DE genes relative to other genes as a line chart:

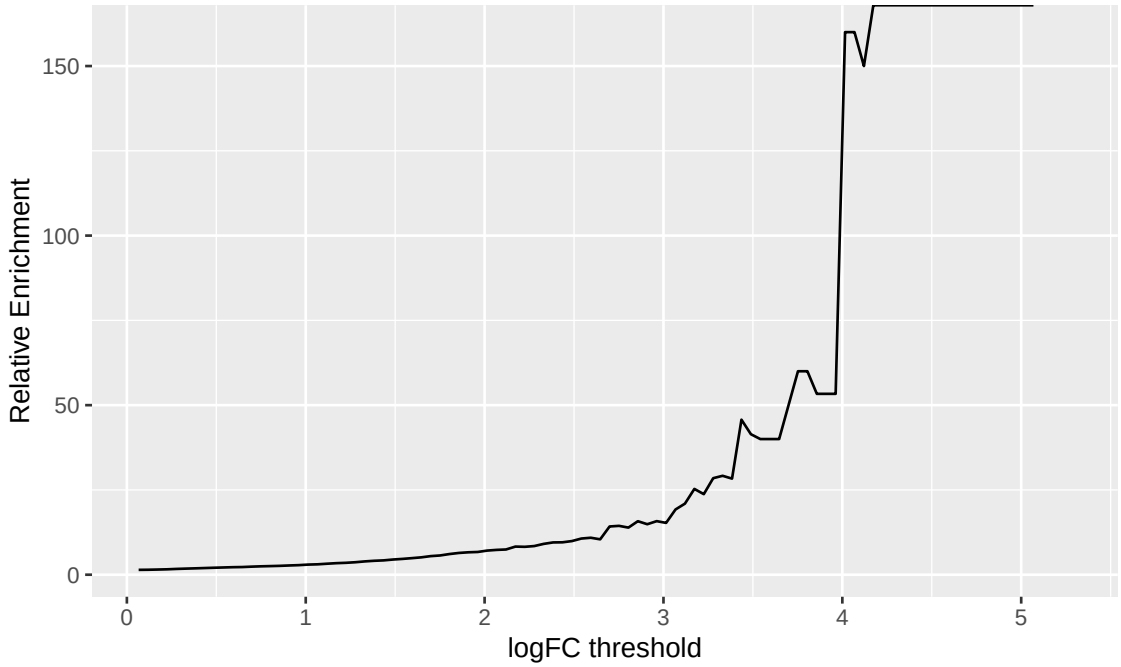


Figure 3.5: *A line chart displaying how relative enrichment of DA peaks change between DE genes compared to non-DE genes as the absolute DA LFC threshold increases.*

```
origin_threshold_counts %>%
  filter(type == "peak") %>%
  pivot_wider(names_from = origin, values_from = count) %>%
  mutate(enrichment =  de / not_de) %>%
  ggplot(aes(x = threshold, y = enrichment)) +
  geom_line() +
  labs(x = "logFC threshold", y = "Relative Enrichment")
```

We computed the sum of DA peaks near the DE genes, for increasing LFC thresholds on the accessibility change. As we increased the threshold, the number of total peaks went down (likewise the mean number of DA peaks per gene). It is also likely the number of DE genes with a DA peak nearby with such a large change went down. We can investigate this with a plot that summarizes many of the aspects underlying the enrichment plot above.

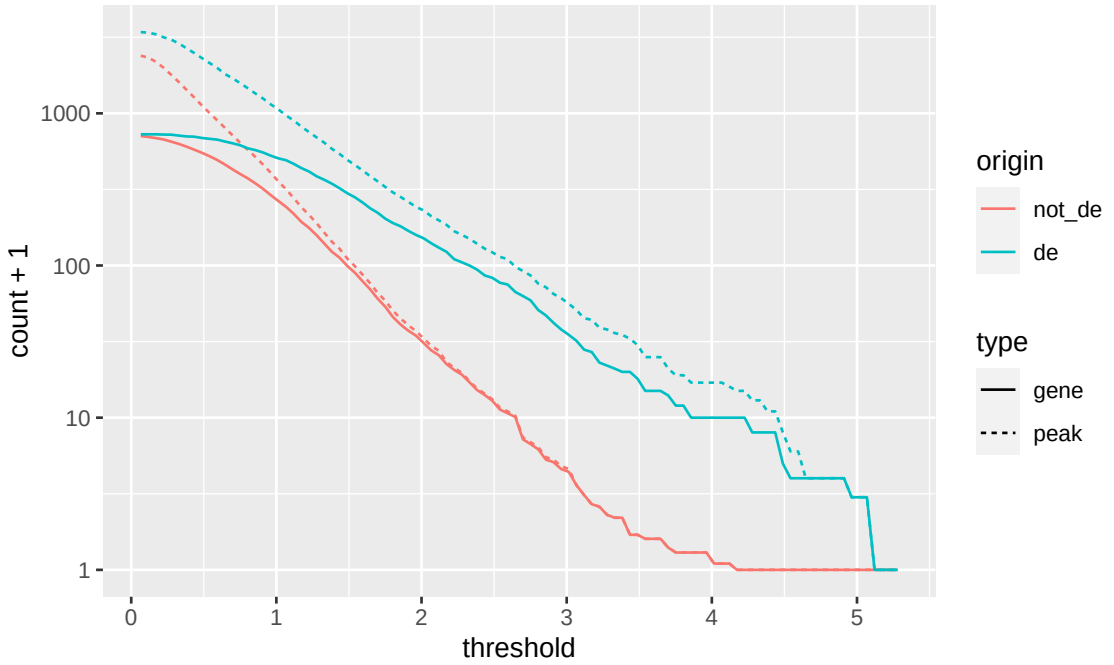


Figure 3.6: *A line chart displaying how gene and peak counts change as the absolute DA LFC threshold increases. Lines are colored according to whether they represent a gene that is DE or not. Note the x-axis is on a log_{10} scale.*

```
origin_threshold_counts %>%
  ggplot(aes(x = threshold,
             y = count + 1,
             color = origin,
             linetype = type)) +
  geom_line() +
  scale_y_log10()
```

3.5 Discussion

We have shown that by using **plyranges** and **tximeta** (with the support of the Bioconductor and **tidyverse** ecosystems) we can iterate through the biological data science workflow: from import, through to modeling, and data integration.

There are several further steps that would be interesting to perform in this analysis; for example, we could modify window size around the TSS to see how it affects enrichment,

and vary the FDR cut-offs for both the DE gene and DA peak sets. We could also have computed variance in addition to the mean of the resampled set, and so drawn an interval around the enrichment line.

Finally, our workflow illustrates the benefits of using appropriate data abstractions provided by Bioconductor such as the *SummarizedExperiment* and *GRanges*. These abstractions provide users with a mental model of their experimental data and are the building blocks for constructing the modular and iterative analyses we have shown here. Consequently, we have been able to interoperate many decoupled R packages (from both Bioconductor and the **tidyverse**) to construct a seamless end-to-end workflow that is far too specialized for a single monolithic tool.

3.6 Software Availability

The workflow materials can be fully reproduced following the instructions found at the Github repository sa-lee/fluentGenomics. Moreover, the development version of the workflow and all downstream dependencies can be installed using the **BiocManager** package by running:

```
# development version from Github
BiocManager::install("sa-lee/fluentGenomics")
# version available from Bioconductor
BiocManager::install("fluentGenomics")
```

Acknowledgements

We would like to thank all participants of the Bioconductor 2019 and BiocAsia 2019 conferences who attended and provided feedback on early versions of this workflow paper.

Chapter 4

Exploratory coverage analysis with superintronic and plyranges

Here we consider a tidy-data approach for exploring estimated coverage from RNA-seq data. We establish a simple framework, for aggregating across experimental design, and annotated genomic regions to discover 'interesting' coverage trace plots. We highlight how this framework can be used to develop data descriptions that find putative genes with intron retention. Our framework is implemented in a software package called **superintronic**, available at `https://github.com/sa-lee/superintronic`.

4.1 Introduction

In high-throughput sequencing data sets, coverage is the estimated number of reads that overlap a single position of the reference genome, and is important for assessing sequencing data quality and used in many different aspects of omics analysis such peak-calling in ChIP-seq or variant calling in DNA-seq (Sims et al., 2014). Here we emphasise looking at coverage traces to find biological events of interest, rather than only relying on numerical summaries of the data (Figure 4.1. By faceting these traces over combinations of the experimental design and with their biological context such as gene annotations, we can gain an insight into biological signal under study. Of course, due to the sheer size of most reference genome annotations it would take an extremely long time for an

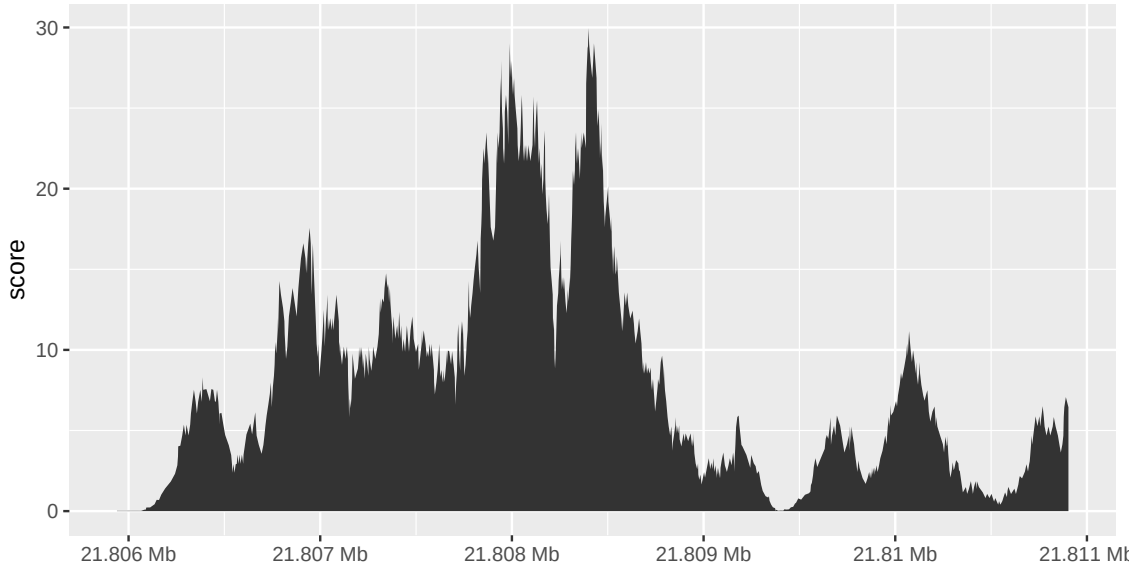


Figure 4.1: *(ref:example-trace)*

analyst to look at every single possible region where there is some interesting signal within the coverage trace. Because visualisation does not scale, we need to search the possible space of coverage traces and provide diagnostics for identifying traces with interesting biology. We have taken inspiration from visualisation literature; in particular the idea of scatter plot diagnostics ("scagnostics") for summarising the space of all possible 2D scatter plots to a small number of descriptors of each scatter plots properties such as density or monotonicity (Friedman and Stuetzle, 2002; Wilkinson, Anand, and Grossman, 2005). Similarly, searching for unusual time series via estimating descriptors such as seasonality or autocorrelation, and visualising those descriptors instead (Hyndman, Wang, and Laptev, 2015).

(ref:example-trace): An illustrative coverage trace plot from Chapter 2. The coverage *score* is defined as the number of reads that overlap a single base position within a reference genome. For example, at position 21.807 of chromosome 8 there are approximately 12 reads overlapping it. One could also think of a coverage trace as a histogram with the bin width varies as you move along the genome.

Although there are many flexible and powerful software tools, like **BEDtools** or **deepTools**, for exploring and estimating coverage from common genomic data formats (Quinlan and Hall, 2010; Ramrez et al., 2014), it is advantageous to have tooling that is tightly coupled

to a statistical computing language such as R. This allows interoperability between other software packages for data wrangling, visualisation and modelling within the ecosystem of the language that may not be possible with a single command line tool. There are also gains in reproducibility as analysts do not have to move between multiple software suites to explore their processed transcriptomics data.

In Lee et al. (2020), we showed that there is evidence that a major source of intron reads in RNA-seq datasets is pre-mRNA, and sought data analysis techniques to unravel different aspects of intron signal. In light of this, we made the assumption that most intron reads do not necessarily point to intron retention (IR) events, and developed a workflow based on combining multiple summary statistics, "data descriptors", to find coverage traces that appear to have IR-like events by collapsing coverage scores over a design matrix alongside the exonic and intronic parts of a gene. To do this we developed a new R package called **superintronic** that provides tooling for exploratory coverage analysis by extending and integrating our previous software package, **plyranges** (Lee, Cook, and Lawrence, 2019).

In this paper, we describe methodology for establishing data descriptors by turning coverage vectors into long form tidy data using **superintronic** and **plyranges**. We provide a workflow below using a zebrafish RNA-seq dataset for developing data descriptors to find IR like coverage traces within genes known to have minor class splicing events.

4.2 Methods

superintronic is an R package used for estimating, representing, and visualising per-base coverage scores, that can then be flexibly summarised over factors within an experimental design and collapsed over regions of the genome using our companion package **plyranges**. The aspects of this combined workflow are summarised in Figure 4.2.

4.2.1 Representation of coverage estimation

The per base coverage score is estimated directly from one or more BAM files that represent the units within the experimental design, along with an optional experimental design table that is returns a long-form tidy *GRanges* data structure. The coverage estimation

Figure 4.2: *An overview of the* ***superintronic*** *and* ***plyranges*** *workflow. Coverage is estimated directly using the design matrix that contains a source column pointing to the locations of BAM files. The long-form representation is output as a* GRanges *object, and contains columns that were part of the design. Additional annotations are added with join functions, here we show the particular case of expanding the coverage* GRanges *to include exonic and intronic parts of a gene. This object can be further analysed using* ***plyranges*** *and our data descriptors approach, and then descriptors can be visualised as a scatter plot matrix with the* ***GGally*** *and* ***ggplot2*** *packages (Schloerke et al., 2020; Wickham, 2016). Coverage traces can be directly generated with* ***superintronic*** *and collapsed over parts of the design matrix to identify differences between groups.*

is computed via the **Rsamtools** package, and users have the ability to estimate coverage in parallel and drop regions in the genome where there is no coverage (Morgan et al., 2020; Lawrence et al., 2013b). This representation is tidy, since each row of the resulting *GRanges* data structure corresponds to position(s) within a given sample with a given coverage score alongside any variables such as biological group. While the long-form representation repeats the same information for a sample within the design, the size of the resulting *GRanges* in memory can be compressed using run-length encoding for any categorical variable, which is a form of data compression where "runs" of a vector are stored rather than their values. For example, the character vector of letters "a", and "b" is shortened so successive values are stored as a single value with their lengths:

```
#>  [1] "b" "a" "b" "b" "a" "a" "b" "a" "a" "a"
```

```
#> character-Rle of length 10 with 6 runs
#>   Lengths:   1   1   2   2   1   3
#>   Values : "b" "a" "b" "a" "b" "a"
```

This representation is memory efficient: at worst it will be the same size as the input, and at best reduce the size by a factor corresponding to the largest run. The allows us to easily transform the coverage scores and integrate annotations using the **plyranges** grammar, and visualise traces using **ggplot2** (Wickham, 2016).

4.2.2 Integration of external annotations

External reference annotations, perhaps transcripts or exons, can be coerced to *GRanges* objects, are incorporated into the coverage *GRanges* by taking the intersection of the annotation with the *GRanges* using an overlap intersect join from the **plyranges** software. The resulting intersection will now contain the per base coverage that are overlapped the genomic features in the annotation, along side any metadata about the features themselves. Since our main workflow interest is in discovering coverage traces with IR profiles, **superintronic** provides some syntactic sugar for unravelling gene annotations into their exonic and intronic parts, and intersecting them with a coverage *GRanges*.

4.2.3 Discovery of regions of interest via 'data descriptors'

Once the coverage *GRanges* has reference genomic features, data descriptors can be computed via collecting summary statistics across the factors of the experimental design and features of interest. This can be achieved using **plyranges** directly by first grouping across variables of interest, computing descriptors defined by **superintronic** and then pivoting the results into a wide form table for additional processing or visualisation. There are many descriptors defined by **superintronic** that are weighted statistics (as we have to account for the number of bases covered, or the width of the range) of the coverage score, such as the mean and standard deviation. There are also descriptors that can be used to find the number of times the coverage trace is above a certain number of bases or score. To find coverage traces that have unusual descriptors, the descriptors can be visualised directly as a scatter plot matrix. After that thresholds can be applied to filter the genomic

features that had extreme descriptors on the coverage *GRanges*, and the traces can be visualised. By default **superintronic** displays coverage traces oriented from the 5′ to 3′ end of the gene, with the `view_coverage()` function. Traces can be highlighted according to a genomic feature of interest, figures in this chapter have orange areas corresponding to intronic parts of the gene, while dark green areas referring to exonic parts.

4.3 A workflow for uncovering intron retention in a zebrafish experiment

In the study of gene regulation, there is much interest in uncovering the effects of aberrant minor class splicing (called U12 splicing) on the transcriptome. Minor class splicing is a regulatory process where a class of introns (called U12 introns in this case but there exist other classes in eukaryotic organisms such as U2 introns) are removed from pre-mRNA prior to gene expression (Turunen et al., 2013). The removal of these introns is catalysed by small ribonucleoproteins (snRNPs) which identify key motifs and branch sites in the intron to begin splicing (Markmiller et al., 2014). Here we describe an exploratory workflow for finding coverage traces with evidence for intron retention using RNA-seq data from a knockout experiment in zebrafish obtained from the Heath Lab at Walter and Eliza Hall Institute. Code for this analysis is available at `https://github.com/sa-lee/thesis/tree/master/scripts/superintronic.R`.

The data consist of 11 zebrafish samples from single-end polyA enriched RNA-seq libraries pooled from zebrafish larvae. The experimental factors looked at combinations of genotype (whether the gene rnpc3 has been knocked out or not) and line (whether the zebrafish larvae have the caliban *cal* or mutant caliban *zm-cal* phenotype). Within each combination there are three biological replicates, except for the combination *zm-cal* and wild-type rnpc3 which had two replicates.

FASTQ files were aligned to the GRCz11 reference genome using subjunc with the default parameters called from **Rsubread** to produce BAM files for each sample (Liao, Smyth, and Shi, 2013, 2019). The coverage was then estimated directly from the set of BAM files using **superintronic** into the long form *GRanges* representation we described above.

The gene annotation files were obtained as GFF files from RefSeq and used to construct the exonic and intronic parts of each gene as *GRanges* object using **superintronic**. We further filtered genes that had a single exon or genes that were overlapping others in the annotation and that were not on the main contigs of the reference genome (i.e. excluding mitochondria) to simplify our analysis and reduce any coverage ambiguity. This left 18,270 genes available for computing data descriptors on in order to detecting IR-like coverage traces. Across each combination of genotype and line, we first $\log_2$ transformed the coverage score with an offset of one, and then intersected the coverage *GRanges* with exonic and intronic features of each gene. For each gene, the mean and standard deviation of the log-transformed coverage score weighted by the number of bases covered were computed over all intron and exon parts within the groupings of genotype and line. We also computed the bases above descriptor for intron features. It refers to the total number of bases within an intron that has a score above the overall average exon coverage score. The scatter plot matrix view of these descriptors is shown in Figure 4.3 for a single biological group in the experiment. Using these views and by summarising over the data descriptors, we came up with thresholds for finding genes that have IR-like traces within each biological group.

We selected genes with the following thresholds: genes have an average exon log-coverage greater than the mean of average exon log-coverage values across all genes, have an average intron log-coverage greater than the average standard deviation of exon log-coverage values over all genes, and the standard deviation of intron log-coverage values is twice the standard deviation of exon log-coverage values. That is, we are selecting genes that are expressed but have large average intron expression that is more variable than the gene's exon expression. This results in a total of 86 genes selected to link back to their underlying coverage traces, with the overlaps shown in the UpSet plot in Figure 4.4 (Lex et al., 2014). The procedure produces gene coverage traces with known minor class splice sites affected by the knockout procedure such as ccdc43 and nat15 (figures 4.5 and 4.6), as well as some that appear to have U12 intron retention like events such as mapk3 or tspan31 (figures 4.7 and 4.8) that affect other parts of the gene.

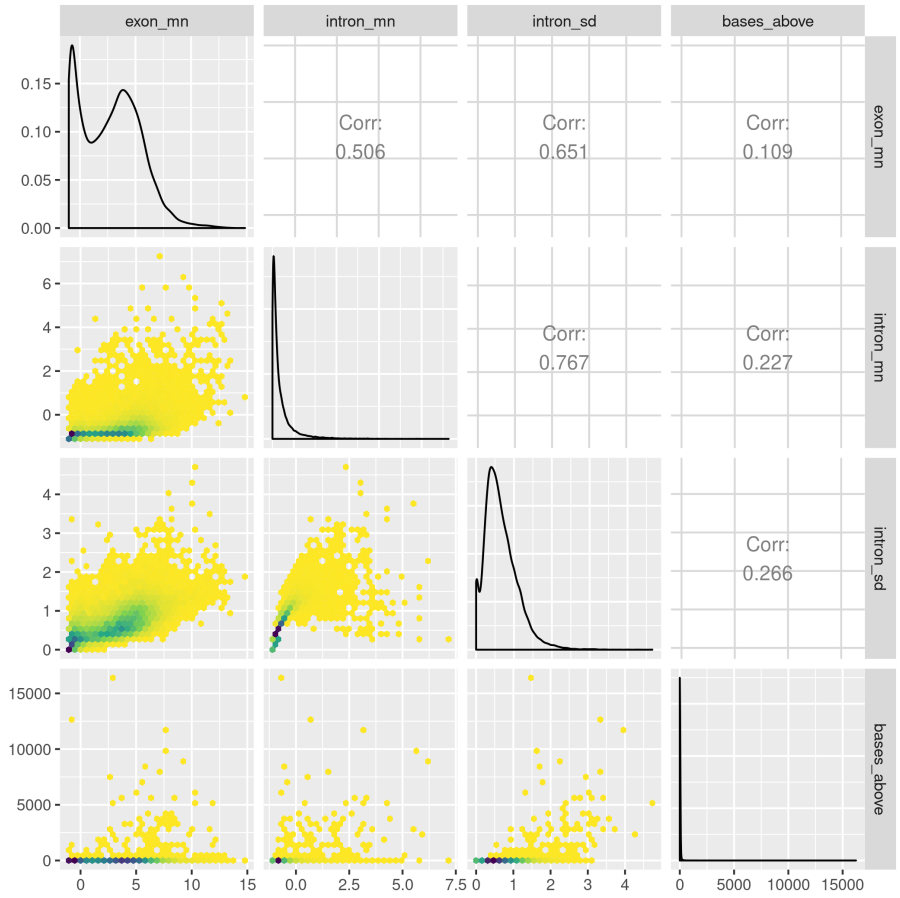


Figure 4.3: *A hexbin scatter plot matrix of the data descriptors estimated for the* cal *rnpc3 knock out zebrafish line. To identify coverage traces with IR like events, we want a set of descriptors that will find genes with the following characteristics: the gene is "expressed" that has a large number of intron bases relative to the coverage of other intron features, and has relatively stable coverage within exon features. To do this we looked at the descriptor, as well as computing the the mean and standard deviation of both exon and intron features.*

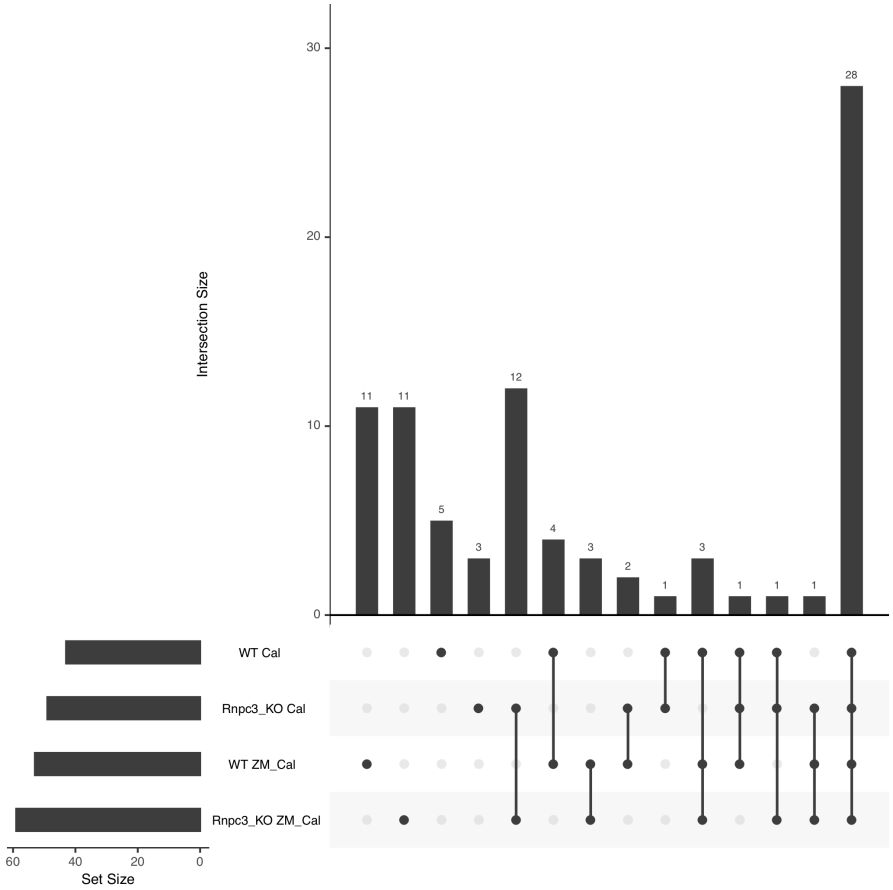


Figure 4.4: *Gene overlaps found between each combination of genotype and line using the thresholds defined in the text. Our procedure mostly finds genes with IR like profiles across all groups (28 shared between all four) or that is unique to a single group, since we do not consider looking at differential IR and run our thresholding separately for each group of replicates. The rnpc3 knockout lines share the largest overlap in results.*

Figure 4.5: *The ccdc43 gene is known to have enhanced U12 intron retention in the caliban phenotype and increase retention when rnpc3 is knocked out as can be seen directly from the intron located at around 31.603Mb.*

Figure 4.6: *The gene nat15 exhibits another example of U12 intron retention, located at around 27.93Mb, in the rnpc3 knockout groups.*

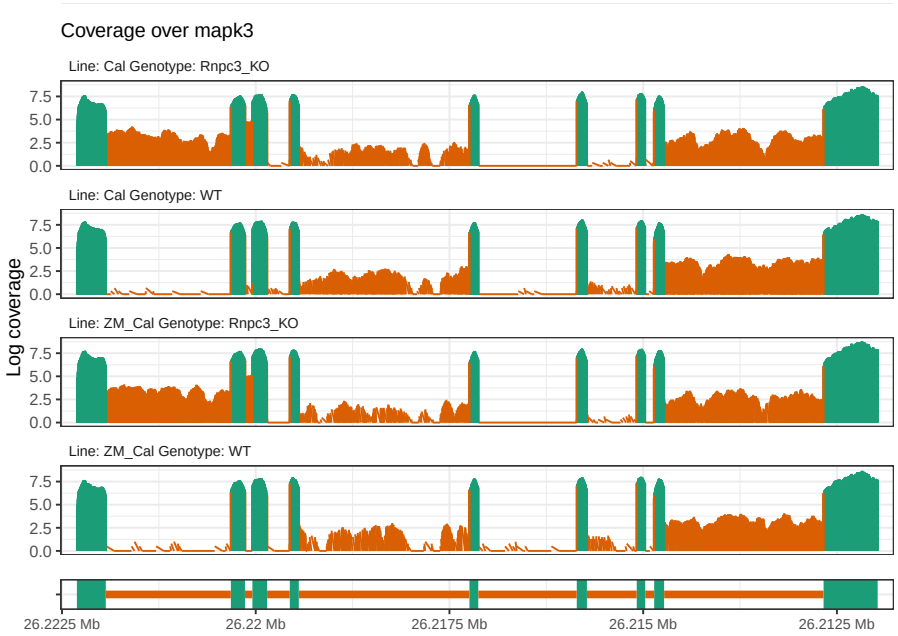


Figure 4.7: *The gene mapk3 appears to have intron retention close to the start of the gene that have different impacts downstream, which result in different IR profiles between the knockout and wild-type groups.*

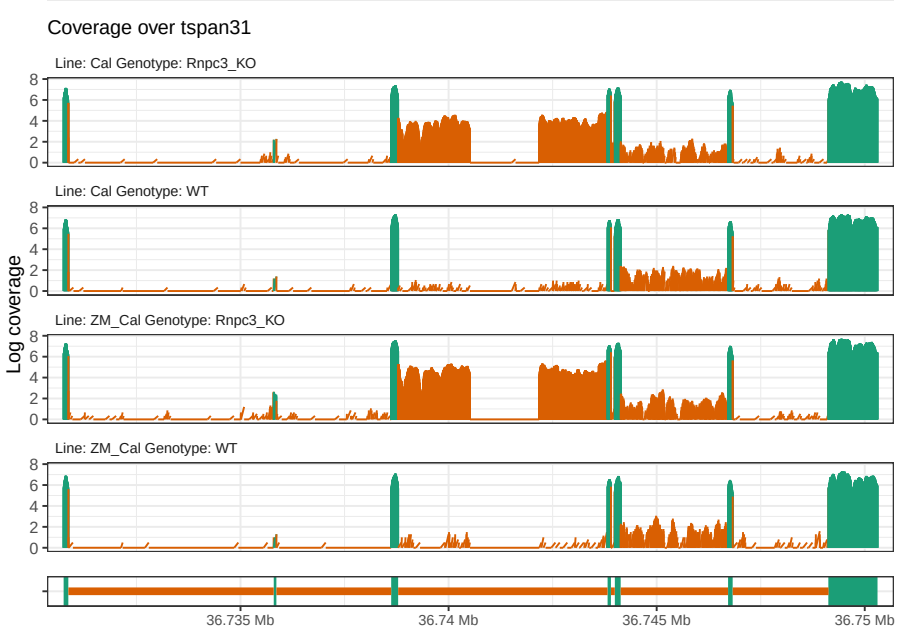


Figure 4.8: *The gene tspan31 has a potential cryptic splice site within the intron retained in the knockout groups.*

4.4 Discussion

We have shown how coverage can be represented in the tidy data framework and integrated with experimental metadata and reference annotations. This framework allowed us to build data descriptions that are simple aggregations of various aspects of genomic features over factors within a designed experiment and link those descriptions to their underlying coverage traces.

Our zebrafish workflow shows that our approach using **superintronic** and **plyranges** is able to uncover interesting biological signals in a purely data-driven manner. We did not include additional information that could have been useful when deriving our selected genes, such as sequence motifs for U12 class of introns, or exploit the experimental design to find differential IR like profiles. However, if that was of interest, one could look at the overlaps, like we did in Figure 4.4, or combine our data descriptors with external estimates using **limma** (Ritchie et al., 2015), like our proposed index method in Lee et al. (2020). The gene candidates obtained by our thresholds have been validated by the Heath lab using qPCR.

Although the example we have explored has related to finding coverage traces with IR-like events, the workflow of building and then visualising data descriptors could be generalised to other types of omics analyses, such as peaking finding in ChIP-seq, and to use more sophisticated methods for identifying thresholds of 'interesting' traces. Our approach would also greatly benefit from interactive graphics that dynamically link say a gene description to an underlying coverage trace, for rapid exploration. This is left for future work.

Acknowledgements

We would like to thank Dr Alexandra Garnham, Dr Stephen Mieruszynski and Associate Professor Joan Heath for providing the zebrafish data and helping interpret the results from the workflow.

Chapter 5

Casting multiple shadows: high-dimensional interactive data visualisation with tours and embeddings

There has been a rapid uptake in the use of non-linear dimensionality reduction (NLDR) methods such as t-distributed stochastic neighbour embedding (t-SNE) in the natural sciences as part of cluster orientation and dimension reduction workflows. The appropriate use of these methods is made difficult by their complex parameterisations and the multitude of decisions required to balance the preservation of local and global structure in the resulting visualisation. We present visual diagnostics for the pragmatic usage of NLDR methods by combining them with a technique called the tour. A tour is a sequence of interpolated linear projections of multivariate data onto a lower dimensional space. The sequence is displayed as a dynamic visualisation, allowing a user to see the shadows the high-dimensional data casts in a lower dimensional view. By linking the tour to a view obtained from an NLDR method, we can preserve global structure and through user interactions like linked brushing observe where the NLDR view may be misleading. We display several case studies from both simulated and real data from single

cell transcriptomics, that shows our approach is useful for cluster orientation tasks. The implementation of our framework is available as an R package called **liminal** available at https://github.com/sa-lee/liminal.

5.1 Introduction

High dimensional data is increasingly prevalent in the natural sciences and beyond but presents a challenge to the analyst in terms of data cleaning, pre-processing and visualisation. Methods to embed data from a high-dimensional space into a low-dimensional one now form a core step of the data analysis workflow where they are used to ascertain hidden structure and de-noise data for downstream analysis .

Choosing an appropriate embedding presents a challenge to the analyst. How does an analyst know whether the embedding has captured the underlying topology and geometry of the high dimensional space? The answer depends on the analyst's workflow. Brehmer et al. (2014) characterised two main workflow steps that an analyst performs when using embedding techniques: dimension reduction and cluster orientation. The first relates to dimension reduction achieved by using an embedding method, here an analyst wants to characterise and map meaning onto the embedded form, for example identifying batch effects from a high throughput sequencing experiment, or identifying a gradient or trajectory along the embedded form (Nguyen and Holmes, 2019). The second relates to using embeddings as part of a clustering workflow. Here analysts are interested in identifying and naming clusters and verifying them by either applying known labels or colouring by variables that are a-priori known to distinguish clusters. Both of these workflow steps rely on the embedding being representative of the original high dimensional dataset, and becomes much more difficult when there is no underlying ground truth.

As part of a visualization workflow, it's important to consider the perception and interpretation of embedding methods as well. Sedlmair, Munzner, and Tory (2013) showed that scatter plots were mostly sufficient for detecting class separation, however they also noted that often multiple embeddings were required. For the task of cluster identification, Lewis, Van der Maaten, and Sa (2012) showed experimentally that novice users of non-linear

embedding techniques were more likely to consider clusters of points on a scatter plot to be the result of a spurious embedding compared to advanced users who were aware of the inner workings of the embedding algorithm.

A complementary approach for visualizing structure in high dimensional data is the tour. A tour is a sequence of projections of a high dimensional dataset onto a low-dimensional basis matrix, and is represented as an animated visualization (Asimov, 1985; Buja and Asimov, 1986). Given the dynamic nature of the tour, user interaction is important for controlling and exploring the visualisation: the tour has been used previously by Wickham, Cook, and Hofmann (2015) for exploring statistical model fits and by Buja, Cook, and Swayne (1996) for exploring the space of factorial experimental designs.

While there has been much work on the algorithmic details of embedding methods, there are relatively few tools designed to assist users to interact with these techniques: when is an embedding sufficient for the task at hand? Several interactive interfaces have been proposed for evaluating or using embedding techniques. Buja et al. (2008) used tours to guide analysts during the optimisation of multidimensional scaling methods by extending their interactive visualisation software called **XGobi** and **GGobi** into a new tool called **GGvis** (Swayne, Cook, and Buja, 1998; Swayne et al., 2003; Swayne and Buja, 2004). Their interface allows the analyst to dynamically modify and check whether an MDS configuration has preserved the locality and closeness of points between the configuration and the original data. Ovchinnikova and Anders (2020) created the **Sleepwalk** interface for checking non-linear embeddings in single cell RNA-seq data. It provides a click and highlight visualisation for colouring points in an embedding according to an estimated pairwise distance in the original high-dimensional space. Similarly, the **TensorFlow** embedding projector is a web interface to running some non-linear embedding methods live in the browser and provides interactions to colour points, and select nearest neighbours (Smilkov et al., 2016). Finally, the work by Pezzotti et al. (2017) provides a user guided and modified form of the t-SNE algorithm, that allows users to modify optimisation parameters in real-time.

There is no one-size fits all: finding an appropriate embedding for a given dataset is a difficult and a somewhat poorly defined problem. For non-linear methods, there are

many parameters to explore that can have an effect on the resulting visualisation and interpretation. Interfaces for evaluating embeddings require interaction but should also be able to be incorporated into an analysts workflow. Instead, we implement a more pragmatic workflow by incorporating interactive graphics and tours with embeddings that allows users to see a global overview of their high dimensional data and assists them with cluster orientation tasks.

The rest of the paper is organised as follows. The next section provides background on dimension reduction methods, including an overview of the tour. Then we describe the visual design of **liminal**, followed by implementation details. Next we provide case studies that show how our interface assists in using embedding methods. Finally, we describe the insights gained by using **liminal** and plans for extensions to the software.

5.2 Overview of Dimension Reduction

In the following section, we provide a brief overview of the goals of DR methods with respect to the data analyst, in addition to their high level the mathematical details. Here we restrict our attention to two recent methods that are commonly used in the literature: t-distributed stochastic neighbour embedding (t-SNE) and uniform manifold alignment and projection (UMAP); however we do mention several other techniques (Maaten and Hinton, 2008; McInnes, Healy, and Melville, 2018).

To begin we suppose the data is in the form rectangular matrix of real numbers, $X = [\mathbf{x_1}, \ldots, \mathbf{x_n}]$, where n is the number of observations in p dimensions. The purpose of any DR algorithm is to find a low-dimensional representation of the data, $Y = [\mathbf{y_1}, \ldots, \mathbf{y_n}]$, such that Y is an $n \times d$ matrix where $d \ll p$. The hope of the analyst is that the DR procedure to produce Y will remove noise in the original dataset while retaining any latent structure.

DR methods can be classified into two broad classes: linear and non-linear methods. Linear methods perform a linear transformation of the data, that is, Y is a linear transformation of X; one example is principal components analysis (PCA) which performs an eigendecomposition of the estimated sample covariance matrix (Hotelling, 1933). The

eigenvalues are sorted in decreasing order and represent the variance explained by each component (eigenvector). A common approach to deciding on the number of principal components to retain is to plot the proportion of variance explained by each component and choose a cut-off.

For non-linear methods Y is generated via a pre-processed form of the input X such as the k-nearest neighbours graph or via a kernel transformation. Multidimensional scaling (MDS) is a class of DR method that aims to construct an embedding Y such that the pair-wise distances (inner products) in Y approximate the pair-wise distances (inner products) in X (Torgerson, 1952; Kruskal, 1964a). There are many variants of MDS, such as non-metric scaling which amounts to replacing distances with ranks instead (Kruskal, 1964b). A related technique is Isomap which uses a k-nearest neighbour graph to estimate the pair-wise geodesic distance of points in X then uses classical MDS to construct Y (Silva and Tenenbaum, 2003). Other approaches are based on diffusion processes such as diffusion maps (Coifman et al., 2005). A recent example of this approach is the PHATE algorithm (Moon et al., 2019). Here an affinity matrix is estimated via the pair-wise distance matrix and k-nearest neighbours graph of X. The algorithm de-noises estimated distances in high-dimensional space via transforming the affinity matrix into a Markov transition probability matrix and diffusing this matrix over a fixed number of time steps. Then the diffused probabilities are transformed once more to construct a distance matrix, and classical MDS is used to generate Y. A general difficulty with using non-linear DR methods for exploratory data analysis is selecting and tuning appropriate parameters. To see the extent of these choices we now examine the underpinnings of t-SNE and UMAP.

The t-SNE algorithm estimates the pair-wise similarity of (Euclidean) distances of points in a high dimensional space using a Gaussian distribution and then estimates a configuration in the low dimensional embedding space by modelling similarities using a t-distribution with 1 degree of freedom (Maaten and Hinton, 2008). There are several subtleties to the to use of the algorithm that are revealed by stepping through its machinery.

To begin, t-SNE transforms pair-wise distances between $\mathbf{x_i}$ and $\mathbf{x_j}$ to similarities using a Gaussian kernel:

$$p_{i|j} = \frac{\exp(-\|\mathbf{x_i} - \mathbf{x_j}\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|\mathbf{x_j} - \mathbf{x_k}\|^2 / 2\sigma_i^2)}$$

The conditional probabilities are then normalised and symmetrised to form a joint probability distribution via averaging:

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2n}$$

The variance parameter of the Gaussian kernel is controlled by the analyst using a fixed value of perplexity for all observations:

$$\text{perplexity}_i = \exp(-\log(2) \sum_{i \neq j} p_{j|i} \log_2(p_{j|i}))$$

As the perplexity increases, σ_i^2 increases, until it is bounded above by the number of observations , $n - 1$, in the data, corresponding to $\sigma_i^2 \to \infty$. This essentially turns $t - SNE$ into a nearest neighbours algorithm, $p_{i|j}$ will be close to zero for all observations that are not in the $\mathcal{O}(\text{perplexity}_i)$ neighbourhood graph of the ith observation (Maaten, 2014).

Next, the target low-dimensional space, Y, pair-wise distances between $\mathbf{y_i}$ and $\mathbf{y_j}$ are modelled as a symmetric probability distribution using a t-distribution with one degree of freedom (Cauchy kernel):

$$q_{ij} = \frac{w_{ij}}{Z} \text{where } w_{ij} = \frac{1}{1 + \|\mathbf{y_i} - \mathbf{y_j}\|^2} \text{ and } Z = \sum_{k \neq l} w_{kl}.$$

The resulting embedding Y is the one that minimizes the Kullback-Leibler divergence between the probability distributions formed via similarities of observations in X, $\mathcal{P}$ and similarities of observations in Y, $\mathcal{Q}$:

$$\mathcal{L}(\mathcal{P}, \mathcal{Q}) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

Re-writing the loss function in terms of attractive (right) and repulsive (left) forces we obtain:

$$\mathcal{L}(\mathcal{P}, \mathcal{Q}) = -\sum_{i \neq j} p_{ij} \log w_{ij} + \log \sum_{i \neq j} w_{ij}$$

So when the loss function is minimised this corresponds to large attractive forces, that is, the pair-wise distances in Y are small when there are non-zero p_{ij}, i.e. x_i and x_j are close together. The repulsive force should also be small when the loss function is minimised, that is, when pair-wise distances in Y are large regardless of the magnitude of the corresponding distances in X.

Taken together, these details reveal the sheer number of decisions that an analyst must make. How does one choose the perplexity? How should the parameters that control the optimisation of the loss function (done with stochastic gradient descent), like the number of iterations, or early exaggeration (a multiplier of the attractive force at the beginning of the optimisation), or the learning rate be selected? It is a known problem that t-SNE can have trouble recovering topology and that configurations can be highly dependent on how the algorithm is initialised and parameterized (Wattenberg, Viégas, and Johnson, 2016; Kobak and Berens, 2019; Melville, 2020). If the goal is cluster orientation a recent theoretical contribution by Linderman and Steinerberger (2019) proved that t-SNE can recover spherical and well separated cluster shapes, and proposed new approaches for tuning the optimisation parameters. However, the cluster sizes and their relative orientation from a $t-SNE$ view can be misleading perceptually, due to the algorithms emphasis on locality.

Another recent method, UMAP, has seen a large rise in popularity (at least in single cell transcriptomics) (McInnes, Healy, and Melville, 2018). It is a method that is related to LargeVis (Tang et al., 2016), and like t-SNE acts on the k-nearest neighbour graph. Its main differences are that it uses a different cost function (cross entropy) which is optimized using stochastic gradient descent and defines a different kernel for similarities in the low dimensional space. Due to its computational speed it is possible to generate UMAP embeddings in more than three dimensions. It appears to suffer from the same

perceptual issues as t-SNE, however it supposedly preserves global structure better than t-SNE (Coenen and Pearce, 2019).

5.2.1 Tours explore the subspace of d-dimensional projections

The tour is a visualisation technique that is grounded in mathematical theory, and is able to ascertain the shape and global structure of a dataset via inspection of the subspace generated by the set of low-dimensional projections (Asimov, 1985; Buja and Asimov, 1986).

Like when using other DR techniques, the tour assumes we have a real data matrix X consisting of n observations in p dimensions. First, the tour generates a sequence of $p \times d$ orthonormal projection matrices (bases) $A_{t\in\mathbb{N}}$, where d is typically 1 or 2. For each pair of orthonormal bases A_t and A_{t+1} that are generated, the geodesic path between them is interpolated to form intermediate frames, and giving the sense of continuous movement from one basis to another. The tour is then the continuous visualisation of the projections $Y_t = XA_t$, that is the projection of X onto A_t as the tour path is interpolated between successive bases. A *grand tour* corresponds to choosing new orthonormal bases at random; allowing a user to ascertain structure via exploring the subspace of d-dimensional projections. In practice, we first sphere our data via principal components to reduce dimensionality of X prior to running the tour. Instead of picking projections at random, a *guided tour* can be used to generate a sequence of 'interesting' projections as quantified by an index function (Cook et al., 1995). While our software, **liminal**, is able to visualise guided tours, our focus in the case studies uses the grand tour to see global structure in the data.

5.3 Visual Design

Tours provide a supportive visualisation to NLDR graphics, and can be easily incorporated into an analysts workflow with our software package, **liminal**. Our interface allows analysts to quickly compare views from embedding methods and see how an embedding method preserves or alters the geometry of their data. Using multiple concatenated and linked views with the tour enhances interaction techniques, and allows analysts to perform

cluster orientation tasks via linked highlighting and brushing (McDonald, 1982; Becker and Cleveland, 1987). This approach allows our interface to achieve the three principles for interactive high-dimensional data visualisation outlined by Buja, Cook, and Swayne (1996): finding gestalt, posing queries, and making comparisons.

5.3.1 Finding Gestalt: focus and context

To investigate latent structure and the shape of a high dimensional dataset, a tour can be run without the use of an external embedding. It is often useful to first run principal components on the input as an initial dimension reduction step, and then tour a subset of those components instead, i.e. by selecting them from a scree plot. The default tour layout is a scatter plot with an axis layout displaying the magnitude and direction of each basis vector. Since the tour is dynamic, it is often useful to be able to pause and highlight a particular view. In addition to pause, play and reset buttons, brushing will pause the tour path, allowing users to identify 'interesting' projections. The domain of the axis scales from running a tour is called the half range, and is computed by rescaling the input data onto hyper-dimensional unit cube. We bind the half range to a mouse wheel event, allowing a user to pan and zoom on the tour view dynamically. This is useful for peeling back dense clumps of points to reveal structure.

5.3.2 Posing Queries: multiple views, many contexts

We have combined the tour view in a side by side layout with a scatter plot view as has been done in previous tour interfaces **XGobi** and **DataViewer** (Buja, Hurley, and McDonald, 1986; Swayne, Cook, and Buja, 1998). These views are linked; analysts can brush regions or highlight collections of points in either view. Linked highlighting can be performed when points have been previously labelled according to some discrete structure, i.e. cluster labels are available. This is achieved via the analyst clicking on groups in the legend, which causes unselected groupings to have their points become less opaque. Consequently, simple linked highlighting can alleviate a known downfall of methods such as UMAP or t-SNE: that is distances between clusters are misleading. By highlighting corresponding clusters in the tour view, the analyst can see the relationship

between clusters, and therefore obtain a more accurate representation of the topology of their data.

Simple linked brushing is achieved via mouse-click and drag movements. By default, when brushing occurs in the tour view, the current projection is paused and corresponding points in the embedding view are highlighted. Likewise, when brushing occurs in the embedding view, corresponding points in the tour view are highlighted. In this case, an analyst can use brushing for manually identifying clusters and verifying cluster locations and shapes: brushing in the embedding view gives analysts a sense of the shape and proximity of cluster in high-dimensional space.

5.3.3 Making comparisons: revising embeddings

As mentioned previously, when using any DR method, we are assuming the embedding is representative of the high-dimensional dataset it was computed from. Defining what it means for embedding to be 'representative' or 'faithful' to high-dimensional data is ill-posed and depends on the underlying task an analyst is trying to achieve. At the very minimum, we are interested in distortions and diffusions of the high-dimensional data. Distortions occur when points that are near each other in the embedding view are far from each other in the original dataset. This implies that the embedding is not continuous. Diffusions occur when points are far from each other in the embedding view are near in the original data. Whether points are near or far is reliant on the distance metric used; distortions and diffusions can be thought of as the preservation of distances or the nearest neighbours graphs between the high-dimensional space and the embedding space. As distances can be noisy in high-dimensions, ranks can be used instead as has been proposed by Lee and Verleysen (2009). Identifying distortions and diffusions allows an analyst to investigate the quality of their embedding and revise them iteratively.

These checks are done visually using our side-by-side tour and embedding views. In the simplest case, a local continuity check can be assessed via one to one linked brushing from the embedding to the tour view. Similarly, diffusions are identified from linked brushing on the tour view, highlighting points in the embedding view.

5.4 Software Infrastructure

We have implemented the above design as an open source R package called **liminal** (Lee and Cook, 2020). The package allows analysts to construct concatenated visualisations, drawn with the **Vega-Lite** grammar of interactive graphics via the **vegawidget** package (Satyanarayan et al., 2017; Lyttle and Vega/Vega-Lite Developers, 2020). It provides an interface for constructing linked and stand alone interfaces for manipulating tour paths via the **shiny** and **tourr** packages (Chang et al., 2020; Wickham et al., 2011).

5.4.1 Tours as a streaming data problem

The process of generating successive bases and interpolating between them to construct intermediary frames, means the tour is a dynamic visualisation technique. Generally, the user would set $d = 2$ and the tour is visualised as an animated scatter plot. This process of constructing bases and intermediate frames and visualising the resulting projection is akin to making a "flip book" animation. Like with a flip book, an interface to the tour requires the ability to interact and modify it in real time. The user interface generated in **liminal** allows a user to play, pause, and reset the tour animation, panning and zooming to modify the scales of the plot to provide context and click events to highlight groups of points if a labelling variable has been placed on the legend.

These interactions are enabled by treating the basis generation as a reactive stream. Instead of realising the entire sequence, which limits the animation to have a discrete number of frames, new bases and their intermediate frames are generated dynamically via pushing the current projection to the visualisation interface. The interface listens to events like pressing a button or mouse-dragging and reacts by pausing the stream. This process allows the user to manipulate the tour in real time rather than having to fix the number of bases ahead of time. Additionally, once the user has identified an interesting projection or is done with the tour, the interface will return the current basis for use downstream.

5.4.2 Linking and highlighting views via interactions

The embedding and tour views are linked together via rectangular brushes; when a brush is active, points will be highlighted in the adjacent view. Because the tour is dynamic, brush events that become active will pause the animation, so that a user can interrogate the current view. By default, brushing on the embedding view will produce a one-to-one linking with the tour view. For interpreting specific combinations of clusters, the multiple guides on the legend can be selected in order to see their relative orientations. The interface is constructed as a **shiny** gadget specifically designed for interactive data analysis. Selections such as brushing regions and the current tour path are returned after the user clicks done on the interface and become available for further investigation.

5.5 Case Studies

The next section steps through case studies of our approach using simulations and an application to single cell RNA-seq data.

The first three case studies use simulations where the cluster structure and geometry of the underlying data is known. We start with a simple example where we generated spherical clusters that are embedded well by t-SNE. Then we move onto more complex examples where the tour provides insight, such as clusters that have substructure and where there is more complex geometry in the data.

In the final case study, we apply our approach to clustering the mouse retina data from Macosko et al. (2015), and apply the tour to the process of verifying marker genes that separate clusters.

We *strongly* recommend viewing the linked videos for each case study while reading. Links to the videos are available in table 5.1 and in the figures for each case study. The videos presented show the visual appearance of the **liminal** interface, and how we can interact with the tour via the controls previously described. If you are unable to view the videos, the figures in each case study consist of screenshots that summarise what is learned from combining the tour and an embedding view.

5.5.1 Case Study 1: Exploring spherical Gaussian clusters

To begin we look at simulated datasets that reproduce known facts about the t-SNE algorithm. Our first data set consists of five spherical 5-*d* Gaussian clusters embedded in 10-*d* space, each cluster has the same covariance matrix. We then computed a t-SNE layout with default settings using the **Rtsne** package (Krijthe, 2015), and set up the **liminal** linked interface with grand tour on the 10-*d* observations.

From the video linked in Figure 5.1, we learn that t-SNE has correctly split out each cluster and laid them out in a star like formation. This is agrees with the tour view, where once we start the animation, the five clusters begin to appear but generally points are more concentrated in the projection view compared to the t-SNE layout (Figure 5.1a). This can be seen via brushing the t-SNE view (Figure 5.1b).

5.5.2 Case Study 2: Exploring spherical Gaussian clusters with hierarchical structure

Next we view Gaussian clusters from the *Multi Challenge Dataset*, a benchmark simulation data set for clustering tasks (Rauber, 2009). This dataset has two Gaussian clusters with equal covariance embedded in 10-*d*, and a third cluster with hierarchical structure. This cluster has two 3-*d* clusters embedded in 10-*d*, where the second cluster is subdivided into three smaller clusters, that are each equidistant from each other and have the same covariance structure. From the video linked in Figure 5.2, we see that t-SNE has correctly identified the sub-clusters. However, their relative locations to each other is distorted, with the orange and blue groups being far from each other in the tour view (Figure 5.2a). We see in this case that is difficult to see the sub-clusters in the tour view, however, once we zoom and highlight they become more apparent (Figure 5.2b). When we brush the sub-clusters in the t-SNE, their relative placement is again exaggerated, with the tour showing that they are indeed much closer than the impression the t-SNE view gives.

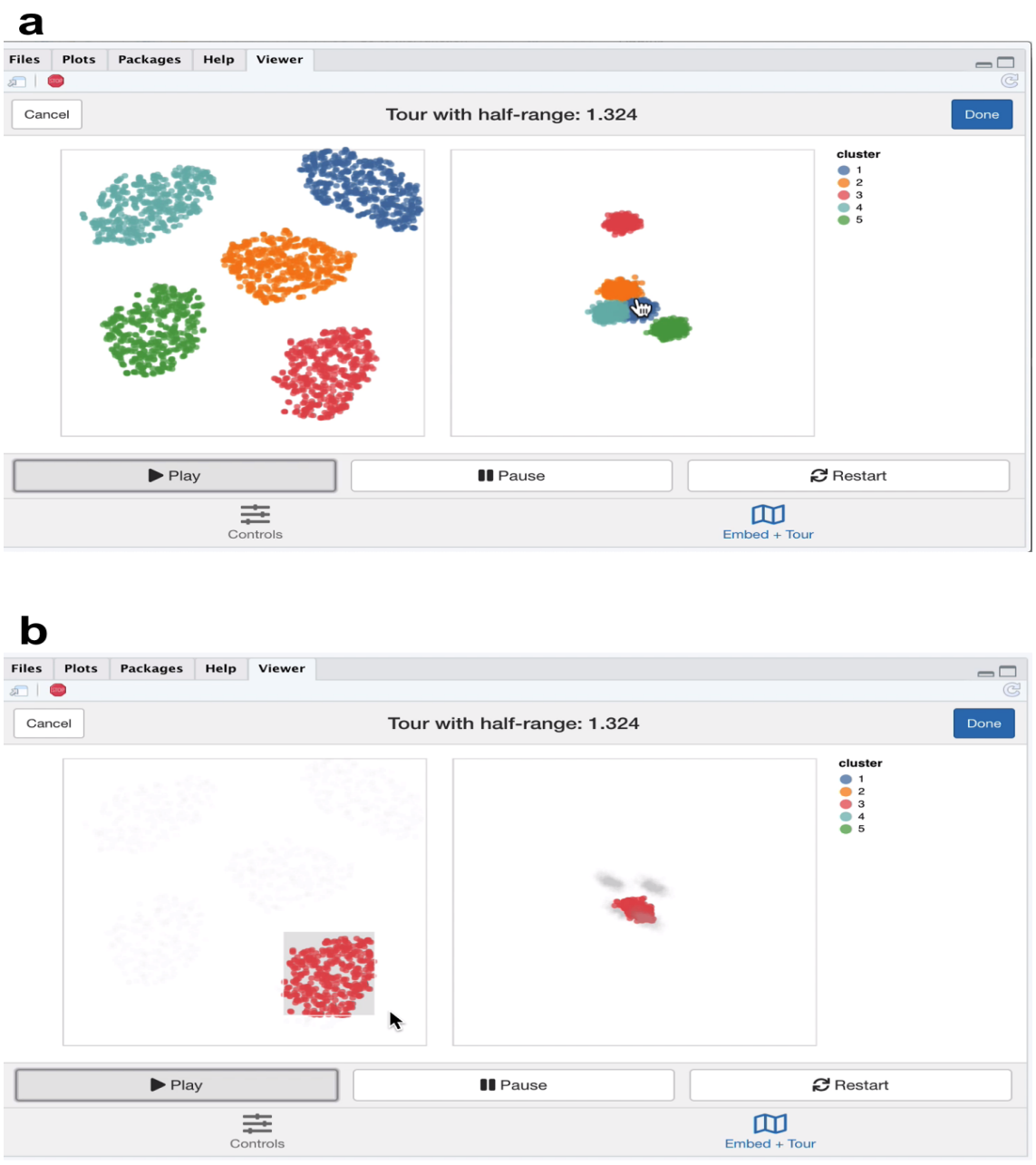


Figure 5.1: *Screenshots of the* ***liminal*** *interface applied to well clustered data, a video of the tour animation is available at* `https://player.vimeo.com/video/439635921`.

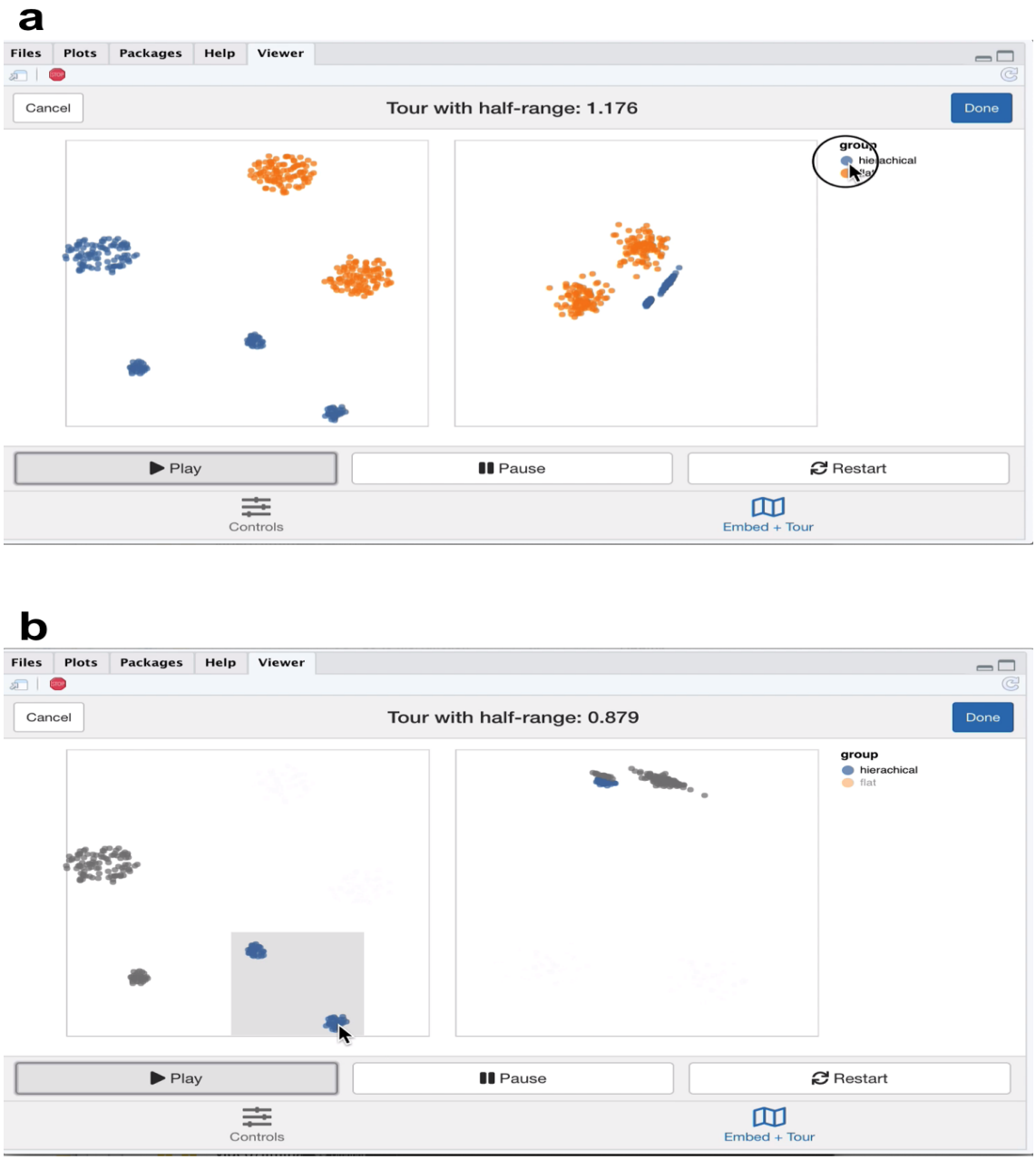


Figure 5.2: *Screenshots of the* ***liminal*** *interface applied to sub-clustered data, a video of the tour animation is available at* `https://player.vimeo.com/video/439635905`.

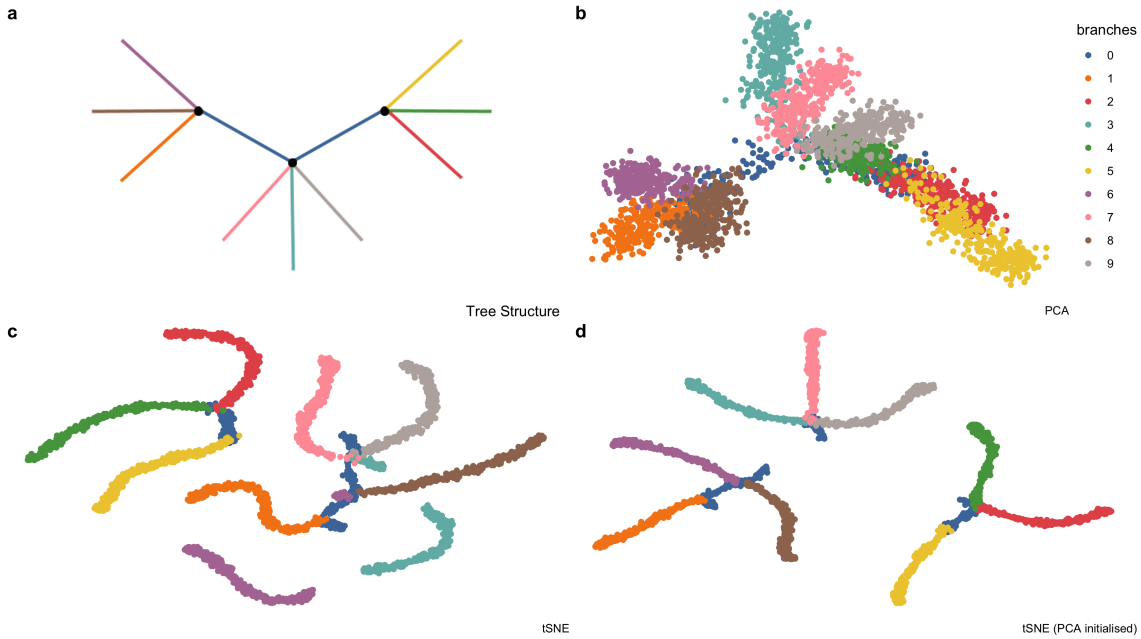


Figure 5.3: *Example high-dimensional tree shaped data, $n = 3000$ and $p = 100$.* (a) *The true data lies on 2-d tree consisting of ten branches. This data is available in the* ***phateR*** *package and is simulated via diffusion-limited aggregation (a random walk along the branches of the tree) with Gaussian noise added (Moon et al., 2019).* (b) *The first two principal components, which form the initial projection for the tour, note that the backbone of the tree is obscured by this view.* (c) *The default t-SNE view breaks the global structure of the tree.* (d) *Altering t-SNE using the first two principal components as the starting coordinates for the embedding, results in clustering the tree at its branching points.*

5.5.3 Case Study 3: Exploring data with piecewise linear structure

Next we explore some simulated noisy tree structured data (Figure 5.3). Our interest here is how t-SNE visualisations break topology of the data, and then seeing if we can resolve this by tweaking the default parameters with reference to the global view of the data set. This simulation aims to mimic branching trajectories of cell differentiation: if there were only mature cells, we would just see the tips of the branches which have a hierarchical pattern of clustering.

First, we apply principal components and restrict the results down to the first twelve principal components (which makes up approximately 70% of the variance explained in the data) to use with the grand tour.

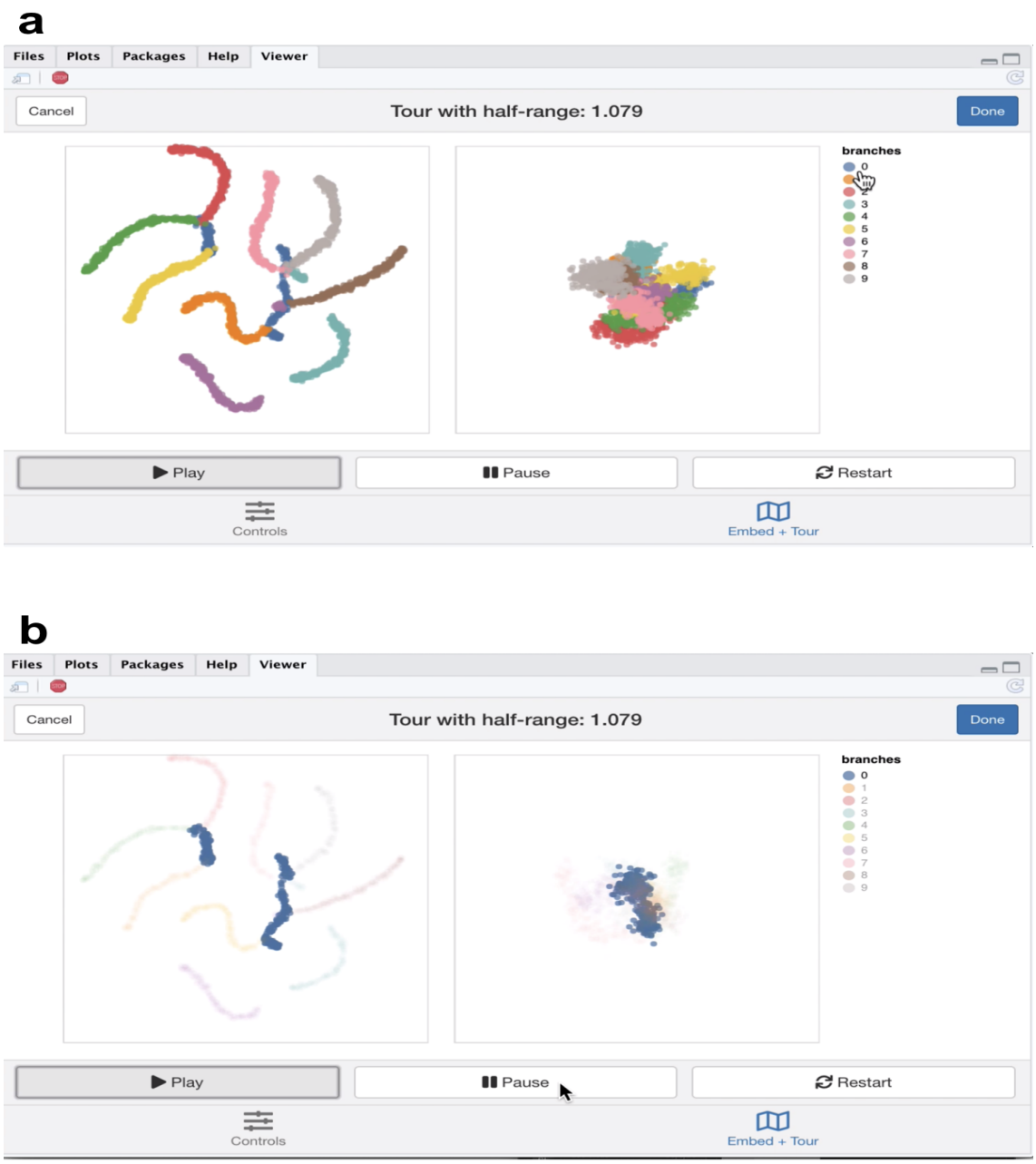


Figure 5.4: *Screenshots of the **liminal** interface applied to tree structured data, a video of the tour animation is available at* `https://player.vimeo.com/video/439635892`.

Moreover, we run t-SNE using the default arguments on the complete data (this keeps the first 50 PCs, sets the perplexity to equal 30 and performs random initialisation). We then create a linked tour with t-SNE layout with **liminal** as shown in Figure 5.4.

From the linked video, we see that the t-SNE view has been unable to recapitulate the topology of the tree - the backbone (blue) branch has been split into three fragments

(Figure 5.4a). We can see this immediately via the linked highlighting over both plots. If we click on the legend for the zero branch, the blue coloured points on each view are highlighted and the remaining points are made transparent. From, here it becomes apparent from the tour view that the blue branch forms the backbone of the tree and is connected to all other branches. From the video it is easy to see that cluster sizes formed via t-SNE can be misleading; from the tour view there is a lot of noise along the branches, while this does not appear to be the case for the t-SNE result (Figure 5.4b).

From the first view, we modify the inputs to the t-SNE view, to try and produce a better trade-off between local structure and retain the topology of the data. We keep every parameter the same except that we initialise Y with the first two PCs (scaled to have standard deviation 1e-4) instead of the default random initialisation and increase the perplexity from 30 to 100. We then combine these results with our tour view as displayed in the linked video in the caption of Figure 5.5.

The video linked in Figure 5.5 shows that this selection of parameters results in the tips of the branches (the three black dots in Figure 5.3a) being split into three clusters representing the terminal branches of the tree. However, there are perceptual issues following the placement of the three groupings on the t-SNE view that become apparent via simple linked brushing. If we brush the tips of the yellow and brown branches (which appear to be close to each other on the t-SNE view), we immediately see the placement is distorted in the t-SNE view, and in the tour view these tips are at opposite ends of the tree (Figure 5.5b). Although, this is a known issue of the t-SNE algorithm, we can easily identify it via simple interactivity without knowing the inner workings of the method.

5.5.4 Case Study 4: Clustering single cell RNA-seq data

A common analysis task in single cell studies is performing clustering to identify groupings of cells with similar expression profiles. Analysts in this area generally use non linear DR methods for verification and identification of clusters and developmental trajectories (i.e., case study 1). For clustering workflows the primary task is verify the existence of clusters and then begin to identify the clusters as cell types using the expression of "known" marker genes. Here a 'faithful' a embedding should ideally preserve the topology of

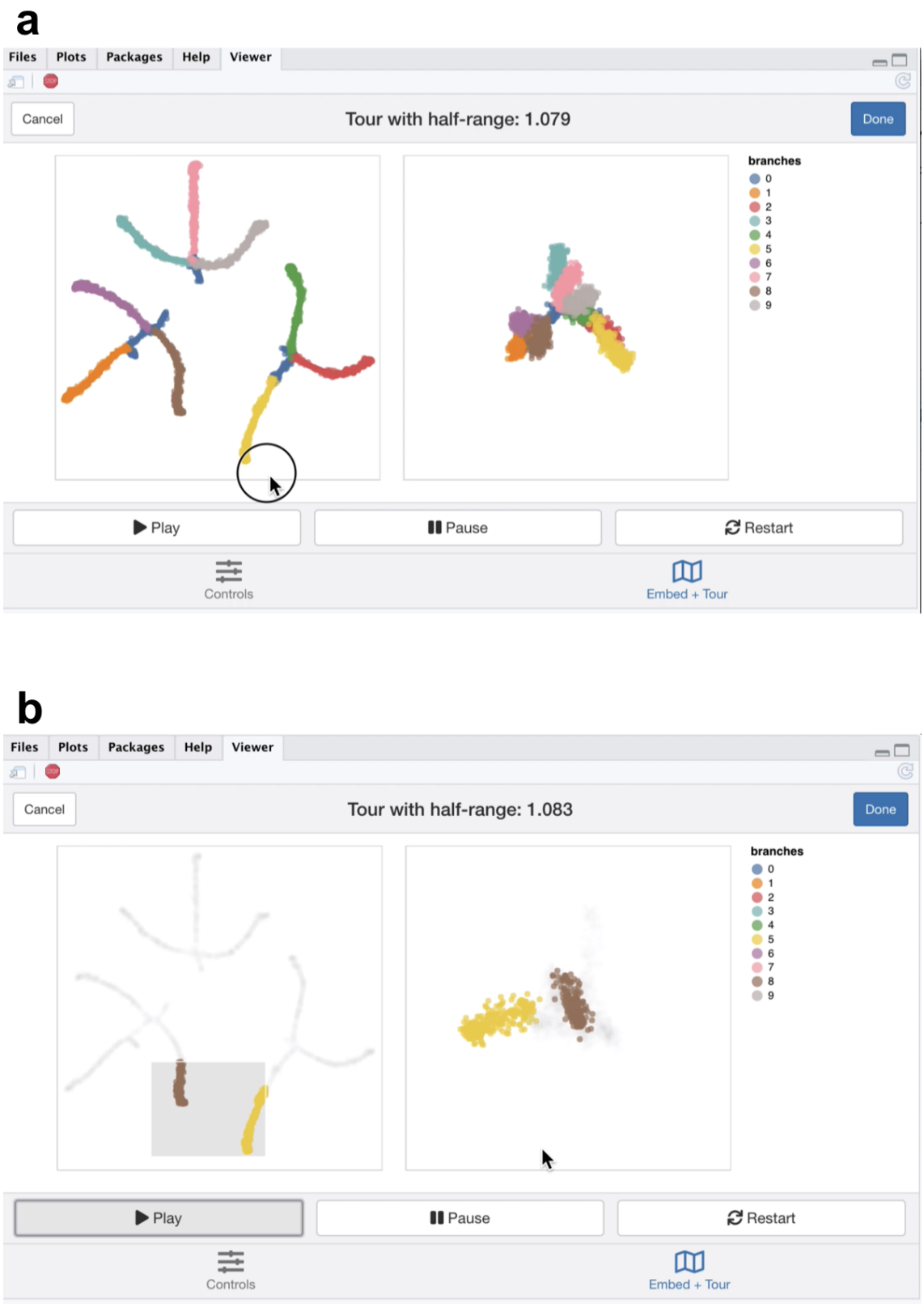


Figure 5.5: *Screenshots of the **liminal** interface applied to tree structured data, a video of the tour animation is available at* https://player.vimeo.com/video/439635863.

the data; cells that correspond to a cell type should lie in the same neighbourhood in high-dimensional space. In this case study we use our linked brushing approaches to look at neighbourhood preservation and look at marker genes through the lens of the tour. The data we have selected for this case study has features similar to those found in case studies 2 and 3.

First, we downloaded the raw mouse retinal single cell RNA-seq data from Macosko et al. (2015) using the **scRNAseq** Bioconductor package (Risso and Cole, 2019). We have followed a standard workflow for pre-processing and normalizing this data (described by Amezquita et al. (2020)): we performed QC using the **scater** package by removing cells with high proportion of mitochondrial gene expression and low numbers of genes detected, we log-transformed and normalised the expression values and finally selected highly variable genes (HVGs) using **scran** (McCarthy et al., 2017; Lun, McCarthy, and Marioni, 2016). The top ten percent of HVGs were used to subset the normalised expression matrix and compute PCA using the first 25 components. Using the PCs we built a shared nearest neighbours graph (with $k = 10$) and used Louvain clustering to generate clusters (Blondel et al., 2008).

To check and verify the clustering we construct a **liminal** view. We tour the first five PCs (approximately 20% of the variance in expression), alongside the t-SNE view which was computed from all 25 PCs. We have selected only the first five PCs because there is a large drop in the percentage of variance explained after the fifth component, with each component after contributing less than one percent of variance. Consequently, increasing the number of PCs to tour would increase the dimensionality and volume of the subspace we are touring but without adding any additional signal to the view.

Due to latency of the **liminal** interface we do a weighted sample of the rows based on cluster membership, leaving us with approximately 10 per cent of the original data size - 4,590 cells. Although this is not ideal, it still allows us to get a sense of the shape of the clusters as seen from the linked video in Figure 5.6. If one was interested in performing more in-depth cluster analysis we recommend an iterative approach of removing large clusters and then re-running the **liminal** view as a way finding more granular cluster structure.

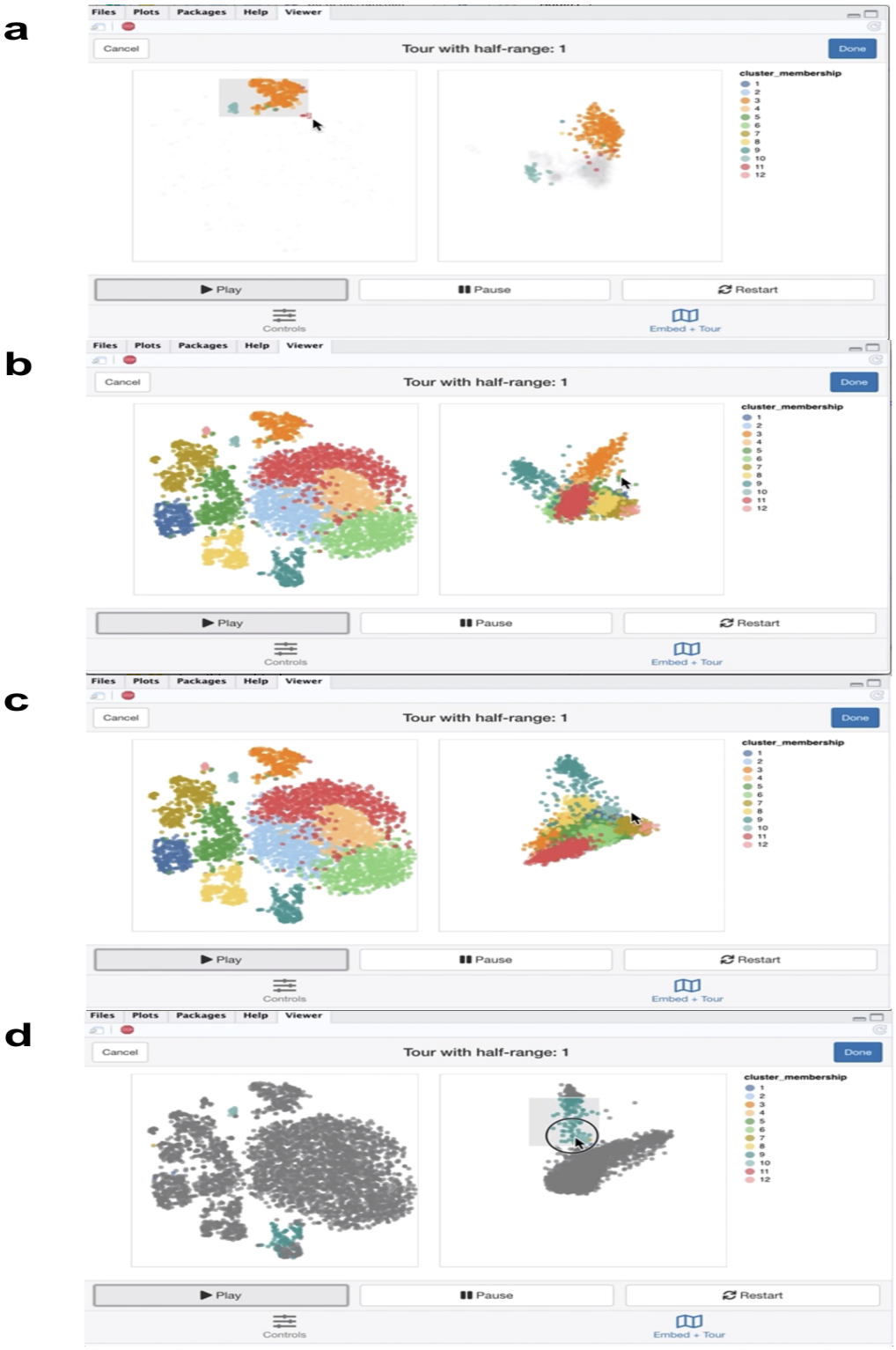


Figure 5.6: *Screenshots of the* ***liminal*** *interface applied to single cell data, a video of the tour animation is available at* `https://player.vimeo.com/video/439635812`.

From the video linked in Figure 5.6, we learn that the embedding has mostly captured the clusters relative location to each other to their location in high dimensional space, with a notable exception of points in cluster 3 and 10 as shown with linked brushing (Figure 5.6a). As expected, t-SNE mitigates the crowding problem that is an issue for tour in this case, where points in clusters 2,4,6, and 11 are clumped together in tour view, but are blown up in the embedding view (Figure 5.6b). The tour appears to form a tetrahedron like shape, with points lying on surface and along the vertices of the tetrahedron in 5-d PCA space - a phenomena that has also been observed in Korem et al. (2015) (Figure 5.6c). Brushing on the tour view, reveals points in cluster 9 that are diffuse in the embedding view, points in cluster 9 are relatively far away and spread apart from other clusters in the tour view, but has points placed in cluster 3 and 9 in the embedding (Figure 5.6d).

Next, we identify marker genes for clusters using one sided Welch t-tests with a minimum log fold change of one as recommended by Amezquita et al. (2020), which uses the testing framework from McCarthy and Smyth (2009). We select the top 10 marker genes that are upregulated in cluster 2, which was one of the clumped clusters when we toured on principal components. Here the tour becomes an alternative to a standard heatmap view for assessing shared markers; the basis generation (shown as the biplot on the left view) reflects the relative weighting of each gene. We run the tour directly on the log normalised expression values using the same subset as before.

From the video linked in Figure 5.7, we see that the expression of the marker genes, appear to separate the previously clumped clusters 2,4,6, and 11 from the other clusters, indicating that these genes are expressed in all four clusters (Figure 5.7a). After zooming, we can see a trajectory forming along the clusters, while the axis view shows that magnitude of expression in the marker genes is similar across these separated clusters which is consistent with the results of marker gene analysis (Figure 5.7b).

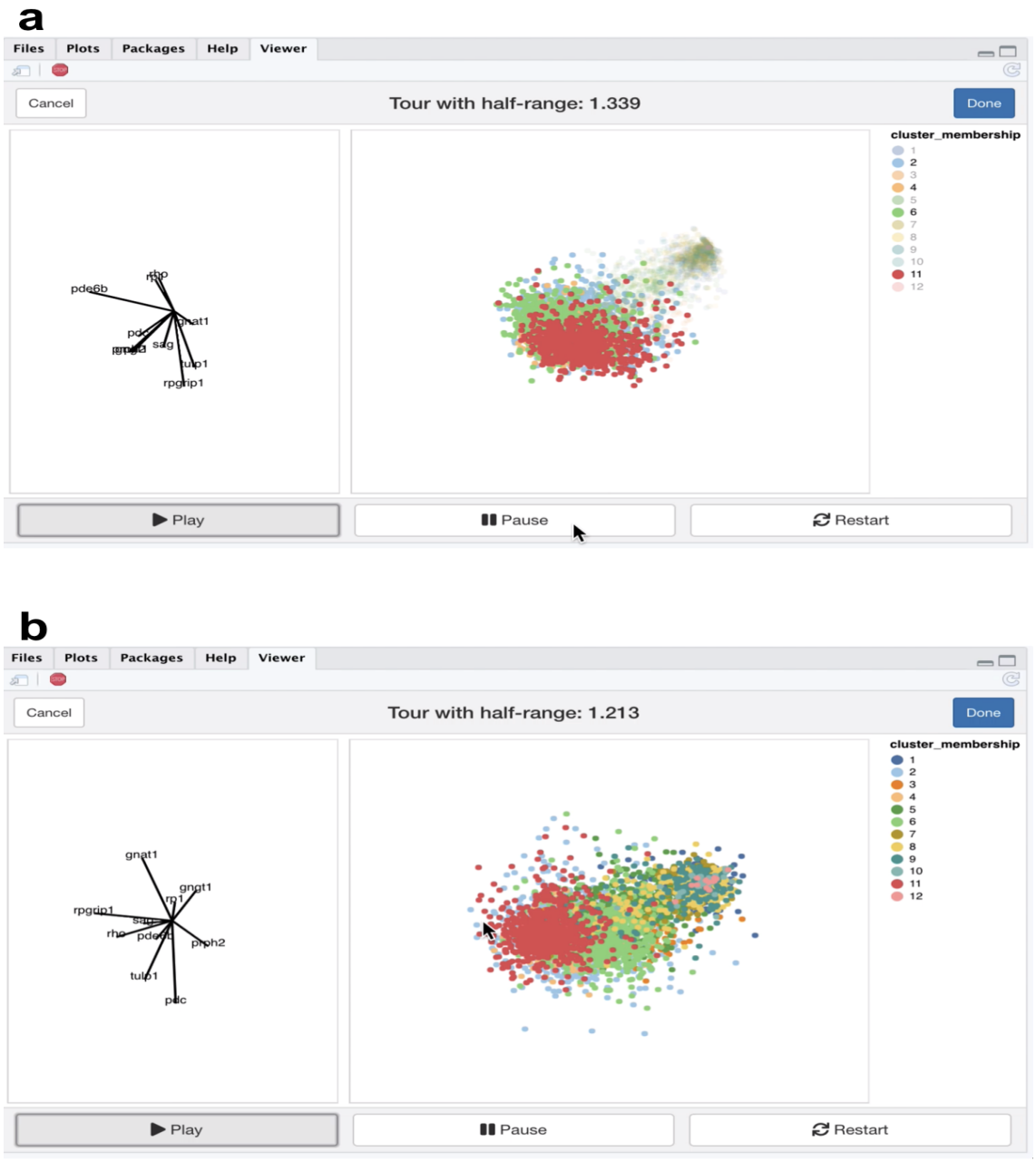


Figure 5.7: *Screenshots of the* ***liminal*** *tour applied to a marker gene set, a video of the tour animation is available at* https://player.vimeo.com/video/439635843.

5.6 Discussion

We have shown that the use of tours as a tool for interacting with high dimensional data provides an additional insight for interrogating views generated from embeddings. The interface we have designed in the **liminal** package, allows a user to gain a deeper understanding of an embedding algorithm, and rectifies perceptual issues associated with NLDR methods via linked interactions with the tour. As we have shown in the simulation case studies, the t-SNE method can produce misleading embeddings which can be detected through the use linked brushing and highlighting. In the case when the data has a piecewise linear geometry, like the tree simulation, the tour preserves the shape of the data which can be obscured by the embedding method.

Our framework can also be useful in practice, as displayed in the fourth case study. The tour when combined with t-SNE allowed us to identify clusters, while giving us an idea of their orientation to each other. Moreover, we could visually inspect the separation of clusters using a tour on marker gene sets. We see our approach as being valuable to the single cell analyst who wants to make their embeddings more interpretable.

We have shown in the case studies, that one to one linked brushing can be used to identify distortions in the embedding, however we would like extend this to one to many linked brushing, which would allow us to directly interrogate neighbourhood preservation. This form of brushing acts directly on a k-nearest neighbours (k-nn) graph computed from a reference dataset: when a user brushes over a region in the embedding, all the points that match the graphs edges are selected on the corresponding tour view. The reference data set for computing nearest neighbours (for example a distance matrix, or the complete data matrix) can be independent of the tour or embedding views. In place of highlighting, one could use opacity or binned colour scales to encode distances or ranks instead of the neighbouring points. We have begun implementing this brush in **liminal**, using the **FNN** or **RcppAnnoy** packages for fast neighbourhood estimation on the server side, however there are still technicalities that need be resolved (Beygelzimer et al., 2019; Eddelbuettel, 2020). Brush composition, such as 'and', 'or', or 'not' brushes, could be used to further

investigate mismatches between the k-nn graphs estimated from both the embedding and tour views.

There are some limitations in using the **liminal** interface for larger datasets. First, t-SNE avoids the crowding problem, points are separated into distinct regions on the display canvas, while for the tour, points are concentrated in the centre of the projection and become difficult to see. We have recently proposed a simple non-linear transformation for the tour called a sage tour that aims to fix this problem (Laa, Cook, and Lee, 2020). Second, as n increases both the embedding view and tour view become harder to read due to over-plotting, while the interactivity and animation become slower as there is more data passing from the server to the client. For the tasks we have looked at in this paper, where shape and density are important to the analyst, we think that better displays and sub-sampling strategies are more useful than being able to look at every single point on the canvas. We showed in our single cell clustering case study that doing a weighted sample based on cluster membership still allowed us to get a sense of relative cluster orientation, however there are alternative sampling approaches that could be applied, like selecting points close to the cluster centres. Alternative displays via statistical transformations could also mitigate the need to show all of the data. Recent work by Laa et al. (2020) is a promising area for further investigation, as well as work from topological statistics (Rieck, 2017; Genovese et al., 2017).

Acknowledgements

We would like to thank Dr David Frazier, Dr Ursula Laa and Dr Paul Harrison for their feedback on this work. The screenshots and images were compiled using the **cowplot** and **ggplot2** packages (Wilke, 2019; Wickham, 2016).

Supplementary Materials

Code, data, and video for reproducing this paper are available at `https://github.com/sa-lee/paper-liminal`. Direct links to videos for viewing online are available in table 5.1.

Table 5.1: *Case Study Videos*

Case Study	Example	URL
1	gaussian	https://player.vimeo.com/video/439635921
2	hierarchical	https://player.vimeo.com/video/439635905
3	trees-01	https://player.vimeo.com/video/439635892
3	trees-02	https://player.vimeo.com/video/439635863
4	mouse-01	https://player.vimeo.com/video/439635812
4	mouse-02	https://player.vimeo.com/video/439635843

Chapter 6

Conclusion

In this thesis, I have designed tools to explore workflow steps that are integral to modern biological data science. In particular, I have implemented software that facilitates the wrangling, integration, and visualisation of high-throughput biological data in a principled and pragmatic manner. The early chapters of this thesis explored the *tidy data* semantic and its extension to range based genomics data. This culminated in the development of "plyranges: a grammar of genomic data transformation" in Chapter 2, which developed a new domain specific language for genomics data analysis. The applicability of the **plyranges** interface and use of the *tidy data* concept were further interrogated in Chapter 3, "Fluent genomics with plyranges and tximeta", which described techniques integrating data along the genome, and emphasised the importance of interoperability between analysis tools. Similarly, Chapter 4, "Exploratory coverage analysis with superintronic and plyranges", tackled data integration from a different angle by looking at multiple summaries of variables measured along the genome to find putative regions of intron retention. In the final part of the thesis I moved towards visualisation issues as they related to working with high-dimensional data common in biological data science. Chapter 5, "Casting multiple shadows: high-dimensional interactive data visualisation with tours and embeddings", explored pragmatic approaches to high dimensional data visualisation in light of the rise of popular non-linear embedding methods.

A significant amount of my work has been devoted to the development of open source R packages and workflows: **plyranges**, **fluentGenomics**, **superintronic** and **liminal**. I have emphasised how coherent software packages are tools for thought; they enable analysts to reason about their data and models through the composition of workflows. To finish, I will discuss the implications of this work and provide suggestions for further research.

6.1 Software Development

The **plyranges** package develops a suite of verbs for interacting with genomic data as a *GRanges* object. Since its release on Bioconductor, it has been relatively successful: it has been downloaded 24,722 times from 13,178 unique IP addresses. I have also had the privilege of teaching workshops on **plyranges** at Bioconductor conferences which also led to the development of the **fluentGenomics** workflow package, outlined in Chapter 3. A broader impact of the work, has been the discussions around the concepts of fluent interfaces and tidy data within the Bioconductor community, which has led to several developments currently in place that are exploring different approaches for fluent interfaces for other types of omics data. The **plyranges** package is available to download from `https://bioconductor.org/packages/plyranges` and the **fluentGenomics** workflow is available to download from `https://bioconductor.org/packages/release/workflows/html/fluentGenomics.html`.

The **superintronic** software described in Chapter 4 has been used in Lee et al. (2020) to disentangle and view intron signal in RNA-seq data. Here, we again show the strengths of providing a long-form representations of genomics data (in this case coverage vectors). By leveraging **plyranges** we were then able to create a set of data descriptors that we could link back to the raw data to discover genes thought to be associated with a real biological signal. An interesting extension to this work would be applying it to single cell and long-read based transcriptomics data, where scalability and much larger design matrices would become an issue. The **superintronic** package is available to download from `https://github.com/sa-lee/superintronic`.

Finally, the **liminal** software aims to provide a more holistic approach to analysis tasks requiring the use of dimensionality reduction algorithms. We showed how to incorporate

interactive graphics and tours to identify problems with embeddings. Based on the case studies provided I believe that the methods used in **liminal** could be broadly applicable to many high dimensional datasets and NLDR methods. The **liminal** package is available to download from `https://github.com/sa-lee/liminal`.

6.2 Further Work

A limitation of the grammar as we have implemented it in **plyranges** is lack of scalability and computational speed for data sets that do not fit in memory. We attempted several techniques for performing delayed operations over range-based data, however a more general approach that allows for data stored on the cloud or in scientific data formats like HDF5 that leverage existing Bioconductor frameworks would be useful. We showed in chapters 3 ad 4 that an analyst is able to do some very complex data transformations and re-sampling procedures via casting results into *GRanges* object. However, it is unclear whether the semantics of our grammar can be extended to data that can not be efficiently reshaped into long form tidy representations. Moreover, further work is required to explore the design space of grammars for data transformations and grammars for graphics when the data are large, multifaceted and non-rectangular.

We showed in Chapter 5 that tours provide a global overview that can be used as tool for exploring model fits. An issue that arises is how to scale the tour as the number of observations increases. There are latencies in sending data from the back end to the visualisation client that causes lag during animation. One could also question whether point based displays are appropriate in this case, and it would be worth exploring the usability of animations based on binning the projections. Moreover, when the number of observations are large, the points in the projections are concentrated in the centre of the tour display obscuring interesting aspects of the data. This is mitigated via having the ability to zoom, but further research into transforming the projections to avoid crowding would be valuable. An added complexity to changes in visual displays are thinking about the design of user interactions, and several promising avenues based on section tours could be explored (Laa et al., 2020; Laa, Cook, and Valencia, 2020).

Appendix A

plyranges vignette

Appendix B

Getting started with the plyranges package

B.1 *Ranges* revisited

In Bioconductor there are two classes, `IRanges` and `GRanges`, that are standard data structures for representing genomics data. Throughout this document I refer to either of these classes as *Ranges* if an operation can be performed on either class, otherwise I explicitly mention if a function is appropriate for an *IRanges* or *GRanges*.

Ranges objects can either represent sets of integers as *IRanges* (which have start, end and width attributes) or represent genomic intervals (which have additional attributes, sequence name, and strand) as *GRanges*. In addition, both types of *Ranges* can store information about their intervals as metadata columns (for example GC content over a genomic interval).

Ranges objects follow the tidy data principle: each row of a *Ranges* object corresponds to an interval, while each column will represent a variable about that interval, and generally each object will represent a single unit of observation (like gene annotations).

Consequently, *Ranges* objects provide a powerful representation for reasoning about genomic data. In this vignette, you will learn more about *Ranges* objects and how via grouping, restriction and aggregation you can perform common data tasks.

B.2 Constructing *Ranges*

To construct an *IRanges* we require that there are at least two columns that represent at either a starting coordinate, finishing coordinate or the width of the interval.

```
#> IRanges object with 7 ranges and 0 metadata columns:
#>           start       end     width
#>       <integer> <integer> <integer>
#>   [1]         2         1         0
#>   [2]         1         1         1
#>   [3]         0         1         2
#>   [4]        -1         1         3
#>   [5]        13        14         2
#>   [6]        14        14         1
#>   [7]        15        14         0
```

To construct a *GRanges* we require a column that represents that sequence name (contig or chromosome id), and an optional column to represent the strandedness of an interval.

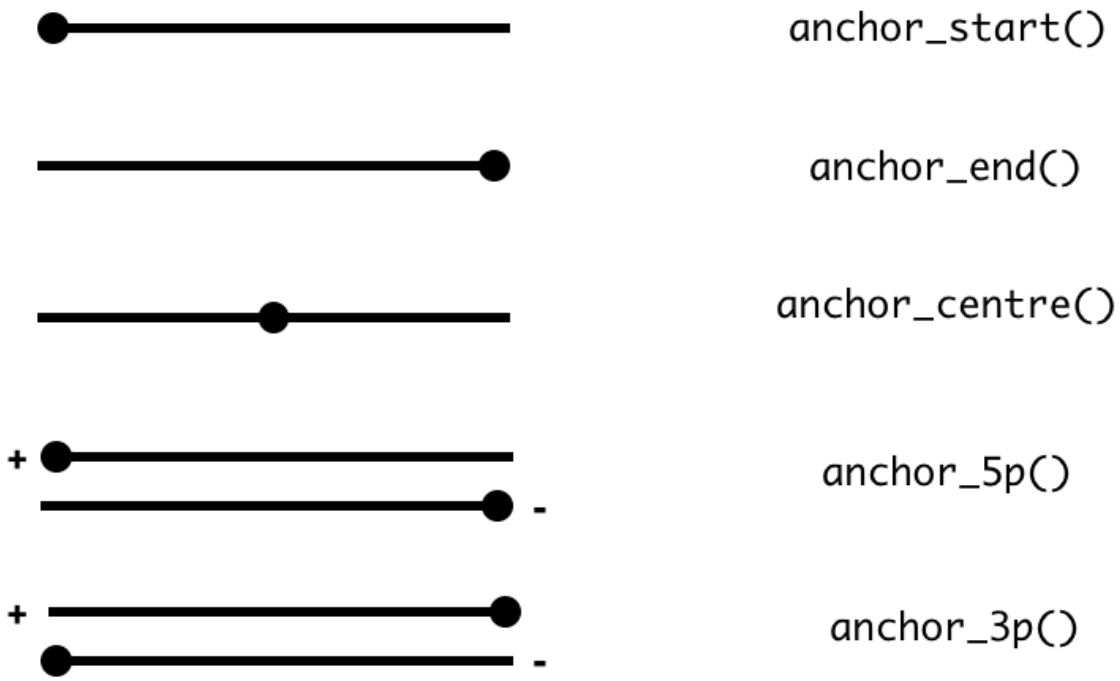
```
#> GRanges object with 7 ranges and 1 metadata column:
#>       seqnames    ranges strand |        gc
#>          <Rle> <IRanges>  <Rle> | <numeric>
#>   [1]     chr2       2-1      - |  0.762551
#>   [2]     chr1         1      - |  0.669022
#>   [3]     chr2       0-1      + |  0.204612
#>   [4]     chr2      -1-1      - |  0.357525
#>   [5]     chr1     13-14      - |  0.359475
#>   [6]     chr1        14      - |  0.690291
```

```
#>   [7]     chr2     15-14      - |  0.535811
#>   -------
#>   seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

B.3 Arithmetic on Ranges

Sometimes you want to modify a genomic interval by altering the width of the interval while leaving the start, end or midpoint of the coordinates unaltered. This is achieved with the `mutate` verb along with `anchor_*` adverbs.

The act of anchoring fixes either the start, end, center coordinates of the `Range` object, as shown in the figure and code below and anchors are used in combination with either `mutate` or `stretch`.




```
#> IRanges object with 3 ranges and 0 metadata columns:
#>           start       end     width
#>       <integer> <integer> <integer>
#>   [1]         1        10        10
#>   [2]         2        11        10
#>   [3]         3        12        10
```

```
#> IRanges object with 3 ranges and 0 metadata columns:
#>           start       end     width
#>       <integer> <integer> <integer>
#>   [1]         1        10        10
#>   [2]         2        11        10
#>   [3]         3        12        10

#> IRanges object with 3 ranges and 0 metadata columns:
#>           start       end     width
#>       <integer> <integer> <integer>
#>   [1]        -4         5        10
#>   [2]        -7         2        10
#>   [3]        -1         8        10

#> IRanges object with 3 ranges and 0 metadata columns:
#>           start       end     width
#>       <integer> <integer> <integer>
#>   [1]        -2         7        10
#>   [2]        -3         6        10
#>   [3]         1        10        10

#> GRanges object with 3 ranges and 0 metadata columns:
#>       seqnames    ranges strand
#>          <Rle> <IRanges>  <Rle>
#>   [1]     seq1      -4-5      +
#>   [2]     seq1      -7-2      *
#>   [3]     seq1      3-12      -
#>   -------
#>   seqinfo: 1 sequence from an unspecified genome; no seqlengths

#> GRanges object with 3 ranges and 0 metadata columns:
#>       seqnames    ranges strand
```

```
#>          <Rle> <IRanges>  <Rle>
#>   [1]     seq1      1-10      +
#>   [2]     seq1      2-11      *
#>   [3]     seq1      -1-8      -
#>   -------
#>   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

Similarly, you can modify the width of an interval using the `stretch` verb. Without anchoring, this function will extend the interval in either direction by an integer amount. With anchoring, either the start, end or midpoint are preserved.

```
#> IRanges object with 3 ranges and 0 metadata columns:
#>           start       end     width
#>       <integer> <integer> <integer>
#>   [1]        -4        10        15
#>   [2]        -3         7        11
#>   [3]        -2        13        16

#> IRanges object with 3 ranges and 0 metadata columns:
#>           start       end     width
#>       <integer> <integer> <integer>
#>   [1]       -14        10        25
#>   [2]       -13         7        21
#>   [3]       -12        13        26

#> IRanges object with 3 ranges and 0 metadata columns:
#>           start       end     width
#>       <integer> <integer> <integer>
#>   [1]        -4        20        25
#>   [2]        -3        17        21
#>   [3]        -2        23        26
```

```
#> GRanges object with 3 ranges and 0 metadata columns:
#>       seqnames    ranges strand
#>          <Rle> <IRanges>  <Rle>
#>   [1]     seq1      -9-5      +
#>   [2]     seq1      -8-2      *
#>   [3]     seq1      3-18      -
#>   -------
#>   seqinfo: 1 sequence from an unspecified genome; no seqlengths

#> GRanges object with 3 ranges and 0 metadata columns:
#>       seqnames    ranges strand
#>          <Rle> <IRanges>  <Rle>
#>   [1]     seq1      1-15      +
#>   [2]     seq1      2-12      *
#>   [3]     seq1      -7-8      -
#>   -------
#>   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

Ranges can be shifted left or right. If strand information is available we can also shift upstream or downstream.

```
#> IRanges object with 3 ranges and 0 metadata columns:
#>           start       end     width
#>       <integer> <integer> <integer>
#>   [1]        -9        -5         5
#>   [2]        -8        -8         1
#>   [3]        -7        -2         6

#> IRanges object with 3 ranges and 0 metadata columns:
#>           start       end     width
#>       <integer> <integer> <integer>
#>   [1]        11        15         5
```

```
#>   [2]        12        12         1
#>   [3]        13        18         6

#> GRanges object with 3 ranges and 0 metadata columns:
#>       seqnames    ranges strand
#>          <Rle> <IRanges>  <Rle>
#>   [1]     seq1     -9--5      +
#>   [2]     seq1        -8      *
#>   [3]     seq1     13-18      -
#>   -------
#>   seqinfo: 1 sequence from an unspecified genome; no seqlengths

#> GRanges object with 3 ranges and 0 metadata columns:
#>       seqnames    ranges strand
#>          <Rle> <IRanges>  <Rle>
#>   [1]     seq1     11-15      +
#>   [2]     seq1        12      *
#>   [3]     seq1     -7--2      -
#>   -------
#>   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

B.4 Grouping *Ranges*

plyranges introduces a new class of *Ranges* called `RangesGrouped`, this is a similar idea to the grouped `data.frame\tibble` in **dplyr**.

Grouping can act on either the core components or the metadata columns of a *Ranges* object.

It is most effective when combined with other verbs such as `mutate()`, `summarise()`, `filter()`, `reduce_ranges()` or `disjoin_ranges()`.

```
#> GRanges object with 7 ranges and 1 metadata column:
#> Groups: strand [2]
```

```
#>       seqnames    ranges strand |        gc
#>          <Rle> <IRanges>  <Rle> | <numeric>
#>   [1]     chr2      1-10      - |  0.889454
#>   [2]     chr2      2-11      + |  0.180407
#>   [3]     chr1      3-12      - |  0.629391
#>   [4]     chr2      4-13      + |  0.989564
#>   [5]     chr1      5-14      - |  0.130289
#>   [6]     chr1      6-15      - |  0.330661
#>   [7]     chr2      7-16      - |  0.865121
#>   -------
#>   seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

B.5 Restricting *Ranges*

The verb `filter` can be used to restrict rows in the *Ranges*. Note that grouping will cause the `filter` to act within each group of the data.

```
#> GRanges object with 2 ranges and 1 metadata column:
#>       seqnames    ranges strand |        gc
#>          <Rle> <IRanges>  <Rle> | <numeric>
#>   [1]     chr2      2-11      + |  0.180407
#>   [2]     chr1      5-14      - |  0.130289
#>   -------
#>   seqinfo: 2 sequences from an unspecified genome; no seqlengths

#> GRanges object with 2 ranges and 1 metadata column:
#> Groups: strand [2]
#>       seqnames    ranges strand |        gc
#>          <Rle> <IRanges>  <Rle> | <numeric>
#>   [1]     chr2      1-10      - |  0.889454
#>   [2]     chr2      4-13      + |  0.989564
```

```
#>   -------
#>   seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

We also provide the convenience methods `filter_by_overlaps` and `filter_by_non_overlaps` for restricting by any overlapping *Ranges*.

```
#> IRanges object with 4 ranges and 0 metadata columns:
#>           start       end     width
#>       <integer> <integer> <integer>
#>   [1]         5         9         5
#>   [2]        10        14         5
#>   [3]        15        19         5
#>   [4]        20        24         5

#> IRanges object with 5 ranges and 0 metadata columns:
#>           start       end     width
#>       <integer> <integer> <integer>
#>   [1]         2         4         3
#>   [2]         3         6         4
#>   [3]         4         8         5
#>   [4]         5        10         6
#>   [5]         6        12         7

#> IRanges object with 2 ranges and 0 metadata columns:
#>           start       end     width
#>       <integer> <integer> <integer>
#>   [1]         5         9         5
#>   [2]        10        14         5

#> IRanges object with 2 ranges and 0 metadata columns:
#>           start       end     width
#>       <integer> <integer> <integer>
```

```
#>   [1]        15        19         5
#>   [2]        20        24         5
```

B.6 Summarising *Ranges*

The `summarise` function will return a `DataFrame` because the information required to return a *Ranges* object is lost. It is often most useful to use `summarise()` in combination with the `group_by()` family of functions.

```
#> DataFrame with 2 rows and 2 columns
#>       query        gc
#>   <integer> <numeric>
#> 1         1  0.675555
#> 2         2  0.635795
```

B.7 Joins, or another way at looking at overlaps between *Ranges*

A join acts on two GRanges objects, a query and a subject.

The join operator is relational in the sense that metadata from the query and subject ranges is retained in the joined range. All join operators in the **plyranges** DSL generate a set of hits based on overlap or proximity of ranges and use those hits to merge the two datasets in different ways. There are four supported matching algorithms: *overlap*, *nearest*, *precede*, and *follow*. We can further restrict the matching by whether the query is completely *within* the subject, and adding the *directed* suffix ensures that matching ranges have the same direction (strand).

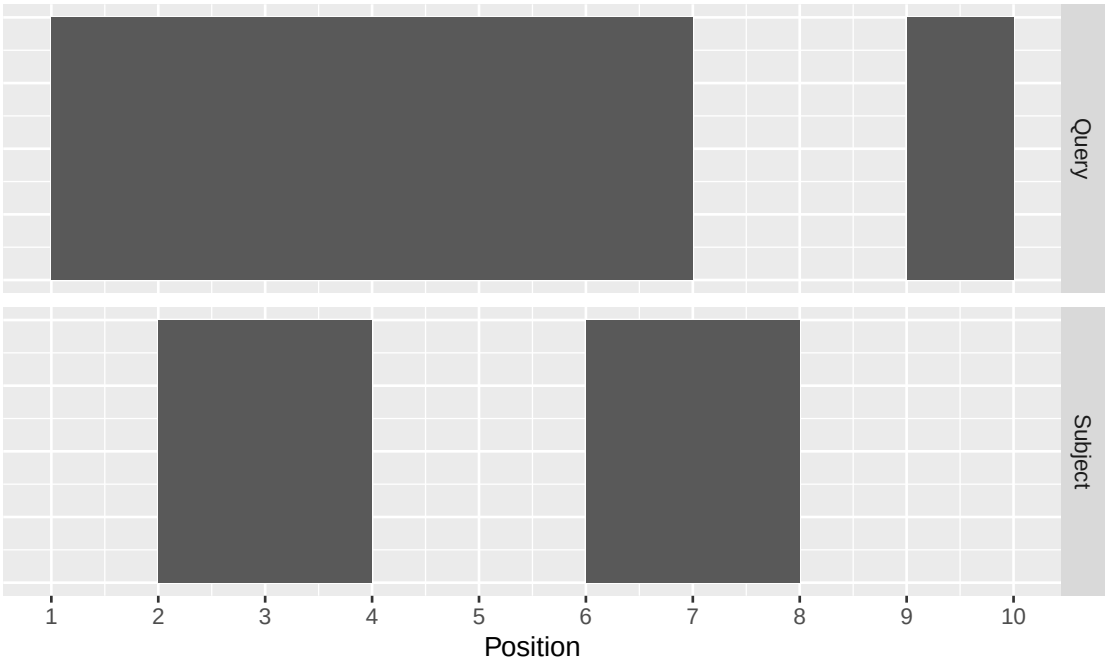
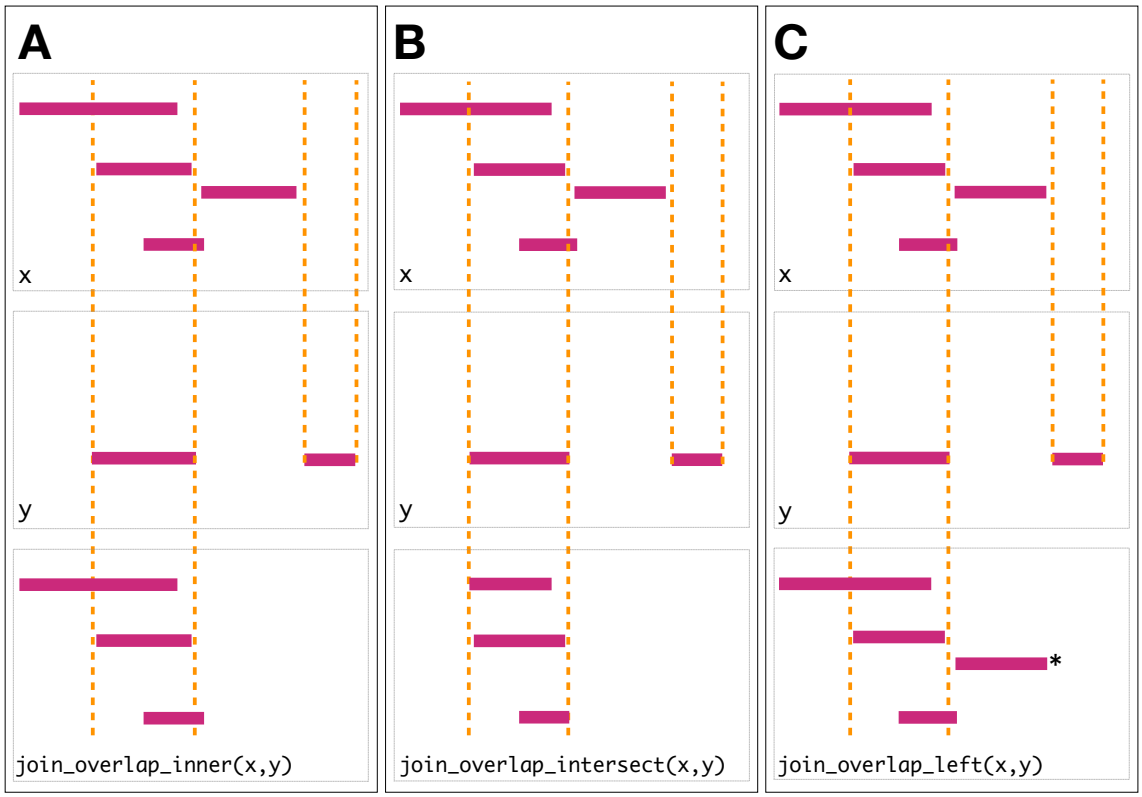


Figure B.1: *Query and Subject Ranges*



The first function, `join_overlap_intersect()` will return a *Ranges* object where the start, end, and width coordinates correspond to the amount of any overlap between the left and

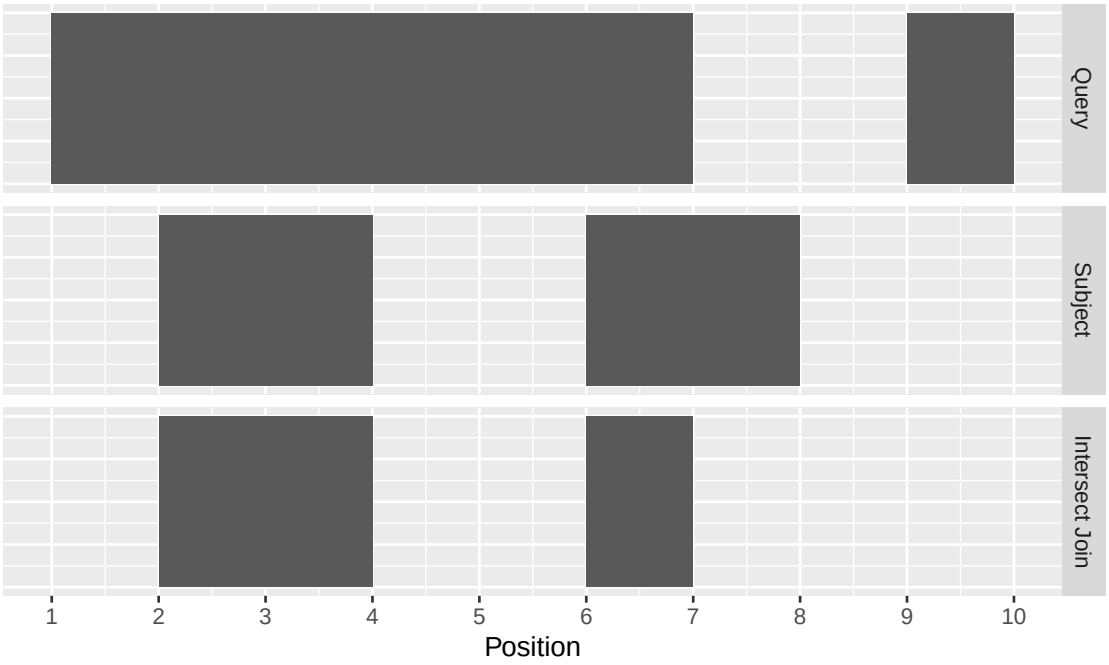


Figure B.2: *Intersect Join*

right input *Ranges*. It also returns any metadata in the subject range if the subject overlaps the query.

```
#> GRanges object with 2 ranges and 2 metadata columns:
#>       seqnames    ranges strand |       key.a       key.b
#>          <Rle> <IRanges>  <Rle> | <character> <character>
#>   [1]     chr1       2-4      + |           a           A
#>   [2]     chr1       6-7      + |           a           B
#>   -------
#>   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

The `join_overlap_inner()` function will return the *Ranges* in the query that overlap any *Ranges* in the subject. Like the `join_overlap_intersect()` function metadata of the subject `Range` is returned if it overlaps the query.

```
#> GRanges object with 2 ranges and 2 metadata columns:
#>       seqnames    ranges strand |       key.a       key.b
#>          <Rle> <IRanges>  <Rle> | <character> <character>
```

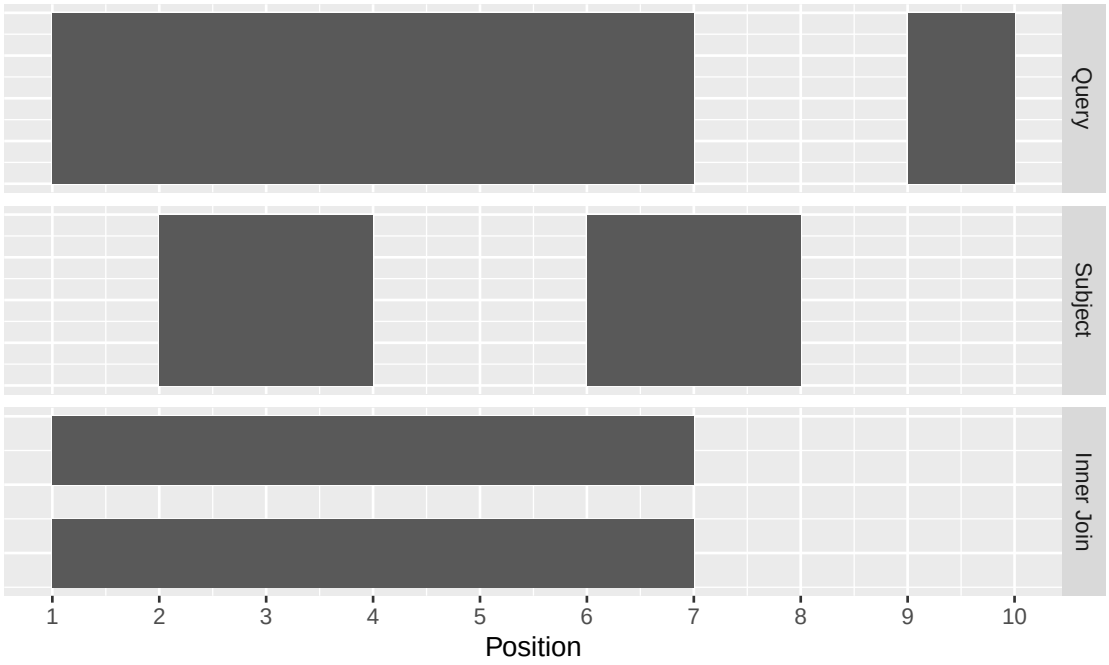


Figure B.3: *Inner Join*

```
#>   [1]     chr1       1-7      + |           a           A
#>   [2]     chr1       1-7      + |           a           B
#>   -------
#>   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

We also provide a convenience method called `find_overlaps` that computes the same result as `join_overlap_inner()`.

```
#> GRanges object with 2 ranges and 2 metadata columns:
#>       seqnames    ranges strand |       key.a       key.b
#>          <Rle> <IRanges>  <Rle> | <character> <character>
#>   [1]     chr1       1-7      + |           a           A
#>   [2]     chr1       1-7      + |           a           B
#>   -------
#>   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

The `join_overlap_left()` method will perform an outer left join.

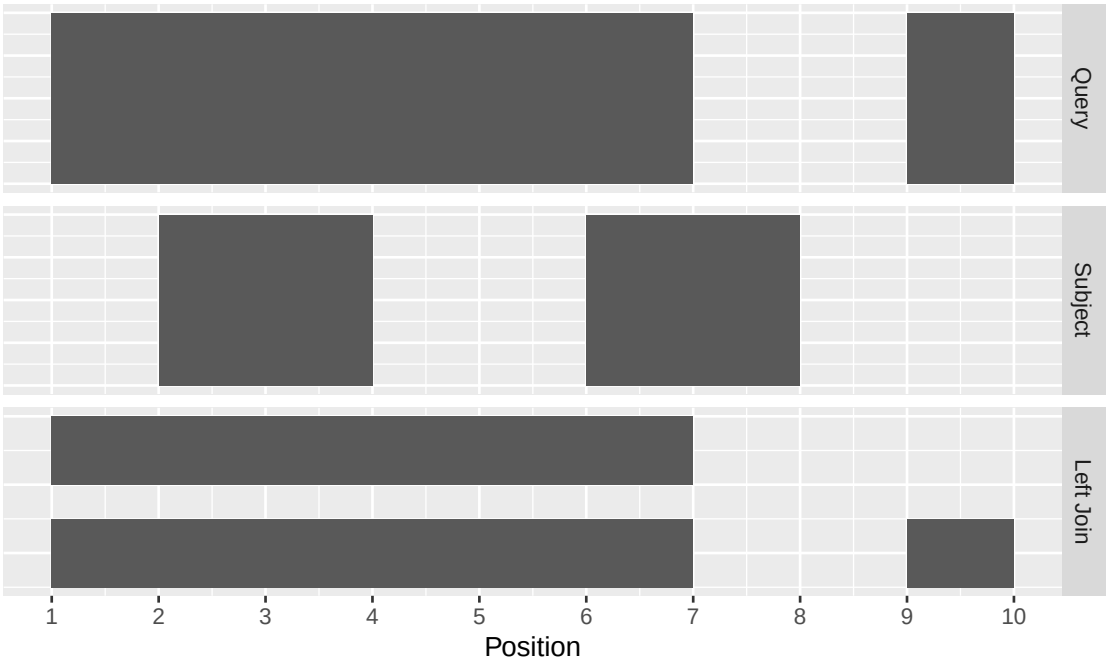


Figure B.4: *Left Join*

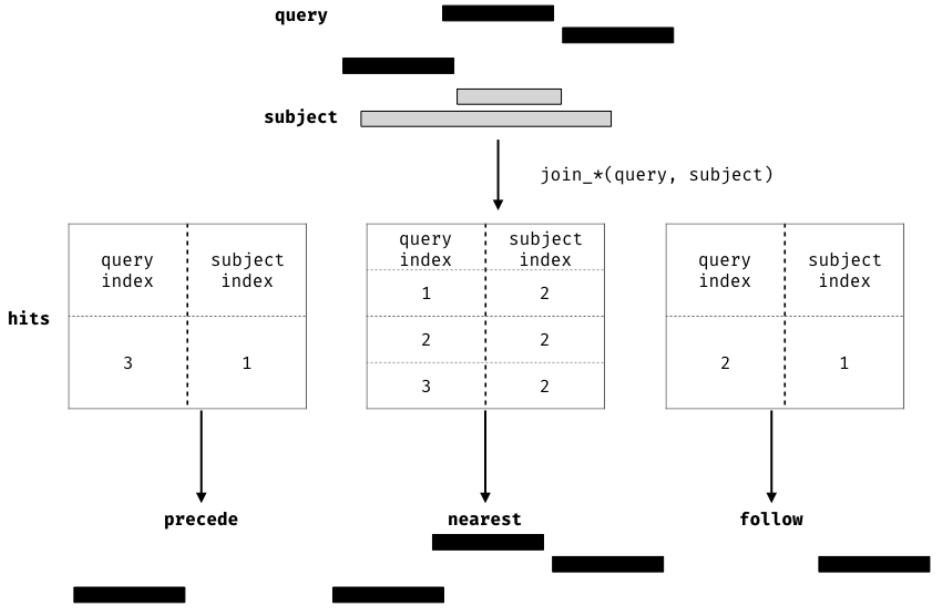
First any overlaps that are found will be returned similar to `join_overlap_inner()`. Then any non-overlapping ranges will be returned, with missing values on the metadata columns.

```
#> GRanges object with 3 ranges and 2 metadata columns:
#>       seqnames    ranges strand |       key.a       key.b
#>          <Rle> <IRanges>  <Rle> | <character> <character>
#>   [1]     chr1       1-7      + |           a           A
#>   [2]     chr1       1-7      + |           a           B
#>   [3]     chr1      9-10      - |           b        <NA>
#>   -------
#>   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

Compared with `filter_by_overlaps()` above, the overlap left join expands the *Ranges* to give information about each interval on the query *Ranges* that overlap those on the subject *Ranges* as well as the intervals on the left that do not overlap any range on the right.

Finding your neighbours

We also provide methods for finding nearest, preceding or following *Ranges*. Conceptually this is identical to our approach for finding overlaps, except the semantics of the join are different.




```
#> IRanges object with 4 ranges and 1 metadata column:
#>           start       end     width |        gc
#>       <integer> <integer> <integer> | <numeric>
#>   [1]         5         9         5 |  0.780359
#>   [2]        10        14         5 |  0.780359
#>   [3]        15        19         5 |  0.780359
#>   [4]        20        24         5 |  0.780359

#> IRanges object with 4 ranges and 1 metadata column:
#>           start       end     width |        gc
#>       <integer> <integer> <integer> | <numeric>
```

```
#>   [1]         5         9         5 |  0.777584
#>   [2]        10        14         5 |  0.603324
#>   [3]        15        19         5 |  0.780359
#>   [4]        20        24         5 |  0.780359

#> IRanges object with 0 ranges and 1 metadata column:
#>        start       end     width |        gc
#>    <integer> <integer> <integer> | <numeric>

#> IRanges object with 5 ranges and 1 metadata column:
#>           start       end     width |        gc
#>       <integer> <integer> <integer> | <numeric>
#>   [1]         2         4         3 |  0.777584
#>   [2]         3         6         4 |  0.827303
#>   [3]         4         8         5 |  0.603324
#>   [4]         5        10         6 |  0.491232
#>   [5]         6        12         7 |  0.780359
```

Example: dealing with multi-mapping

This example is taken from the Bioconductor support site.

We have two *Ranges* objects. The first contains single nucleotide positions corresponding to an intensity measurement such as a ChIP-seq experiment, while the other contains coordinates for two genes of interest.

We want to identify which positions in the `intensities` *Ranges* overlap the genes, where each row corresponds to a position that overlaps a single gene.

First we create the two *Ranges* objects

```
#> GRanges object with 6 ranges and 0 metadata columns:
#>       seqnames    ranges strand
#>          <Rle> <IRanges>  <Rle>
```

```
#>   [1]       VI      3320      *
#>   [2]       VI      3321      *
#>   [3]       VI      3330      *
#>   [4]       VI      3331      *
#>   [5]       VI      3341      *
#>   [6]       VI      3342      *
#>   -------
#>   seqinfo: 1 sequence from an unspecified genome; no seqlengths

#> GRanges object with 2 ranges and 1 metadata column:
#>       seqnames    ranges strand |     gene_id
#>          <Rle> <IRanges>  <Rle> | <character>
#>   [1]       VI 3322-3846      * |     YFL064C
#>   [2]       VI 3030-3338      * |     YFL065C
#>   -------
#>   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

Now to find where the positions overlap each gene, we can perform an overlap join. This will automatically carry over the gene_id information as well as their coordinates (we can drop those by only selecting the gene_id).

```
#> GRanges object with 8 ranges and 1 metadata column:
#>       seqnames    ranges strand |     gene_id
#>          <Rle> <IRanges>  <Rle> | <character>
#>   [1]       VI      3320      * |     YFL065C
#>   [2]       VI      3321      * |     YFL065C
#>   [3]       VI      3330      * |     YFL065C
#>   [4]       VI      3330      * |     YFL064C
#>   [5]       VI      3331      * |     YFL065C
#>   [6]       VI      3331      * |     YFL064C
#>   [7]       VI      3341      * |     YFL064C
#>   [8]       VI      3342      * |     YFL064C
```

```
#>   -------
#>   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

Several positions match to both genes. We can count them using `summarise` and grouping by the `start` position:

```
#> DataFrame with 6 rows and 2 columns
#>       start         n
#>   <integer> <integer>
#> 1      3320         1
#> 2      3321         1
#> 3      3330         2
#> 4      3331         2
#> 5      3341         1
#> 6      3342         1
```

B.8 Grouping by overlaps

It's also possible to group by overlaps. Using this approach we can count the number of overlaps that are greater than 0.

```
#> IRanges object with 6 ranges and 2 metadata columns:
#> Groups: query [2]
#>           start       end     width |        gc     query
#>       <integer> <integer> <integer> | <numeric> <integer>
#>   [1]         5         9         5 |  0.827303         1
#>   [2]         5         9         5 |  0.603324         1
#>   [3]         5         9         5 |  0.491232         1
#>   [4]         5         9         5 |  0.780359         1
#>   [5]        10        14         5 |  0.491232         2
#>   [6]        10        14         5 |  0.780359         2
```

```
#> IRanges object with 6 ranges and 3 metadata columns:
#> Groups: query [2]
#>           start       end     width |        gc     query n_overlaps
#>       <integer> <integer> <integer> | <numeric> <integer>  <integer>
#>   [1]         5         9         5 |  0.827303         1          4
#>   [2]         5         9         5 |  0.603324         1          4
#>   [3]         5         9         5 |  0.491232         1          4
#>   [4]         5         9         5 |  0.780359         1          4
#>   [5]        10        14         5 |  0.491232         2          2
#>   [6]        10        14         5 |  0.780359         2          2
```

Of course we can also add overlap counts via the `count_overlaps()` function.

```
#> IRanges object with 4 ranges and 1 metadata column:
#>           start       end     width | n_overlaps
#>       <integer> <integer> <integer> |  <integer>
#>   [1]         5         9         5 |          4
#>   [2]        10        14         5 |          2
#>   [3]        15        19         5 |          0
#>   [4]        20        24         5 |          0
```

B.9 Reading Genomic Files

We provide convenience functions via **rtracklayer** and **GenomicAlignments** for reading/writing the following data formats to/from *Ranges* objects.

plyranges functions	File Format
`read_bam()`	BAM
`read_bed()/write_bed()`	BED
`read_bed_graph()/ write_bed_graph()`	BEDGraph
`read_narrowpeaks()/write_narrowpeaks()`	narrowPeaks
`read_gff() / write_gff()`	GFF(1-3)/ GTF

plyranges functions	File Format
`read_bigwig()` / `write_bigwig()`	BigWig
`read_wig()` /`write_wig()`	Wig

B.10 Learning more

There are many other resources and workshops available to learn to use **plyranges** and related Bioconductor packages, especially for more realistic analyses than the ones covered here:

- The fluentGenomics workflow package is an end-to-end workflow package for integrating differential expression results with differential accessibility results.
- The Bioc 2018 Workshop book has worked examples of using **plyranges** to analyse publicly available genomics data.
- The extended vignette in the plyrangesWorkshops package has a detailed walk through of using plyranges for coverage analysis.
- The case study by Michael Love using plyranges with tximeta to follow up on interesting hits from a combined RNA-seq and ATAC-seq analysis.

Bibliography

Alasoo, K, J Rodrigues, S Mukhopadhyay, A Knights, A Mann, K Kundu, HIPSCI-Consortium, C Hale, D G, and D Gaffney (2018). Shared genetic effects on chromatin and gene expression indicate a role for enhancer priming in immune response. *Nature Genetics* **50**, 424–431.

Alasoo, K and D Gaffney (2017). *Processed read counts from macrophage RNA-seq and ATAC-seq experiments*. Zenodo. https://doi.org/10.5281/zenodo.1188300.

Amezquita, RA, ATL Lun, E Becht, VJ Carey, LN Carpp, L Geistlinger, F Marini, K Rue-Albrecht, D Risso, C Soneson, L Waldron, H Pagès, ML Smith, W Huber, M Morgan, R Gottardo, and SC Hicks (2020). Orchestrating single-cell analysis with Bioconductor. *Nat. Methods* **17**(2), 137–145.

Asimov, D (1985). The Grand Tour: A Tool for Viewing Multidimensional Data. *SIAM J. Sci. and Stat. Comput.* **6**(1), 128–143.

Bache, SM and H Wickham (2014). *magrittr: A Forward-Pipe Operator for R*. R package version 1.5. https://CRAN.R-project.org/package=magrittr.

Barrett, T, SE Wilhite, P Ledoux, C Evangelista, IF Kim, M Tomashevsky, KA Marshall, KH Phillippy, PM Sherman, M Holko, A Yefanov, H Lee, N Zhang, CL Robertson, N Serova, S Davis, and A Soboleva (2013). NCBI GEO: archive for functional genomics data setsupdate. *Nucleic Acids Research* **41**(D1), D991–D995. eprint: /oup/backfile/content_public/journal/nar/41/d1/10.1093/nar/gks1193/2/gks1193.pdf.

Becker, RA and WS Cleveland (1987). Brushing Scatterplots. *Technometrics* **29**(2), 127–142.

Beygelzimer, A, S Kakadet, J Langford, S Arya, D Mount, and S Li (2019). *FNN: Fast Nearest Neighbor Search Algorithms and Applications*. R package version 1.1.3. https://CRAN.R-project.org/package=FNN.

Blondel, VD, JL Guillaume, R Lambiotte, and E Lefebvre (2008). Fast unfolding of communities in large networks. *J. Stat. Mech.* **2008**(10), P10008.

Brehmer, M, M Sedlmair, S Ingram, and T Munzner (2014). Visualizing dimensionally-reduced data: Interviews with analysts and a characterization of task sequences. In: *Proceedings of the Fifth Workshop on Beyond Time and Errors: Novel Evaluation Methods for Visualization*. dl.acm.org, pp.1–8. https://dl.acm.org/doi/abs/10.1145/2669557.2669559.

Buja, A and D Asimov (1986). Grand tour methods: an outline. In: *Proceedings of the Seventeenth Symposium on the interface of computer sciences and statistics on Computer science and statistics*. Lexington, Kentucky, USA: Elsevier North-Holland, Inc., pp.63–67. https://dl.acm.org/doi/10.5555/26036.26046.

Buja, A, D Cook, and DF Swayne (1996). Interactive High-Dimensional Data Visualization. *J. Comput. Graph. Stat.* **5**(1), 78–99.

Buja, A, C Hurley, and J McDonald (1986). A data viewer for multivariate data. In: *Computing Science and Statistics: Proceedings of the 18th Symposium on the Interface*. Washington: American Statistical Association, pp.171–174.

Buja, A, DF Swayne, ML Littman, N Dean, H Hofmann, and L Chen (2008). Data Visualization With Multidimensional Scaling. *J. Comput. Graph. Stat.* **17**(2), 444–472. eprint: http://dx.doi.org/10.1198/106186008X318440.

Chang, W, J Cheng, JJ Allaire, Y Xie, and J McPherson (2020). *shiny: Web Application Framework for R*. https://CRAN.R-project.org/package=shiny.

Coenen, A and A Pearce (2019). *Understanding UMAP*. https://pair-code.github.io/understanding-umap/. Accessed: 2020-4-17. https://pair-code.github.io/understanding-umap/.

Coifman, RR, S Lafon, AB Lee, M Maggioni, B Nadler, F Warner, and SW Zucker (2005). Geometric diffusions as a tool for harmonic analysis and structure definition of data: diffusion maps. *Proc. Natl. Acad. Sci. U. S. A.* **102**(21), 7426–7431.

Cook, D, A Buja, J Cabrera, and C Hurley (1995). Grand Tour and Projection Pursuit. *J. Comput. Graph. Stat.* **4**(3), 155–172.

Dale, RK, BS Pedersen, and AR Quinlan (2011). Pybedtools: a flexible Python library for manipulating genomic datasets and annotations. *Bioinformatics* **27**(24), 3423–3424.

Dudoit, S, YH Yang, MJ Callow, and TP Speed (2002). STATISTICAL METHODS FOR IDENTIFYING DIFFERENTIALLY EXPRESSED GENES IN REPLICATED cDNA MICROARRAY EXPERIMENTS. *Stat. Sin.* **12**(1), 111–139.

Eddelbuettel, D (2020). *RcppAnnoy: 'Rcpp' Bindings for 'Annoy', a Library for Approximate Nearest Neighbors*. R package version 0.0.16. https://CRAN.R-project.org/package=RcppAnnoy.

Fowler, M (n.d.). *FluentInterface*. https://martinfowler.com/bliki/FluentInterface.html. Accessed: 2020-2-23. https://martinfowler.com/bliki/FluentInterface.html.

Frankish, A, GENCODE-consoritum, and P Flicek (2018). GENCODE reference annotation for the human and mouse genomes. *Nucleic Acids Research*.

Friedman, JH and W Stuetzle (2002). John W. Tukey's work on interactive graphics. *Ann. Stat.* **30**(6), 1629–1639.

Genovese, C, M Perone-Pacifico, I Verdinelli, and L Wasserman (2017). Finding Singular Features. *J. Comput. Graph. Stat.* **26**(3), 598–609.

Green, TRG and M Petre (1996). Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework. *Journal of Visual Languages & Computing* **7**(2), 131–174.

Grolemund, G and H Wickham (2017). *R for Data Science*, p. 89. http://r4ds.had.co.nz/.

Hansen, KD, RA Irizarry, and Z Wu (2012). Removing technical variability in RNA-seq data using condition quantile normalization. *Biostatistics* **13** (2), 204–16.

Henry, L and H Wickham (2017). *rlang: Functions for Base Types and Core R and 'Tidyverse' Features*. http://rlang.tidyverse.org, https://github.com/r-lib/rlang. http://rlang.tidyverse.org.

Hotelling, H (1933). Analysis of a complex of statistical variables into principal components. en. *J. Educ. Psychol.* **24**(6), 417.

Huber, W, VJ Carey, R Gentleman, S Anders, M Carlson, BS Carvalho, HC Bravo, S Davis, L Gatto, T Girke, R Gottardo, F Hahne, KD Hansen, RA Irizarry, M Lawrence, MI Love, J MacDonald, V Obenchain, AK Ole, H Pagès, A Reyes, P Shannon, GK Smyth, D Tenenbaum, L Waldron, and M Morgan (2015a). Orchestrating high-throughput genomic analysis with Bioconductor. *Nat. Methods* **12**(2), 115–121.

Huber, W, VJ Carey, R Gentleman, S Anders, M Carlson, BS Carvalho, HC Bravo, S Davis, L Gatto, T Girke, R Gottardo, F Hahne, KD Hansen, RA Irizarry, M Lawrence, MI Love, J MacDonald, V Obenchain, AK Ole, H Pagès, A Reyes, P Shannon, GK Smyth, D Tenenbaum, L Waldron, and M Morgan (2015b). Orchestrating high-throughput genomic analysis with Bioconductor. *Nature Methods* **12**(2), 115–121.

Hyndman, RJ, E Wang, and N Laptev (2015). Large-Scale Unusual Time Series Detection. In: *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, pp.1616–1619. http://dx.doi.org/10.1109/ICDMW.2015.104.

Kaitoua, A, Pinoli, P, Bertoni, M, and Ceri, S (2017). Framework for Supporting Genomic Operations. *IEEE Trans. Comput.* **66**(3), 443–457.

Kobak, D and P Berens (2019). The art of using t-SNE for single-cell transcriptomics. *Nat. Commun.* **10**(1), 5416.

Korem, Y, P Szekely, Y Hart, H Sheftel, J Hausser, A Mayo, ME Rothenberg, T Kalisky, and U Alon (2015). Geometry of the Gene Expression Space of Individual Cells. *PLoS Comput. Biol.* **11**(7), e1004224.

Köster, J and S Rahmann (2012). Snakemakea scalable bioinformatics workflow engine. *Bioinformatics* **28**(19), 2520–2522.

Kozanitis, C and DA Patterson (2016). GenAp: a distributed SQL interface for genomic data. *BMC Bioinformatics* **17**, 63.

Kozanitis, Christos, Heiberg, Andrew, Varghese, George, and Bafna, Vineet (2014). Using Genome Query Language to uncover genetic variation. *Bioinformatics* **30**(1), 1–8.

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

Kruskal, JB (1964a). Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika* **29**(1), 1–27.

Kruskal, JB (1964b). Nonmetric multidimensional scaling: A numerical method. *Psychometrika* **29**(2), 115–129.

Laa, U, D Cook, A Buja, and G Valencia (2020). Hole or grain? A Section Pursuit Index for Finding Hidden Structure in Multiple Dimensions. arXiv: 2004.13327 [stat.CO].

Laa, U, D Cook, and S Lee (2020). Burning sage: Reversing the curse of dimensionality in the visualization of high-dimensional data. arXiv: 2009.10979 [stat.CO].

Laa, U, D Cook, and G Valencia (2020). A slice tour for finding hollowness in high-dimensional data. *J. Comput. Graph. Stat.*, 1–10.

Law, CW, M Alhamdoosh, S Su, X Dong, L Tian, GK Smyth, and ME Ritchie (2018). RNA-seq analysis is easy as 1-2-3 with limma, Glimma and edgeR. *F1000 Research* **5**(1408), 1408.

Lawrence, M, W Huber, H Pagès, P Aboyoun, M Carlson, R Gentleman, M Morgan, and V Carey (2013a). Software for Computing and Annotating Genomic Ranges. *PLoS Comput. Biol.* **9**.

Lawrence, M, W Huber, H Pagès, P Aboyoun, M Carlson, R Gentleman, M Morgan, and V Carey (2013b). Software for Computing and Annotating Genomic Ranges. *PLoS Comput. Biol.* **9**.

Lee, JA and M Verleysen (2009). Quality assessment of dimensionality reduction: Rank-based criteria. *Neurocomputing* **72**(7), 1431–1443.

Lee, S and D Cook (2020). *liminal: Multivariate Data Visualization With Tours and Embeddings*. R package version 0.0.5.9999. https://github.com/sa-lee/liminal.

Lee, S, D Cook, and M Lawrence (2019). plyranges: a grammar of genomic data transformation. *Genome Biology* **20**(1), 4.

Lee, S, M Lawrence, and D Cook (2018). *plyranges: a grammar of genomic data transformation*. https://doi.org/10.5281/zenodo.1469841.

Lee, S, M Lawrence, and MI Love (2020). Fluent genomics with *plyranges* and *tximeta*. *F1000Res.* **9**(109), 109.

Lee, S and M Love (n.d.). *fluentGenomics: A plyranges and tximeta workflow*. R package version 1.1.1. https://github.com/sa-lee/fluentGenomics.

Lee, S, AY Zhang, S Su, AP Ng, AZ Holik, ML Asselin-Labat, ME Ritchie, and CW Law (2020). Covering all your bases: incorporating intron signal from RNA-seq data. *NAR Genom Bioinform* **2**(3).

Lewis, J, L Van der Maaten, and V de Sa (2012). A behavioral investigation of dimensionality reduction. In: *Proceedings of the Annual Meeting of the Cognitive Science Society*. Vol. 34.

Lex, A, N Gehlenborg, H Strobelt, R Vuillemot, and H Pfister (2014). UpSet: Visualization of Intersecting Sets. *IEEE Trans. Vis. Comput. Graph.* **20**(12), 1983–1992.

Liao, Y, GK Smyth, and W Shi (2013). The Subread aligner: fast, accurate and scalable read mapping by seed-and-vote. *Nucleic Acids Res.* **41**(10), e108.

Liao, Y, GK Smyth, and W Shi (2019). The R package Rsubread is easier, faster, cheaper and better for alignment and quantification of RNA sequencing reads. *Nucleic Acids Res.* **47**(8), e47.

Linderman, GC and S Steinerberger (2019). Clustering with t-SNE, Provably. *SIAM Journal on Mathematics of Data Science* **1**(2), 313–332.

Love, MI, S Anders, V Kim, and W Huber (2016). RNA-Seq workflow: gene-level exploratory analysis and differential expression. *F1000 Research* **4**(1070), 1070.

Love, MI, C Soneson, PF Hickey, LK Johnson, N Tessa Pierce, L Shepherd, M Morgan, and R Patro (2019). Tximeta: reference sequence checksums for provenance identification in RNA-seq. *bioRxiv*, 777888.

Love, MI, W Huber, and S Anders (2014). Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome Biology* **15** (12), 550.

Lun, ATL, DJ McCarthy, and JC Marioni (2016). A step-by-step workflow for low-level analysis of single-cell RNA-seq data with Bioconductor. *F1000Res.* **5**, 2122.

Lyttle, I and Vega/Vega-Lite Developers (2020). *vegawidget: 'Htmlwidget' for 'Vega' and 'Vega-Lite'*. https://CRAN.R-project.org/package=vegawidget.

Maaten, L van der (2014). Accelerating t-SNE using tree-based algorithms. *J. Mach. Learn. Res.*

Maaten, L van der and G Hinton (2008). Visualizing Data using t-SNE. *J. Mach. Learn. Res.* **9**(Nov), 2579–2605.

Macosko, EZ, A Basu, R Satija, J Nemesh, K Shekhar, M Goldman, I Tirosh, AR Bialas, N Kamitaki, EM Martersteck, JJ Trombetta, DA Weitz, JR Sanes, AK Shalek, A Regev, and SA McCarroll (2015). Highly Parallel Genome-wide Expression Profiling of Individual Cells Using Nanoliter Droplets. *Cell* **161**(5), 1202–1214.

Markmiller, S, N Cloonan, RM Lardelli, K Doggett, MC Keightley, Y Boglev, AJ Trotter, AY Ng, SJ Wilkins, H Verkade, EA Ober, HA Field, SM Grimmond, GJ Lieschke, DYR Stainier, and JK Heath (2014). Minor class splicing shapes the zebrafish transcriptome during development. *Proc. Natl. Acad. Sci. U. S. A.* **111**(8), 3062–3067.

McCarthy, DJ and GK Smyth (2009). Testing significance relative to a fold-change threshold is a TREAT. *Bioinformatics* **25**(6), 765–771.

McCarthy, DJ, KR Campbell, ATL Lun, and QF Willis (2017). Scater: pre-processing, quality control, normalisation and visualisation of single-cell RNA-seq data in R. *Bioinformatics* **33** (8), 1179–1186.

McDonald, JA (1982). "Interactive graphics for data analysis". PhD thesis. Stanford University. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.331.6673&rep=rep1&type=pdf.

McInnes, L, J Healy, and J Melville (2018). UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. arXiv: 1802.03426 [stat.ML].

Melville, J (2020). *t-SNE Initialization Options*. https://jlmelville.github.io/smallvis/init.html. Accessed: 2020-3-23. https://jlmelville.github.io/smallvis/init.html.

Meng, C, OA Zeleznik, GG Thallinger, B Kuster, AM Gholami, and AC Culhane (2016). Dimension reduction techniques for the integrative analysis of multi-omics data. *Brief. Bioinform.* **17**(4), 628–641.

Moon, KR, D van Dijk, Z Wang, S Gigante, DB Burkhardt, WS Chen, K Yim, A van den Elzen, MJ Hirn, RR Coifman, NB Ivanova, G Wolf, and S Krishnaswamy (2019). "Visualizing Structure and Transitions for Biological Data Exploration". https://www.biorxiv.org/content/10.1101/120378v4.

Morgan, M (2017). *AnnotationHub: Client to access AnnotationHub resources*. R package version 2.13.1.

Morgan, M, H Pagès, V Obenchain, and N Hayden (2020). *Rsamtools: Binary alignment (BAM), FASTA, variant call (BCF), and tabix file import*. http://bioconductor.org/packages/Rsamtools.

Nguyen, LH and S Holmes (2019). Ten quick tips for effective dimensionality reduction. *PLoS Comput. Biol.* **15**(6), e1006907.

Ovchinnikova, S and S Anders (2020). Exploring dimension-reduced embeddings with Sleepwalk. *Genome Res.* **30**(5), 749–756.

Pagès, H, P Aboyoun, R Gentleman, and S DebRoy (2018). *Biostrings: Efficient manipulation of biological strings*. R package version 2.49.0.

Patro, R, G Duggal, M Love, R Irizarry, and C Kingsford (2017). Salmon provides fast and bias-aware quantification of transcript expression. *Nature Methods* **14**, 417–419.

Pezzotti, N, BPF Lelieveldt, L Van Der Maaten, T Hollt, E Eisemann, and A Vilanova (2017). Approximated and User Steerable tSNE for Progressive Visual Analytics. *IEEE Trans. Vis. Comput. Graph.* **23**(7), 1739–1752.

Quinlan, AR and IM Hall (2010). BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics* **26**(6), 841–842.

R Core Team (2019). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. https://www.R-project.org/.

Ramrez, F, F Dündar, S Diehl, BA Grüning, and T Manke (2014). deepTools: a flexible platform for exploring deep-sequencing data. *Nucleic Acids Res.* **42**(Web Server issue), W187–91.

Ramos, M, L Schiffer, A Re, R Azhar, A Basunia, C Rodriguez, T Chan, P Chapman, SR Davis, D Gomez-Cabrero, AC Culhane, B Haibe-Kains, KD Hansen, H Kodali, MS Louis, AS Mer, M Riester, M Morgan, V Carey, and L Waldron (2017). Software for the Integration of Multiomics Experiments in Bioconductor. en. *Cancer Res.* **77**(21), e39–e42.

Rauber, A (2009). *Multi-challenge Data Set*. http://ifs.tuwien.ac.at/dm/dataSets.html. Accessed: 2020-09-17. http://ifs.tuwien.ac.at/dm/dataSets.html.

Rieck, B (2017). "Persistent Homology in Multivariate Data Visualization". PhD thesis. Ruprecht-Karls-Universität Heidelberg.

Riemondy, KA, RM Sheridan, A Gillen, Y Yu, CG Bennett, and JR Hesselberth (2017). valr: Reproducible Genome Interval Arithmetic in R. *F1000Research*.

Risso, D and M Cole (2019). *scRNAseq: Collection of Public Single-Cell RNA-Seq Datasets*. R package version 2.0.2.

Ritchie, ME, B Phipson, D Wu, Y Hu, CW Law, W Shi, and GK Smyth (2015). limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Res.* **43**(7), e47.

Roadmap Epigenomics Consortium, A Kundaje, W Meuleman, J Ernst, M Bilenky, A Yen, A Heravi-Moussavi, P Kheradpour, Z Zhang, J Wang, MJ Ziller, V Amin, JW Whitaker, MD Schultz, LD Ward, A Sarkar, G Quon, RS Sandstrom, ML Eaton, YC Wu, AR Pfenning, X Wang, M Claussnitzer, Y Liu, C Coarfa, RA Harris, N Shoresh,

CB Epstein, E Gjoneska, D Leung, W Xie, RD Hawkins, R Lister, C Hong, P Gascard, AJ Mungall, R Moore, E Chuah, A Tam, TK Canfield, RS Hansen, R Kaul, PJ Sabo, MS Bansal, A Carles, JR Dixon, KH Farh, S Feizi, R Karlic, AR Kim, A Kulkarni, D Li, R Lowdon, G Elliott, TR Mercer, SJ Neph, V Onuchic, P Polak, N Rajagopal, P Ray, RC Sallari, KT Siebenthall, NA Sinnott-Armstrong, M Stevens, RE Thurman, J Wu, B Zhang, X Zhou, AE Beaudet, LA Boyer, PL De Jager, PJ Farnham, SJ Fisher, D Haussler, SJM Jones, W Li, MA Marra, MT McManus, S Sunyaev, JA Thomson, TD Tlsty, LH Tsai, W Wang, RA Waterland, MQ Zhang, LH Chadwick, BE Bernstein, JF Costello, JR Ecker, M Hirst, A Meissner, A Milosavljevic, B Ren, JA Stamatoyannopoulos, T Wang, and M Kellis (2015). Integrative analysis of 111 reference human epigenomes. *Nature* **518**(7539). NCBI GEO accession numbers GSM433167 https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM433167 and GPL18952 https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GPL18952.

Satyanarayan, A, D Moritz, K Wongsuphasawat, and J Heer (2017). Vega-Lite: A Grammar of Interactive Graphics. *IEEE Trans. Vis. Comput. Graph.* **23**(1), 341–350.

Schloerke, B, J Crowley, D Cook, F Briatte, M Marbach, E Thoen, A Elberg, and J Larmarange (2020). *GGally: Extension to 'ggplot2'*. R package version 1.5.0. https://CRAN.R-project.org/package=GGally.

Sedlmair, M, T Munzner, and M Tory (2013). Empirical guidance on scatterplot and dimension reduction technique choices. *IEEE Trans. Vis. Comput. Graph.* **19**(12), 2634–2643.

Shepherd, L and M Morgan (2019). *BiocFileCache: Manage Files Across Sessions*. R package version 1.10.2.

Silva, VD and JB Tenenbaum (2003). Global versus local methods in nonlinear dimensionality reduction. *Adv. Neural Inf. Process. Syst.*

Sims, D, I Sudbery, NE Ilott, A Heger, and CP Ponting (2014). Sequencing depth and coverage: key considerations in genomic analyses. *Nat. Rev. Genet.* **15**(2), 121–132.

Smilkov, D, N Thorat, C Nicholson, E Reif, FB Viégas, and M Wattenberg (2016). Embedding Projector: Interactive Visualization and Interpretation of Embeddings. arXiv: 1611.05469 [stat.ML].

Smyth, GK (2004). Linear Models and Empirical Bayes Methods for Assessing Differential Expression in Microarray Experiments. *Statistical Applications in Genetics and Molecular Biology* **3**(1).

Soneson, C, MI Love, and M Robinson (2015). Differential analyses for RNA-seq: transcript-level estimates improve gene-level inferences. *F1000Research* **4** (1521).

Stein-O'Brien, GL, R Arora, AC Culhane, AV Favorov, LX Garmire, CS Greene, LA Goff, Y Li, A Ngom, MF Ochs, Y Xu, and EJ Fertig (2018). Enter the Matrix: Factorization Uncovers Knowledge from Omics. en. *Trends Genet.* **34**(10), 790–805.

Swayne, DF and A Buja (2004). Exploratory Visual Analysis of Graphs in GGOBI. In: *COMPSTAT 2004 — Proceedings in Computational Statistics*. Physica-Verlag HD, pp.477–488. http://dx.doi.org/10.1007/978-3-7908-2656-2_39.

Swayne, DF, D Cook, and A Buja (1998). XGobi: Interactive Dynamic Data Visualization in the X Window System. *J. Comput. Graph. Stat.* **7**(1), 113–130.

Swayne, DF, DT Lang, A Buja, and D Cook (2003). GGobi: Evolving from XGobi into an extensible framework for interactive data visualization. *Comput. Stat. Data Anal.* **43**(4), 423–444.

Tang, J, J Liu, M Zhang, and Q Mei (2016). Visualizing Large-scale and High-dimensional Data. arXiv: 1602.00370 [cs.LG].

Torgerson, WS (1952). Multidimensional scaling: I. Theory and method. *Psychometrika* **17**(4), 401–419.

Tukey, JW (1977). *Exploratory data analysis*. Vol. 2. Reading, Mass.: Addison-Wesley Pub. Co.

Turunen, JJ, EH Niemelä, B Verma, and MJ Frilander (2013). The significant other: splicing by the minor spliceosome. *Wiley Interdiscip. Rev. RNA* **4**(1), 61–76.

Ushey, K (2019). *renv: Project Environments*. R package version 0.7.0-54. https://rstudio.github.io/renv.

Wattenberg, M, F Viégas, and I Johnson (2016). How to Use t-SNE Effectively. *Distill* **1**(10).

Wickham, H (2014). Tidy Data. *Journal of Statistical Software, Articles* **59**(10), 1–23.

Wickham, H (2016). *ggplot2: Elegant Graphics for Data Analysis*. Use R! Springer International Publishing. https://play.google.com/store/books/details?id=RTMFswEACAAJ.

Wickham, H, M Averick, J Bryan, W Chang, LD McGowan, R François, G Grolemund, A Hayes, L Henry, J Hester, M Kuhn, TL Pedersen, E Miller, SM Bache, K Müller, J Ooms, D Robinson, DP Seidel, V Spinu, K Takahashi, D Vaughan, C Wilke, K Woo, and H Yutani (2019). Welcome to the tidyverse. *Journal of Open Source Software* **4**(43), 1686.

Wickham, H, D Cook, and H Hofmann (2015). Visualizing statistical models: Removing the blindfold. *Statistical Analysis and Data Mining: The ASA Data Science Journal* **8**(4), 203–225.

Wickham, H, D Cook, H Hofmann, and A Buja (2011). tourr: An R Package for Exploring Multivariate Data with Projections. *Journal of Statistical Software, Articles* **40**(2), 1–18.

Wickham, H, R Francois, L Henry, and K Müller (2017). *dplyr: A Grammar of Data Manipulation*. R package version 0.7.4. https://CRAN.R-project.org/package=dplyr.

Wilke, CO (2019). *cowplot: Streamlined Plot Theme and Plot Annotations for 'ggplot2'*. R package version 1.0.0. https://CRAN.R-project.org/package=cowplot.

Wilkinson, L (2005). *The Grammar of Graphics*. Statistics and Computing. Springer Science & Business Media. https://market.android.com/details?id=book-_kRX4LoFfGQC.

Wilkinson, L, A Anand, and R Grossman (2005). Graph-Theoretic Scagnostics. In: *Proceedings of the Proceedings of the 2005 IEEE Symposium on Information Visualization*. INFOVIS '05. Washington, DC, USA: IEEE Computer Society, pp.21–. https://doi.org/10.1109/INFOVIS.2005.14.

Xie, Y (2016). *bookdown: Authoring Books and Technical Documents with R Markdown*. CRC Press. https://play.google.com/store/books/details?id=8nm0DQAAQBAJ.

Xie, Y (2017). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC. https://www.taylorfrancis.com/books/9781315382487.