METHOD ARTICLE

recount workflow: Accessing over 70,000 human RNA-seq samples with Bioconductor [version 1; referees: 1 approved with reservations]

Leonardo Collado-Torres1,2, Abhinav Nellore3-5, Andrew E. Jaffe1,2,6,7

1Lieber Institute for Brain Development, Baltimore, MD, 21205, USA
2Center for Computational Biology, Johns Hopkins University, Baltimore, MD, 21205, USA
3Department of Biomedical Engineering, Oregon Health and Science University, Portland, OR, 97239, USA
4Department of Surgery, Oregon Health and Science University, Portland, OR, 97239, USA
5Computational Biology Program, Oregon Health and Science University, Portland, OR, 97239, USA
6Department of Biostatistics, Johns Hopkins Bloomberg School of Public Health, Baltimore, MD, 21205, USA
7Department of Mental Health, Johns Hopkins Bloomberg School of Public Health, Baltimore, MD, 21205, USA

Abstract
The recount2 resource is composed of over 70,000 uniformly processed human RNA-seq samples spanning TCGA and SRA, including GTEx. The processed data can be accessed via the recount2 website and the recount Bioconductor package. This workflow explains in detail how to use the recount package and how to integrate it with other Bioconductor packages for several analyses that can be carried out with the recount2 resource. In particular, we describe how the coverage count matrices were computed in recount2 as well as different ways of obtaining public metadata, which can facilitate downstream analyses. Step-by-step directions show how to do a gene-level differential expression analysis, visualize base-level genome coverage data, and perform an analyses at multiple feature levels. This workflow thus provides further information to understand the data in recount2 and a compendium of R code to use the data.

This article is included in the Bioconductor gateway.
Corresponding author: Leonardo Collado-Torres (leo.collado@libd.org)

Author roles: Collado-Torres L: Conceptualization, Software, Visualization, Writing – Original Draft Preparation, Writing – Review & Editing;
Nellore A: Conceptualization, Writing – Review & Editing; Jaffe AE: Conceptualization, Funding Acquisition, Supervision, Writing – Review & Editing

Competing interests: No competing interests were disclosed.

How to cite this article: Collado-Torres L, Nellore A and Jaffe AE. recount workflow: Accessing over 70,000 human RNA-seq samples with Bioconductor [version 1; referees: 1 approved with reservations] F1000Research 2017, 6:1558 (doi: 10.12688/f1000research.12223.1)

Copyright: © 2017 Collado-Torres L et al. This is an open access article distributed under the terms of the Creative Commons Attribution Licence, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Grant information: LCT and AEJ were supported by the National Institutes of Health (grant R21 MH109956-01). LCT and AN were supported by the National Institutes of Health (grant R01 GM105705).

The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Introduction

RNA sequencing (RNA-seq) is now the most widely used high-throughput assay for measuring gene expression. In a typical RNA-seq experiment, several million reads are sequenced per sample. The reads are often aligned to the reference genome using a splice-aware aligner to identify where reads originated. Resulting alignment files are then used to compute count matrices for several analyses such as identifying differentially expressed genes. The Bioconductor project has many contributed packages that specialize in analyzing this type of data and previous workflows have explained how to use them. Initial steps are typically focused on generating the count matrices. Some pre-computed matrices have been made available via the ReCount project or Bioconductor Experiment data packages such as the airway dataset. The pre-computed count matrices in ReCount have been useful to RNA-seq methods developers and to researchers seeking to avoid the computationally intensive process of creating these matrices. In the years since ReCount was published, hundreds of new RNA-seq projects have been carried out, and researchers have shared the data publicly.

We recently uniformly processed over 70,000 publicly available human RNA-seq samples, and made the data available via the recount2 resource at jhubiostatistics.shinyapps.io/recount. Samples in recount2 are grouped by project (over 2,000) originating from the Sequence Read Archive, the Genotype-Tissue Expression study (GTEx) and the Cancer Genome Atlas (TCGA). The processed data can be accessed via the recount Bioconductor package available at bioconductor.org/packages/recount. Together, recount2 and the recount Bioconductor package should be considered a successor to ReCount.

Due to space constraints, the recount2 publication did not cover how to use the recount package and other useful information for carrying out analyses with recount2 data. We describe how the count matrices in recount2 were generated. We also review the R code necessary for using the recount2 data, whose details are important because some of this code involves multiple Bioconductor packages and changing default options. We further show: a) how to augment metadata that comes with datasets with metadata learned from natural language processing of associated papers as well as expression data b) how to perform differential expression analyses, and c) how to visualize the base-pair data available from recount2.

Analysis of RNA-seq data available at recount2

recount2 overview

The recount2 resource provides expression data summarized at different feature levels to enable novel cross-study analyses. Generally when investigators use the term expression, they think about gene expression. But more information can be extracted from RNA-seq data. Once RNA-seq reads have been aligned to the reference genome it is possible to determine the number of aligned reads overlapping each base-pair resulting in the genome base-pair coverage curve as shown in Figure 1. In the example shown in Figure 1, most of the reads overlap known exons from a gene. Those reads can be used to compute a count matrix at the exon or gene feature levels. Some reads span exon-exon junctions (jx) and while most match the annotation, some do not (jx 3 and 4). An exon-exon junction count matrix can be used to identify differentially expressed junctions, which can show which isoforms are differentially expressed given sufficient coverage. For example, junctions 2 and 5 are unique to isoform 2, while junction 6 is unique to isoform 1. The genome base-pair coverage data can be used with derfinder to identify expressed regions; some of these could be unannotated exons, which together with the exon-exon junction data could help establish new isoforms.

recount2 provides gene, exon, and exon-exon junction count matrices both in text format and RangedSummarizedExperiment objects (rse) as shown in Figure 2. These rse objects provide information about the expression features (for example gene IDs) and the samples. In this workflow we will explain how to add metadata to the rse objects in recount2 in order to ask biological questions. recount2 also provides coverage data in the form of bigWig files. All four features can be accessed with the recount Bioconductor package. recount also allows sending queries to snaptron to search for specific exon-exon junctions.
**Figure 1. Overview of the data available in recount2.** Reads (pink boxes) aligned to the reference genome can be used to compute a base-pair coverage curve and identify exon-exon junctions (split reads). Gene and exon count matrices are generated using annotation information providing the gene (green boxes) and exon (blue boxes) coordinates together with the base-level coverage curve. The reads spanning exon-exon junctions (jx) are used to compute a third count matrix that might include unannotated junctions (jx 3 and 4). Without using annotation information, expressed regions (orange box) can be determined from the base-level coverage curve to then construct data-driven count matrices.

**Packages used in the workflow**

In this workflow we will use several Bioconductor packages. To reproduce the entirety of this workflow, install the packages using the following code after installing R 3.4.x from CRAN in order to use Bioconductor version 3.5 or newer.

```r
## Install packages from Bioconductor
source("https://bioconductor.org/biocLite.R")
```

Once they are installed, load all the packages with the following code.

```r
library("recount")
library("GenomicRanges")
library("limma")
library("edgeR")
library("DESeq2")
library("regionReport")
library("clusterProfiler")
library("org.Hs.eg.db")
library("gplots")
library("derfinder")
library("rtracklayer")
library("GenomicFeatures")
library("bumphunter")
library("derfinderPlot")
library("devtools")
```
Figure 2. recount2 provides coverage count matrices in RangedSummarizedExperiment (rse) objects. Once the rse object has been downloaded and loaded into R, the feature information is accessed with rowRanges(rse) (blue box), the counts with assays(rse)$counts (pink box) and the sample metadata with colData(rse) (green box). The sample metadata can be expanded using add_predictions(rse) (orange box) or with custom code (brown box) matching by a unique sample identifier such as the SRA Run ID. The rse object is inside the purple box and matching data is highlighted in each box.

Coverage counts provided by recount2
The most accessible features are the gene, exon and exon-exon junction count matrices. This section explains them in greater detail. Figure 3 shows 16 RNA-seq reads, each 3 base-pairs long, and a reference genome.

Reads in the recount2 resource were aligned with the splice-aware Rail-RNA aligner\(^1\). Figure 4 shows the reads aligned to the reference genome. Some of the reads are split as they span an exon-exon junction. Two of the reads were soft clipped meaning that just a portion of the reads aligned (top left in purple).

In order to compute the gene and exon count matrices we first have to process the annotation, which for recount2 is Gencode v25 (CHR regions) with hg38 coordinates. Although recount can generate count matrices for other annotations using hg38 coordinates. Figure 5 shows two isoforms for a gene composed of 3 different exons.

The coverage curve is at base-pair resolution so if we are interested in gene counts we have to be careful not to double count base-pairs 1 through 5 that are shared by exons 1 and 3 (Figure 5). Using the function disjoin() from GenomicRanges\(^2\) we identified the distinct exonic sequences (disjoint exons). The following code defines the exon coordinates that match Figure 5 and the resulting disjoint exons for our example gene. The resulting disjoint exons are shown in Figure 6.
Figure 3. RNA-seq starting data. 16 RNA-seq un-aligned RNA-seq reads 3 base-pairs long are shown (pink boxes) alongside a reference genome that is 16 base-pairs long (white box).

Figure 4. Aligned RNA-seq reads. Spice-aware RNA-seq aligners such as Rail-RNA are able to find the coordinates to which the reads map, even if they span exon-exon junctions (connected boxes). Rail-RNA soft clips some reads (purple boxes with rough edges) such that a portion of these reads align to the reference genome.

Figure 5. Gene annotation. A single gene with two isoforms composed by three distinct exons (blue boxes) is illustrated. Exons 1 and 3 share the first five base-pairs while exon 2 is common to both isoforms.

Figure 6. Disjoint exons. Windows of distinct exonic sequence for the example gene. Disjoint exons 1 and 2 form exon 1.

```r
library("GenomicRanges")
exons <- GRanges("seq", IRanges(start = c(1, 1, 13), end = c(5, 8, 15)))
exons
```

```
## GRanges object with 3 ranges and 0 metadata columns:
##  seqlengths
##    <Rle>  <IRanges>  <Rle>
## [1] seq     [ 1,  5]      *
## [2] seq     [ 1,  8]      *
## -------
## seqinfo: 1 sequence from an unspecified genome; no seqlengths
```
Now that we have disjoint exons, we can compute the base-pair coverage for each of them as shown in Figure 7. That is, for each base-pair that corresponds to exonic sequence, we compute the number of reads overlapping that given base-pair. For example, the first base-pair is covered by 3 different reads and it does not matter whether the reads themselves were soft clipped. Not all reads or bases of a read contribute information to this step, as some do not overlap known exonic sequence (light pink in Figure 7).

With base-pair coverage for the exonic sequences computed, the coverage count for each distinct exon is simply the sum of the base-pair coverage for each base in a given distinct exon. For example, the coverage count for disjoint exon 2 is $2 + 2 + 3 = 7$ as shown in Figure 8. The gene coverage count is then $\sum_i \text{coverage}$, where $n$ is the number of exonic base-pairs for the gene and is equal to the sum of the coverage counts for its disjoint exons as shown in Figure 8.

**Figure 7.** Base-pair coverage counting for exonic base-pairs. At each exonic base-pair we compute the number of reads overlapping that given base-pair. The first base (orange arrow) has 3 reads overlapping that base-pair. Base-pair 11 has a coverage of 3 but does not overlap known exonic sequence, so that information is not used for the gene and exon count matrices (grey arrow). If a read partially overlaps exonic sequence, only the portion that overlaps is used in the computation (see right most read).

**Figure 8.** Exon and gene coverage counts. The coverage counts for each disjoint exon are the sum of the base-pair coverage. The gene coverage count is the sum of the disjoint exons coverage counts.
For the exons, recount2 provides the disjoint exons coverage count matrix. It is possible to reconstruct the exon coverage count matrix by summing the coverage count for the disjoint exons that compose each exon. For example, the coverage count for exon 1 would be the sum of the coverage counts for disjoint exons 1 and 2, that is $19 + 7 = 26$. Some methods might assume that double counting of the shared base-pairs was performed while others assume or recommend the opposite.

**Scaling coverage counts**

The coverage counts described previously are the ones actually included in the rse objects in recount2 instead of typical read count matrices. This is an important difference to keep in mind as most methods were developed for read count matrices. Part of the sample metadata available from recount2 includes the read length and number of mapped reads. Given a target library size (40 million reads by default), the coverage counts in recount2 can be scaled to read counts for a given library size as shown in Equation (1). Note that the resulting scaled read counts are not necessarily integers so it might be necessary to round them if a differential expression (DE) method assumes integer data.

$$\frac{\sum \text{coverage}}{\text{Read Length} \times \text{mapped}} \times \text{target} = \text{scaled read counts}$$  \hspace{1cm} (1)

From Figure 4 we know that Rail-RNA soft clipped some reads, so a more precise measure than the denominator of Equation (1) is the area under coverage (AUC) which is the sum of the coverage for all base-pairs of the genome, regardless of the annotation as shown in Figure 9. Without soft clipping reads, the AUC would be equal to the number of reads mapped multiplied by the read length. So for our example gene, the scaled counts for a library size of 20 reads would be $\frac{36}{45} \times 20 = 16$ and in general calculated with Equation (2). The following code shows how to compute the AUC given a set of aligned reads and reproduce a portion of Figure 9.

$$\frac{\sum \text{coverage}}{\text{AUC}} \times \text{target} = \text{scaled read counts}$$  \hspace{1cm} (2)

**Figure 9. Area under coverage (AUC).** The area under coverage is the sum of the base-pair coverage for all positions in the genome regardless of the annotation. It is the area under the base-level coverage curve shown as the light blue area under the pink curve.
### Take the example and translate it to R code

```r
library("GenomicRanges")
reads <- GRanges("seq", IRanges(
    start = rep(
        c(1, 2, 3, 4, 5, 7, 8, 9, 10, 13, 14),
        c(3, 1, 2, 1, 2, 1, 2, 4, 1),
    ),
    width = rep(
        c(1, 3, 2, 3, 1, 2, 1, 3, 2, 3, 2, 1, 3),
        c(1, 4, 1, 2, 1, 2, 1, 2, 1, 2, 1, 1)
    )
))
## Get the base-level genome coverage curve
cov <- as.integer(coverage(reads)$seq)
## [1] 45
## AUC
sum(cov)
## Code for reproducing the bottom portion of Figure 8.
par(mar = c(5, 6, 4, 2) + 0.1)
plot(cov, type = "o", col = "violetred1", lwd = 10, ylim = c(0, 5),
    xlab = "Genome", ylab = "Coverage", cex.axis = 2, cex.lab = 3,
    bty = "n")
polygon(c(1, seq_len(length(cov)), length(cov)), c(0, cov, 0),
    border = NA, density = -1, col = "light blue")
points(seq_len(length(cov)), cov, col = "violetred1", type = "o",
    lwd = 10)
dev.off()
```

The `recount` function `scale_counts()` computes the scaled read counts for a target library size of 40 million reads and we highly recommend using it before doing other analyses. The following code shows how to use `scale_counts()` and that the resulting read counts per sample can be lower than the target size (40 million). This happens when not all mapped reads overlap known exonic base-pairs of the genome. In our example, the gene has a scaled count of 16 reads for a library size of 20 reads, meaning that 4 reads did not overlap exonic sequences.

```r
## Check that the number of reads is less than or equal to 40 million
## after scaling.
library("recount")
reads_scaled <- scale_counts(reads_scaled, round = FALSE)
summary(colSums(as.matrix(reads_scaled)) / 1e6)
```

### Enriching the annotation

Data in `recount2` can be used for annotation-agnostic analyses and enriching the known annotation. Just like exon and gene coverage count matrices, `recount2` provides exon-exon junction count matrices. These matrices can be used to identify new isoforms (Figure 10) or identify differentially expressed isoforms. For example, exon-exon junctions 2, 5 and 6 in Figure 1 are only present in one annotated isoform. `Snaptron` allows programatic and high-level queries of the exon-exon junction information and its graphical user interface is specially useful for visualizing this data. Inside R, the `recount` function `snaptron_query()` can be used for searching specific exon-exon junctions in `recount2`.

The base-pair coverage data from `recount2` can be used together with `derfinder` to identify expressed regions of the genome, providing another annotation-agnostic analysis of the expression data. Using the function `expressed_regions()` we can identify regions of expression based on a given data set in `recount2`. These regions might overlap known exons but can also provide information about intron retention events (Figure 11), improve detection of exon boundaries (Figure 12), and help identify new exons (Figure 1) or expressed sequences in intergenic regions. Using `coverage_matrix()` we can compute a coverage matrix based on the expressed regions or
Figure 10. Exon-exon junctions go beyond the annotation. Reads spanning exon-exon junctions are highlighted and compared against the annotation. Three of them match the annotated junctions, but one (blue and orange read) spans an unannotated exon-exon junction with the left end matching the annotation and the right end hinting at a possible new isoform for this gene (blue and orange isoform).

Figure 11. Intron retention events. Some reads might align with known intronic segments of the genome and provide information for exploring intron retention events (pink read). Some might support an intron retention event or a new isoform when coupled with exon-exon junction data (orange read).

Figure 12. Exon boundaries. Reads that go beyond the known exon boundaries can inform us of whether the annotated boundaries are correct or if there was a run-off transcription event.

another set of genomic intervals. The resulting matrix can then be used for a DE analysis, just like the exon, gene and exon-exon junction matrices.

Gene level analysis
Having reviewed how the coverage counts in recount2 were produced, we can now do a DE analysis. We will use data from 72 individuals spanning the human lifespan, split into 6 age groups with SRA accession SRP04563813. The function `download_study()` requires a SRA accession which can be found using `abstract_search()`. `download_study()` can then be used to download the gene coverage count data as well as other expression features. The files are saved in a directory named after the SRA accession, in this case SRP045638.
library("recount")

## Find the project ID by searching abstracts of studies
abstract_search("human brain development by age")

## number_samples species
## 1296   72   human
## abstract
## 1296 RNAseq data of 36 samples across human brain development by age group from LIBD
## project
## 1296 SRP045638

## Download the data if it is not there
if(!file.exists(file.path("SRP045638", "rse_gene.Rdata"))) {
  download_study("SRP045638", type = "rse-gene")
}

## 2017-07-30 10:11:16 downloading file rse_gene.Rdata to SRP045638

## Check that the file was downloaded
file.exists(file.path("SRP045638", "rse_gene.Rdata"))

## [1] TRUE

## Load the data
load(file.path("SRP045638", "rse_gene.Rdata"))

The coverage count matrices are provided as `RangedSummarizedExperiment` objects (rse). These objects store information at the feature level, the samples and the actual count matrix as shown in Figure 1 of Love et al., 2016. Figure 2 shows the actual rse objects provided by recount2 and how to access the different portions of the data. Using a unique sample ID such as the SRA Run ID it is possible to expand the sample metadata. This can be done using the predicted phenotype provided by add_predictions()14, pulling information from GEO via find_geo() and geo_characteristics(), or with custom code.

**Metadata**

Using the `colData()` function we can access sample metadata. More information on these metadata is provided in the Supplementary material of the recount2 paper, and we provide a brief review here. The rse objects for SRA data sets include 21 columns with mostly technical information. The GTEx and TCGA rse objects include additional metadata as available from the raw sources. In particular, we compiled metadata for GTEx using the v6 phenotype information available at gtexportal.org, and we put together a large table of TCGA case and sample information by combining information accumulated across Seven Bridges’ Cancer Genomics Cloud and TCGAbiolinks15.

## One row per sample, one column per phenotype variable
dim(colData(rse_gene))

## [1] 72 21

## Mostly technical variables are included
colnames(colData(rse_gene))

## [1] "project"
## [2] "sample"
## [3] "experiment"
## [4] "run"
## [5] "read_count_as_reported_by_sra"
## [6] "reads_downloaded"
## [7] "proportion_of_reads_reported_by_sra_downloaded"
## [8] "paired_end"
## [9] "sra_misreported_paired_end"
## [10] "mapped_read_count"
## [11] "auc"
## [12] "sharq_beta_tissue"
## [13] "sharq_beta_cell_type"
## [14] "biosample_submission_date"
Technical variables Several of these technical variables include the number of reads as reported by SRA, the actual number of reads Rail-RNA was able to download (which might be lower in some cases), the number of reads mapped by Rail-RNA, whether the sample is paired-end or not, the coverage AUC and the average read length (times 2 for paired-end samples). Note that the sample with SRA Run ID SRR2071341 has about 240.8 million reads as reported by SRA, while it has 120.4 million spots reported in https://trace.ncbi.nlm.nih.gov/Traces/sra/?run=SRR2071341; that is because it is a paired-end sample (2 reads per spot). These details are important for those interested in writing alternative scaling functions to `scale_counts()`.

```r
## Input reads: number reported by SRA might be larger than number of reads Rail-RNA downloaded
colData(rse_gene)[, c("read_count_as_reported_by_sra", "reads_downloaded")]
```

```r
dataFrame with 72 rows and 2 columns
read_count_as_reported_by_sra reads_downloaded
## <integer>             <integer>
## SRR2071341 240797206 240797206
## SRR2071345  82266652  82266652
## SRR2071346 132911310 132911310
## SRR2071347  74051302  74051302
## SRR2071348 250259914 250259914
## ...       ...       ...
## SRR1554541 186250218 162403466
## SRR1554554 140038024 121793680
## SRR1554535 106244496  91185969
## SRR1554558 200687480 170754145
## SRR1554553  90579486  51803404

summary(colData(rse_gene)$proportion_of_reads_reported_by_sra_downloaded)
```

```r
## Min. 1st Qu. Median   Mean 3rd Qu.   Max.
## 0.5719  0.9165 0.9788 0.9532  1.0000 1.0000
```

```r
## AUC information used by scale_counts() by default
head(colData(rse_gene)$auc)
```

```r
## [1] 22950214241 7553726235 12018044330 7041243857 24062460144 45169026301
```

```r
## Alternatively, scale_counts() can use the number of mapped reads and other information
colData(rse_gene)[, c("mapped_read_count", "paired_end", "avg_read_length")]
```

```r
dataFrame with 72 rows and 3 columns
mapped_read_count paired_end avg_read_length
## <integer>             <logical>       <integer>
## SRR2071341 232970536       TRUE     200
## SRR2071345  78431778       TRUE     200
## SRR2071346 124493632       TRUE     200
## SRR2071347  71742875       TRUE     200
## SRR2071348 242992735       TRUE     200
## ...       ...       ...
## SRR1554541 162329325       TRUE     174
## SRR1554554 121738246       TRUE     173
## SRR1554535  91120421       TRUE     171
## SRR1554550 170648458       TRUE     170
## SRR1554553  51684462       TRUE     114
```
**Biological information** Other metadata variables included provide more biological information, such as the SHARQ beta tissue and cell type predictions, which are based on processing the abstract of papers. This information is available for some of the SRA projects.

```r
## SHARQ tissue predictions: not present for all studies
colData(rse_gene)$sharq_beta_tissue
```

```r
## [1] NA NA NA NA NA NA
```

```r
colData(rse_gene_SRP009615)$sharq_beta_tissue
```

```r
## [1] "blood" "blood" "blood" "blood" "blood" "blood"
```

For some data sets we were able to find the GEO accession IDs, which we then used to create the title and characteristics variables. If present, the characteristics information can be used to create additional metadata variables by parsing the CharacterList in which it is stored. Since the input is free text, sometimes more than one type of wording is used to describe the same information, meaning that we might have to process that information in order to build a more convenient variable, such as a factor vector.

```r
## GEO information was absent for the SRP045638 data set
colData(rse_gene)[, c("geo_accession", "title", "characteristics")]
```

```r
## DataFrame with 72 rows and 3 columns
##             geo_accession       title characteristics
## SRR2071341            NA          NA               NA
## SRR2071345            NA          NA               NA
## SRR2071346            NA          NA               NA
## SRR2071347            NA          NA               NA
## SRR2071348            NA          NA               NA
## ...                  ...         ...              ...
## SRR1554541            NA          NA               NA
## SRR1554554            NA          NA               NA
## SRR1554535            NA          NA               NA
## SRR1554558            NA          NA               NA
## SRR1554553            NA          NA               NA
```

```r
## GEO information for the SRP009615 data set
colData(rse_gene_SRP009615)$geo_accession
```

```r
## [1] "GSM836270" "GSM836271" "GSM836272" "GSM836273" "GSM847561" "GSM847562"
```

```r
colData(rse_gene_SRP009615)$title, 2
```

```r
## [1] "K562 cells with shRNA targeting SRF gene cultured with no
doxycline (uninduced - UI), repl."
## [2] "K562 cells with shRNA targeting SRF gene cultured with
doxycline for 48 hours (48 hr), repl."
```

```r
colData(rse_gene_SRP009615)$characteristics, 2
```

```r
## CharacterList of length 2
## [[1]] cells: K562 shRNA expression: no treatment: Puromycin
## [[2]] cells: K562 shRNA expression: yes, targeting SRF treatment: Puromycin, doxycycline
```

```r
## Similar but not exactly the same wording used for two different samples
colData(rse_gene_SRP009615)$characteristics[[1]]
```

```r
## [1] "cells: K562"   "shRNA expression: no" "treatment: Puromycin"
```

```r
colData(rse_gene_SRP009615)$characteristics[[11]]
```

```r
## [1] "cell line: K562"
## [2] "shRNA expression: no shRNA expression"
## [3] "treatment: Puromycin"
```
## Extract the target information

```r
target <- sapply(colData(rse_gene_SRP009615)$characteristics, "[", 2)
target
```

```
[1] "shRNA expression: no"
[2] "shRNA expression: yes, targeting SRF"
[3] "shRNA expression: no"
[4] "shRNA expression: yes targeting SRF"
[6] "shRNA expression: expressing shRNA targeting EGR1"
[7] "shRNA expression: no shRNA expression"
[8] "shRNA expression: expressing shRNA targeting EGR1"
[9] "shRNA expression: no shRNA expression"
[10] "shRNA expression: expressing shRNA targeting ATF3"
[12] "shRNA expression: expressing shRNA targeting ATF3"
```

## Build a useful factor vector, set the reference level and append the result to the `colData()` slot

```r
target_factor <- sapply(strsplit(target, "targeting "), "[", 2)
target_factor[is.na(target_factor)] <- "none"
target_factor <- factor(target_factor)
target_factor <- relevel(target_factor, "none")
target_factor
```

```
[1] none SRF  none SRF none EGR1 none EGR1 none ATF3 none ATF3
Levels: none ATF3 EGR1 SRF
colData(rse_gene_SRP009615)$target_factor <- target_factor
```

As shown in Figure 2, we can expand the biological metadata information by adding predictions based on RNA-seq data\textsuperscript{14}. The predictions include information about sex, sample source (cell line vs tissue), tissue and the sequencing strategy used. To add the predictions, simply use the function `add_predictions()` to expand the `colData()` slot.

## Before adding predictions

```r
dim(colData(rse_gene))
```

```
[1] 72 21
```

## Add the predictions

```r
rse_gene <- add_predictions(rse_gene)
```

```
2017-07-30 10:11:20 downloading the predictions to
/var/folders/cx/n9s558kkx6fb7jf52z_pgszgb8000gn/T/RetmpLufhkr/PredictedPhenotypes_v0.0.03.rda
```

## After adding the predictions

```r
dim(colData(rse_gene))
```

```
[1] 72 33
```

## Explore the variables

```r
colData(rse_gene), 22:ncol(colData(rse_gene))]
```

```
DataFrame with 72 rows and 12 columns
## reported_sex predicted_sex accuracy_sex reported_samplesource
## <factor> <factor> <numeric> <factor>
## SRR2071341 female female 0.8428571 NA
## SRR2071345 male male 0.8428571 NA
## SRR2071346 male male 0.8428571 NA
## SRR2071347 female female 0.8428571 NA
## SRR2071348 female female 0.8428571 NA
## ... ... ... ... ... ...
## SRR1554541 male female 0.8428571 NA
## SRR1554554 female female 0.8428571 NA
## SRR1554535 male male 0.8428571 NA
```
Adding more information Ultimately, more sample metadata information could be available elsewhere, which


For our example use case, project SRP045638 has a few extra biologically relevant variables via the SRA Run selector
https://trace.ncbi.nlm.nih.gov/Traces/study/?acc=SRP045638. We can download that information into text file named SraRunTable.txt by default, then load it into R, sort it appropriately and then append it to the colData() slot. Below we do so for the SRP045638 project.

```r
## Save the information from
## https://trace.ncbi.nlm.nih.gov/Traces/study/?acc=SRP045638
to a table. We saved the file as SRP045638/SraRunTable.txt.
file.exists(file.path("SRP045638", "SraRunTable.txt"))
## [1] TRUE
## Read the table
sra <- read.table(file.path("SRP045638", "SraRunTable.txt"),
                 header = TRUE, sep = "\t")
## Explore it
head(sra)
```
## AssemblyName_s AvgSpotLen_l BioSample_s Experiment_s
## 1         GRCh37          179 SAMN02731372    SRX683791
## 2         GRCh37          179 SAMN02731373    SRX683792
## 3         GRCh37          171 SAMN02999518    SRX683793
## 4         GRCh37          184 SAMN02999519    SRX683794
## 5         GRCh37          182 SAMN02999520    SRX683795
## 6         GRCh37          185 SAMN02999521    SRX683796

## Library_Name_s LoadDate_s MBases_l MBytes_l RIN_s
## 1 R2835_DLPFC_polyA_RNAseq_total 2014-08-21     6452     3571   8.3
## 2 R2857_DLPFC_polyA_RNAseq_total 2014-08-21     6062     2879   8.4
## 3 R2869_DLPFC_polyA_RNAseq_total 2014-08-21     8696     4963   8.7
## 4 R3098_DLPFC_polyA_RNAseq_total 2014-08-21     4479     2643   5.3
## 5 R3452_DLPFC_polyA_RNAseq_total 2014-08-21    11634     6185   9.6
## 6 R3462_DLPFC_polyA_RNAseq_total 2014-08-21    14050     7157   6.4

## ReleaseDate_s      Run_s SRA_Sample_s Sample_Name_s   age_s disease_s
## 1    2014-11-13 SRR1554533    SRS686961   R2835_DLPFC 67.7800   Control
## 2    2014-11-13 SRR1554534    SRS686962   R2857_DLPFC 40.4200   Control
## 3    2014-11-13 SRR1554535    SRS686963   R2869_DLPFC 41.5800   control
## 4    2014-11-13 SRR1554536    SRS686964   R3098_DLPFC 44.1700   control
## 5    2014-11-13 SRR1554537    SRS686965   R3452_DLPFC -0.3836   control
## 6    2014-11-13 SRR1554538    SRS686966   R3462_DLPFC -0.4027   control

## isolate_s race_s  sex_s Assay_Type_s BioProject_s BioSampleModel_s
## 1     DLPFC     AA female      RNA-Seq  PRJNA245228            Human
## 2     DLPFC     AA   male      RNA-Seq  PRJNA245228            Human
## 3     R2869     AA   male      RNA-Seq  PRJNA245228            Human
## 4     R3098     AA female      RNA-Seq  PRJNA245228            Human
## 5     R3452     AA female      RNA-Seq  PRJNA245228            Human
## 6     R3462     AA female      RNA-Seq  PRJNA245228            Human

## Consent_s Fraction_s InsertSize_l        Instrument_s LibraryLayout_s
## 1    public      total            0 Illumina HiSeq 2000          PAIRED
## 2    public      total            0 Illumina HiSeq 2000          PAIRED
## 3    public      total            0 Illumina HiSeq 2000          PAIRED
## 4    public      total            0 Illumina HiSeq 2000          PAIRED
## 5    public      total            0 Illumina HiSeq 2000          PAIRED
## 6    public      total            0 Illumina HiSeq 2000          PAIRED

## LibrarySelection_s LibrarySource_s   Organism_s Platform_s SRA_Study_s
## 1               cDNA  TRANSCRIPTOMIC Homo sapiens   ILLUMINA   SRP045638
## 2               cDNA  TRANSCRIPTOMIC Homo sapiens   ILLUMINA   SRP045638
## 3               cDNA  TRANSCRIPTOMIC Homo sapiens   ILLUMINA   SRP045638
## 4               cDNA  TRANSCRIPTOMIC Homo sapiens   ILLUMINA   SRP045638
## 5               cDNA  TRANSCRIPTOMIC Homo sapiens   ILLUMINA   SRP045638
## 6               cDNA  TRANSCRIPTOMIC Homo sapiens   ILLUMINA   SRP045638

## biomaterial_provider_s tissue_s
## 1                   LIBD    DLPFC
## 2                   LIBD    DLPFC
## 3                   LIBD    DLPFC
## 4                   LIBD    DLPFC
## 5                   LIBD    DLPFC
## 6                   LIBD    DLPFC

We will remove some trailing '_s' from the variable names
colnames(sra) <- gsub("_s$", "", colnames(sra))

Choose some variables we want to add
sra_vars <- c("sex", "race", "RIN", "age", "isolate", "disease", "tissue")

Re-organize the SRA table based on the SRA Run IDs we have
sra <- sra[match(colData(rse_gene)$run, sra$Run), ]

Double check the order
identical(colData(rse_gene)$run, as.character(sra$Run))

(1) TRUE

Append the variables of interest
colData(rse_gene) <- cbind(colData(rse_gene), sra[, sra_vars])
## Final dimensions

\[
\text{dim(colData(rse_gene))}
\]

\[
[1] 72 40
\]

## Explore result

\[
\text{colData(rse_gene)[, 34:ncol(colData(rse_gene))]
}\]

## DataFrame with 72 rows and 7 columns

<table>
<thead>
<tr>
<th>sex</th>
<th>race</th>
<th>RIN</th>
<th>age</th>
<th>isolate</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>female</td>
<td>AA</td>
<td>8.3</td>
<td>67.78</td>
<td>DLPFC</td>
<td>Control</td>
</tr>
<tr>
<td>male</td>
<td>AA</td>
<td>8.4</td>
<td>40.42</td>
<td>DLPFC</td>
<td>Control</td>
</tr>
<tr>
<td>male</td>
<td>AA</td>
<td>8.7</td>
<td>41.58</td>
<td>R2869</td>
<td>control</td>
</tr>
<tr>
<td>female</td>
<td>AA</td>
<td>9.6</td>
<td>-0.38</td>
<td>R3452</td>
<td>control</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>male</td>
<td>CAUC</td>
<td>9.1</td>
<td>16.70</td>
<td>R4028</td>
<td>control</td>
</tr>
<tr>
<td>male</td>
<td>CAUC</td>
<td>8.4</td>
<td>0.39</td>
<td>R3652</td>
<td>control</td>
</tr>
</tbody>
</table>

## tissue

<table>
<thead>
<tr>
<th>DLPFC</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLPFC</td>
</tr>
<tr>
<td>DLPFC</td>
</tr>
<tr>
<td>DLPFC</td>
</tr>
<tr>
<td>DLPFC</td>
</tr>
<tr>
<td>DLPFC</td>
</tr>
</tbody>
</table>

Since we have the predicted sex as well as the reported sex via the SRA Run Selector, we can check whether they match.

\[
\text{table("Predicted" = colData(rse_gene)predicted_sex,}
\]

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>female</td>
<td>male</td>
</tr>
<tr>
<td>24</td>
<td>8</td>
</tr>
<tr>
<td>male</td>
<td>0</td>
</tr>
<tr>
<td>Unassigned</td>
<td>0</td>
</tr>
</tbody>
</table>

## DE setup

Now that we have all the metadata available we can perform a DE analysis. The original study for project SRP045638 looked at differences between 6 age groups: prenatal, infant, child, teen, adult and late life. The following code creates these six age groups.

\[
\text{colData(rse_gene)$age_group <- factor}\]

ifelse(colData(rse_gene)$age < 0, "prenatal",
  ifelse(colData(rse_gene)$age >= 0 & colData(rse_gene)$age < 1, "infant",
    ifelse(colData(rse_gene)$age >= 1 & colData(rse_gene)$age < 10, "child",
      ifelse(colData(rse_gene)$age >= 10 & colData(rse_gene)$age < 20, "teen",
        ifelse(colData(rse_gene)$age >= 20 & colData(rse_gene)$age < 50, "adult",
          "late life")))))

levels = c("prenatal", "infant", "child", "teen", "adult", "late life")}
Most of the DE signal from the original study was between the prenatal and postnatal samples. To simplify the analysis, we will focus on this comparison.

```r
## Create prenatal factor
colData(rse_gene)$prenatal <- factor(
    ifelse(colData(rse_gene)$age_group == "prenatal", "prenatal", "postnatal"),
    levels = c("prenatal", "postnatal"))
```

As we saw earlier in Figure 9, it is important to scale the coverage counts to read counts. To highlight the fact that we scaled the counts, we will use a new object name and delete the previous one. However, in practice we would simply overwrite the `rse` object with the output of `scale_counts(rse)`.

```r
## Scale counts
rse_gene_scaled <- scale_counts(rse_gene)
```

Having scaled the counts, we then filter out genes that are lowly expressed and extract the count matrix.

```r
## To highlight that we scaled the counts
rm(rse_gene)
```

DE analysis
Now that we have scaled the counts, there are multiple DE packages we could use, as described elsewhere\(^2\). Since we have 12 samples per group, which is a moderate number, we will use `limma-voom\(^1\)` due to its speed. The model we use tests for DE between prenatal and postnatal samples adjusting for sex and RIN, which is a measure of quality of the input sample. We check the data with multi-dimensional scaling plots (Figure 13 and Figure 14) as well as the mean-variance plot (Figure 15). In a real use case we might have to explore the results with different models and perform sensitivity analyses.

![Figure 13. Multi-dimensional scaling plot of the gene level data by age group.](image)
library("limma")
library("edgeR")

## Build DGEList object
dge <- DGEList(counts = counts[filter, ])

## Calculate normalization factors
dge <- calcNormFactors(dge)

## Explore the data
plotMDS(dge, labels = substr(colData(rse_gene_scaled)$prenatal, 1, 2) )
plotMDS(dge, labels = substr(colData(rse_gene_scaled)$sex, 1, 1) )

tapply(colData(rse_gene_scaled)$RIN, colData(rse_gene_scaled)$prenatal, summary)

## $prenatal
##    Min. 1st Qu. Median  Mean 3rd Qu.  Max.
##   5.700   6.400  8.150 7.767   8.600 9.600
##
## $postnatal
##    Min. 1st Qu. Median  Mean 3rd Qu.  Max.
##   5.300   8.100  8.300 8.197   8.700 9.100

## Specify our design matrix
design <- with(colData(rse_gene_scaled), model.matrix(~ sex + RIN + prenatal))

## Run voom
v <- voom(dge, design, plot = TRUE)

## Run remaining parts of the DE analysis
fit <- lmFit(v, design)
fit <- eBayes(fit)

Having run the DE analysis, we can explore some of the top results either with an MA plot (Figure 16) and a volcano plot Figure (17). Both reveal very strong and widespread DE signal.

## Visually explore DE results
limma::plotMA(fit, coef = 4)

limma::volcanoplot(fit, coef = 4)

![MA plot of the gene level data](image)

**Figure 16.** MA plot of the gene level data. Testing for prenatal and postnatal DE adjusting for sex and RIN.
Now that we have the DE results, we can use some of the tools with the biocView ReportWriting to create a report. One of them is regionReport, which can create reports from DESeq2 and edgeR results. It can also handle limma-voom results by making them look like DESeq2 results. To do so, we need to extract the relevant information from the limma-voom objects using topTable() and build DESeqDataSet and DESeqResults objects as shown below. A similar conversion is needed to use ideal, which is another package in the ReportWriting biocView category.

```r
## Extract data from limma-voom results
top <- topTable(fit, number = Inf, sort.by = "none", coef = "prenatalpostnatal")
## Build a DESeqDataSet with the count data and model we used
library("DESeq2")
dds <- DESeqDataSet(rse_gene_scaled[filter, ], ~ sex + RIN + prenatal)
## converting counts to integer mode
## Add gene names keeping only the Ensembl part of the Gencode IDs
rownames(dds) <- gsub("\..*", "", rownames(dds))
## Build a DESeqResults object with the relevant information
## Note that we are transforming the baseMean so it will look ok
## with DESeq2's plotting functions.
limma_res <- DESeqResults(DataFrame(pvalue = top[, "P.Value"],
                      log2FoldChange = top[, "logFC"],
                      baseMean = exp(top[, "AveExpr"]),
                      padj = top[, "adj.P.Val"]))
rownames(limma_res) <- rownames(dds)
## Specify FDR cutoff to use
metadata(limma_res)[["alpha"]] <- 0.001
## Add gene symbols so they will be displayed in the report
limma_res$symbol <- rowRanges(rse_gene_scaled)$symbol[filter]
## Some extra information used by the report function
mcols(dds) <- limma_res
mcols(mcols(dds)) <- DataFrame(type = "results",
                         description = "manual incomplete conversion from limma-voom to DESeq2")
```

Figure 17. Volcano plot of the gene level data. Testing for prenatal and postnatal DE adjusting for sex and RIN.
Having converted our limma-voom results to DESeq2 results, we can now create the report, which should open automatically in a browser.

```r
library("regionReport")
## This takes about 20 minutes to run
report <- DESeq2Report(dds, project = "SRP045638 gene results with limma-voom",
                         output = "gene_report", outdir = "SRP045638",
                         intgroup = c("prenatal", "sex"), res = limma_res, software = "limma")
```

If the report doesn’t open automatically, we can open it with `browseURL()`. A pre-computed version is available as Supplementary File 1.

`browseURL(file.path("SRP045638", "gene_report.html"))`

**GO enrichment**

Using `clusterProfiler` we can then perform several enrichment analyses using the Ensembl gene IDs. Here we show how to perform an enrichment analysis using the biological process ontology (Figure 18).

![Figure 18. Biological processes enriched in the DE genes.](image-url)
library("clusterProfiler")
library("org.Hs.eg.db")

## Remember that limma_res had ENSEMBL IDs for the genes
head(rownames(limma_res))
## [1] "ENSG00000000003" "ENSG00000000005" "ENSG00000000419" "ENSG00000000457"
## [5] "ENSG00000000460" "ENSG00000000938"

## Perform enrichment analysis for Biological Process (BP)
## Note that the argument is keytype instead of keyType in Bioconductor 3.5
enrich_go <- enrichGO(gene = rownames(limma_res)[limma_res$padj < 0.001],
  OrgDb = org.Hs.eg.db, keyType = "ENSEMBL", ont = "BP",
  pAdjustMethod = "BH", pvalueCutoff = 0.01, qvalueCutoff = 0.05,
  universe = rownames(limma_res))

## Visualize enrichment results
dotplot(enrich_go, font.size = 7)

Several other analyses can be performed with the resulting list of differentially expressed genes as described previously\(^1\), although that is beyond the scope of this workflow.

Other features
As described in Figure 1, recount2 provides data for expression features beyond genes. In this section we perform a DE analysis using exon data as well as the base-pair resolution information.

Exon and exon-exon junctions
The exon and exon-exon junction coverage count matrices are similar to the gene level one and can also be downloaded with download_study(). However, these coverage count matrices are much larger than the gene one. Aggressive filtering of lowly expressed exons or exon-exon junctions can reduce the matrix dimensions if this impacts the performance of the DE software used.

Below we repeat the gene level analysis for the disjoint exon data. We first download the exon data, add the expanded metadata we constructed for the gene analysis, explore the data (Figure 19), and then perform the DE analysis using limma-voom.

![voom: Mean-variance trend](image)

**Figure 19.** voom mean-variance plot of the exon level data.
## Download the data if it is not there

```r
if (!file.exists(file.path("SRP045638", "rse_exon.Rdata"))) {
  download_study("SRP045638", type = "rse-exon")
}
```

## 2017-07-30 10:37:11 downloading file rse_exon.Rdata to SRP045638

### Load the data

```r
load(file.path("SRP045638", "rse_exon.Rdata"))
```

### Scale and add the metadata (it is in the same order)

```r
identical(colData(rse_exon)$run, colData(rse_gene_scaled)$run)
```

```r
[1] TRUE
```

```
colData(rse_exon) <- colData(rse_gene_scaled)
```

```
rse_exon_scaled <- scale_counts(rse_exon)
```

### Filter lowly expressed exons

```r
filter_exon <- rowMeans(assays(rse_exon_scaled)$counts) > 0.5
```

```r
round(table(filter_exon) / length(filter_exon) * 100, 2)
```

```r
# FALSE TRUE
# 29.08 70.92
```

### Build DGEList object

```r
dge_exon <- DGEList(counts = assays(rse_exon_scaled)$counts[filter_exon, ])
```

### Calculate normalization factors

```r
dge_exon <- calcNormFactors(dge_exon)
```

### Run voom

```r
v_exon <- voom(dge_exon, design, plot = TRUE)
```

### Run remaining parts of the DE analysis

```r
fit_exon <- lmFit(v_exon, design)
```

```r
fit_exon <- eBayes(fit_exon)
```

### Visualize inspect results

```r
limma::volcanoplot(fit_exon, coef = 4)
```

### Get p-values and other statistics

```r
top_exon <- topTable(fit_exon, number = Inf, sort.by = "none",
  coef = "prenatalpostnatal")
table(top_exon$adj.P.Val < 0.001)
```

```r
# FALSE TRUE
# 107303 126075
```

Just like at the gene level, we see many exons differentially expressed between prenatal and postnatal samples (Figure 20). As a first step to integrate the results from the two features, we can compare the list of genes that are differentially expressed versus the genes that have at least one exon differentially expressed.

```r
# Get the gene IDs for genes that are DE at the gene level or that have at least one exon with DE signal.
```

```r
genes_w_de_exon <- unique(rownames(rse_exon_scaled)[top_exon$adj.P.Val < 0.001])
genes_de <- rownames(rse_gene_scaled)[which(filter)[top$adj.P.Val < 0.001]]
```
### Make a venn diagram

```r
library("gplots")

vinfo <- venn(list("genes" = genes_de, "exons" = genes_w_de_exon),
               names = c("genes", "exons"), show.plot = FALSE)

plot(vinfo) +
     title("Genes/exons with DE signal")
```

Not all differentially expressed genes have differentially expressed exons. Moreover, genes with at least one differentially expressed exon are not necessarily differentially expressed (Figure 21). This is in line with what was described in Figure 2B of Soneson et al., 2015.

This was just a quick example of how we can perform DE analyses at the gene and exon feature levels. We envision that more involved pipelines could be developed that leverage both feature levels, such as in Jaffe et al., 2017. For instance, we could focus on the differentially expressed genes with at least one differentially expressed exon, and compare the direction of the DE signal versus the gene level signal as shown in Figure 22.

---

**Figure 20.** Volcano plot of the exon level data. Testing for prenatal and postnatal DE adjusting for sex and RIN.

**Figure 21.** Venn diagram of the overlap between DE genes and genes with at least one exon DE.
Keep only the DE exons that are from a gene that is also DE

```r
## Keep only the DE exons that are from a gene that is also DE
top_exon_de <- top_exon[top_exon$adj.P.Val < 0.001 &
top_exon$ID %in% attr(vinfo, "intersections")[["genes:exons"]], ]
```

Find the fold change that is the most extreme among the DE exons of a gene

```r
## Find the fold change that is the most extreme among the DE exons of a gene
exon_max_fc <- tapply(top_exon_de$logFC, top_exon_de$ID, function(x) {
    x[which.max(abs(x))]
})
```

Keep only the DE genes that match the previous selection

```r
top_gene_de <- top[match(names(exon_max_fc), rownames(top)), ]
```

Make the plot

```r
## Make the plot
plot(top_gene_de$logFC, exon_max_fc, pch = 20, col = adjustcolor("black", 1/5),
     ylab = "Most extreme exon log FC",
     xlab = "Gene log FC",
     main = "DE genes with at least one DE exon")
abline(a = 0, b = 1, col = "red")
abline(h = 0, col = "grey80")
abline(v = 0, col = "grey80")
```

The fold change for most exons shown in Figure 22 agrees with the gene level fold change. However, some of them have opposite directions and could be interesting to study further.

**Base-pair resolution**

recount2 provides bigWig coverage files (unscaled) for all samples, as well as a mean bigWig coverage file per project where each sample was scaled to 40 million 100 base-pair reads. The mean bigWig files are exactly what is needed to start an *expressed regions* analysis with derfinder⁶. recount provides two related functions: `expressed_regions()` which is used to define a set of regions based on the mean bigWig file for a given project, and `coverage_matrix()` which based on a set of regions builds a count coverage matrix in a *RangedSummarizedExperiment* object just like the ones that are provided for genes and exons. Both functions ultimately use `import.bw()` from *rtracklayer*²⁴ which currently is not supported on Windows machines. While this presents a portability disadvantage, on the other side it allows reading portions of bigWig files from the web without having to fully download them. `download_study()` with `type = "mean"` or `type = "samples"` can be used to download the bigWig files, which we recommend doing when working with them extensively.
For illustrative purposes, we will use the data from chromosome 21 for the SRP045638 project. First, we obtain the expressed regions using a relatively high mean cutoff of 5. We then filter the regions to keep only the ones longer than 100 base-pairs to shorten the time needed for running `coverage_matrix()`.

```r
## Define expressed regions for study SRP045638, only for chromosome 21
regions <- expressed_regions("SRP045638", "chr21", cutoff = 5L, maxClusterGap = 3000L)
```

```r
## 2017-07-30 10:39:06 loadCoverage: loading bigWig file
## http://duffel.rail.bio/recount/SRP045638/bw/mean_SRP045638.bw
## 2017-07-30 10:39:16 loadCoverage: applying the cutoff to the merged data
## 2017-07-30 10:39:16 filterData: originally there were 46709983 rows,
## now there are 46709983 rows. Meaning that 0 percent was filtered.
## 2017-07-30 10:39:16 findRegions: identifying potential segments
## 2017-07-30 10:39:16 .getSegmentsRle: segmenting with cutoff(s) 5
## 2017-07-30 10:39:17 findRegions: identifying candidate regions
## 2017-07-30 10:39:17 findRegions: identifying region clusters
## Explore the resulting expressed regions

## GRanges object with 3853 ranges and 6 metadata columns:
## seqnames ranges strand | value
## <Rle> <IRanges> <Rle> | <numeric>
## 1 chr21 [5026549, 5026630] * | 6.48181250037217
## 2 chr21 [5027935, 5027961] * | 6.19690331706294
## 3 chr21 [5028108, 5028225] * | 8.99329216197386
## 4 chr21 [5032053, 5032117] * | 7.06828071887676
## 5 chr21 [5032148, 5032217] * | 6.48832686969212
## ...                      ... ... ... .              ...
## 3849 chr21 [46695774, 46695774] * | 5.0290150642395
## 3850 chr21 [46695784, 46695843] * | 5.38047295411428
## 3851 chr21 [46695865, 46695869] * | 5.1128270149231
## 3852 chr21 [46696463, 46696486] * | 5.25689166784286
## 3853 chr21 [46696508, 46696534] * | 5.22988386507387
## area indexStart IndexEnd cluster clusterL
## <numeric> <integer> <integer> <Rle>    <Rle>
## 1 531.508625030518 5026549 5026630 1 1677
## 2 167.316389560699 5027935 5027961 1 1677
## 3 1061.20847511292 5028108 5028225 1 1677
## 4 459.43824672699 5032053 5032117 2 8283
## 5 454.182880878448 5032148 5032217 2 8283
## ...                      ... ... ... ... ... ...
## 3849 5.0290150642395 46695774 46695774 708 7508
## 3850 322.828377246857 46695784 46695843 708 7508
## 3851 25.5641350746155 46695865 46695869 708 7508
## 3852 126.165490028229 46696463 46696486 708 7508
## 3853 141.206864356995 46696508 46696534 708 7508
## seqinfo: 1 sequence from an unspecified genome

summary(width(regions))
## Min. 1st Qu.  Median   Mean 3rd Qu.    Max.
## 1.0    6.0    68.0 186.2   151.0 11709.0
```
table(width(regions) >= 100)

## FALSE TRUE
## 2284 1569

## Keep only the ones that are at least 100 bp long
regions <- regions[width(regions) >= 100]
length(regions)
## [1] 1569

Now that we have a set of regions to work with, we proceed to build a `RangedSummarizedExperiment` object with the coverage counts, add the expanded metadata we built for the gene level, and scale the counts. Note that `coverage_matrix()` scales the base-pair coverage counts by default, which we turn off in order to use `scale_counts()`.

## Compute coverage matrix for study SRP045638, only for chromosome 21
## Takes about 4 minutes
rse_er <- coverage_matrix("SRP045638", "chr21", regions, chunksize = 2000,
   verboseLoad = FALSE, scale = FALSE)

## 2017-07-30 10:39:19 railMatrix: processing regions 1 to 1569

## Use the expanded metadata we built for the gene model
colData(rse_er) <- colData(rse_gene_scaled)

## Scale the coverage matrix
rse_er_scaled <- scale_counts(rse_er)

## To highlight that we scaled the counts
rm(rse_er)

Now that we have a scaled count matrix for the expressed regions, we can proceed with the DE analysis just like we did at the gene and exon feature levels (Figure 23, Figure 24, Figure 25, and Figure 26).

**Figure 23.** Multi-dimensional scaling plot of the expressed regions level data by age group.
Figure 24. Multi-dimensional scaling plot of the expressed regions level data by sex.

Figure 25. voom mean-variance plot of the expressed regions level data.
Figure 26. Volcano plot of the expressed regions level data. Testing for prenatal and postnatal DE adjusting for sex and RIN.

```r
## Build DGEList object
dge_er <- DGEList(counts = assays(rse_er_scaled)$counts)

## Calculate normalization factors
dge_er <- calcNormFactors(dge_er)

## Explore the data
plotMDS(dge_er, labels = substr(colData(rse_er_scaled)$prenatal, 1, 2) )
plotMDS(dge_er, labels = substr(colData(rse_er_scaled)$sex, 1, 1) )

## Run voom
v_er <- voom(dge_er, design, plot = TRUE)

## Run remaining parts of the DE analysis
fit_er <- lmFit(v_er, design)
fit_er <- eBayes(fit_er)

## Visually explore the results
limma::volcanoplot(fit_er, coef = 4)

## Number of DERs
top_er <- topTable(fit_er, number = Inf, sort.by = "none",
                   coef = "prenatalpostnatal")
table(top_er$adj.P.Val < 0.001)

## FALSE  TRUE
## 609   960

Having identified the differentially expressed regions (DERs), we can sort all regions by their adjusted p-value.

```r
## Sort regions by q-value
regions_by_padj <- regions[order(top_er$adj.P.Val, decreasing = FALSE)]

## Look at the top 10
regions_by_padj[1:10]
```
Visualize regions
Since the DERs do not necessarily match the annotation, it is important to visualize them. The code for visualizing DERs can easily be adapted to visualize other regions. Although, the width and number of the regions will influence the computing resources needed to make the plots.

Because the unscaled bigWig files are available in recount2, several visualization packages can be used such as epivizr, wiggleplotr and derfinderPlot. With all of them it is important to remember to scale the data except when visualizing the mean bigWig file for a given project.

First, we need to get the list of URLs for the bigWig files. We can either manually construct them or search them inside the recount_url table.

## Construct the list of bigWig URLs
They have the following form:
http://duffel.rail.bio/recount/
/project id
/bw/
sample run id
/bw
bws <- paste0("http://duffel.rail.bio/recount/SRP045638/bw/",
colData(rse_er_scaled)$bigwig_file)

## Use the sample run IDs as the sample names
names(bws) <- colData(rse_er_scaled)$run

We visualize the DERs using derfinderPlot, similar to what was done in the original publication. However, we first add a little padding to the regions: 100 base-pairs on each side.
## Add 100 bp padding on each side
regions_resized <- resize(regions_by_padj[1:10],
    width(regions_by_padj[1:10]) + 200, fix = "center")

Next, we obtain the base-pair coverage data for each DER and scale the data to a library size of 40 million 100 base-pair reads, using the coverage AUC information we have in the metadata.

## Get the bp coverage data for the plots
library("derfinder")
regionCov <- getRegionCoverage(regions = regions_resized, files = bws,
    targetSize = 40 * 1e6 * 100, totalMapped = colData(rse_er_scaled)$auc,
    verbose = FALSE)

The function plotRegionCoverage() requires several pieces of annotation information for the plots that use a TxDb object. For recount2 we used Gencode v25 hg38’s annotation, which means that we need to process it manually instead of using a pre-computed TxDb package.

To create a TxDb object for Gencode v25, first we need to import the data. Since we are working only with chromosome 21 for this example, we can subset it. Next we need to add the relevant chromosome information. Some of the annotation functions we use can handle Entrez or Ensembl IDs, but not Gencode IDs. So we will make sure that we are working with Ensembl IDs before finally creating the Gencode v25 TxDb object.

## Import the Gencode v25 hg38 gene annotation
library("rtracklayer")
    "gencode.v25.annotation.gtf.gz"))

## Keep only the chr21 info
gencode_v25_hg38 <- keepSeqlevels(gencode_v25_hg38, "chr21",
    pruning.mode="coarse")

## Get the chromosome information for hg38
library("GenomicFeatures")
chrInfo <- getChromInfoFromUCSC("hg38")

## Download and preprocess the 'chrominfo' data frame ...
## OK
chrInfo$chrom <- as.character(chrInfo$chrom)
chrInfo <- chrInfo[chrInfo$chrom %in% seqlevels(regions), ]
chrInfo$isCircular <- FALSE

## Assign the chromosome information to the object we will use to create the txdb object
si <- with(chrInfo, Seqinfo(as.character(chrom), length, isCircular,
    genome = "hg38"))
seqinfo(gencode_v25_hg38) <- si

## Switch from Gencode gene IDs to Ensembl gene IDs
gencode_v25_hg38$gene_id <- gsub("\..*,", ",", gencode_v25_hg38$gene_id)

## Create the TxDb object
gencode_v25_hg38_txdb <- makeTxDbFromGRanges(gencode_v25_hg38)

## Explore the TxDb object
gencode_v25_hg38_txdb

## TxDb object:
## # Db type: TxDb
## # Supporting package: GenomicFeatures
## # Genome: hg38
## # transcript_nrow: 2413
## # exon_nrow: 7670
Now that we have a TxDb object for Gencode v25 on hg38 coordinates, we can use `bumphunter`'s annotation functions for annotating the original 10 regions we were working with. Since we are using Ensembl instead of Entrez gene IDs, we need to pass this information to `annotateTranscripts()`. Otherwise, the function will fail to retrieve the gene symbols.

```r
library("bumphunter")
#
# Annotate all transcripts for gencode v25 based on the TxDb object
# we built previously.
ann_gencode_v25_hg38 <- annotateTranscripts(gencode_v25_hg38_txdb,
   annotationPackage = "org.Hs.eg.db",
   mappingInfo = list("column" = "ENTREZID", "keytype" = "ENSEMBL",
   "multiVals" = "first"))
```

## Getting TSS and TSE.
## Getting CSS and CSE.
## Getting exons.
## Annotating genes.
## 'select()' returned 1:many mapping between keys and columns

The final piece we need to run is information about which base-pairs are exonic, intronic, etc. This is done via the `annotateRegions()` function in `derfinder`, which itself requires prior processing of the TxDb information by `makeGenomicState()`.

```r
## Create the genomic state object using the gencode TxDb object
gs_gencode_v25_hg38 <- makeGenomicState(gencode_v25_hg38_txdb,
   chrs = seqlevels(regions))
## 'select()' returned 1:1 mapping between keys and columns

## Annotate the original regions
regions_ann <- annotateRegions(regions_resized,
   gs_gencode_v25_hg38$fullGenome)
 2017-07-30 10:50:35 annotateRegions: counting
 2017-07-30 10:50:35 annotateRegions: annotating
```

We can finally use `plotRegionCoverage()` to visualize the top 10 regions coloring by whether they are prenatal or postnatal samples. Known exons are shown in dark blue, introns in light blue.

```r
library("derfinderPlot")
plotRegionCoverage(regions = regions_resized, regionCoverage = regionCov,
   groupInfo = colData(rse_er_scaled)$prenatal,
   nearestAnnotation = nearest_ann,
   annotatedRegions = regions_ann,
   txdb = gencode_v25_hg38_txdb,
   scalefac = 1, ylab = "Coverage (RP40M, 100bp)",
   ask = FALSE, verbose = FALSE)
```
In these plots we can see that some DERs match known exons (Figure 28, Figure 34, Figure 36), some are longer than known exons (Figure 27, Figure 33, Figure 35), and others are exon fragments (Figure 29–Figure 32) which could be due to the cutoff used. Note that Figure 33 could be shorter than a known exon due to a coverage dip.

**Figure 27.** Base-pair resolution plot of differentially expressed region 1.

**Figure 28.** Base-pair resolution plot of differentially expressed region 2.

**Figure 29.** Base-pair resolution plot of differentially expressed region 3.
Figure 30. Base-pair resolution plot of differentially expressed region 4.

Figure 31. Base-pair resolution plot of differentially expressed region 5.

Figure 32. Base-pair resolution plot of differentially expressed region 6.
Figure 33. Base-pair resolution plot of differentially expressed region 7.

Figure 34. Base-pair resolution plot of differentially expressed region 8.

Figure 35. Base-pair resolution plot of differentially expressed region 9.
Summary
In this workflow we described in detail the available data in recount2, how the coverage count matrices were computed, the metadata included in recount2 and how to get new phenotypic information from other sources. We showed how to perform a DE analysis at the gene and exon levels as well as use an annotation-agnostic approach. Finally, we explained how to visualize the base-pair information for a given set of regions. This workflow constitutes a strong basis to leverage the recount2 data for human RNA-seq analyses.

Competing interests
No competing interests were disclosed.

Grant information
LCT and AEJ were supported by the National Institutes of Health (grant R21 MH109956-01). LCT and AN were supported by the National Institutes of Health (grant R01 GM105705).

The author confirms that the funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Acknowledgments
We would like to acknowledge the members of Andrew Jaffe (Lieber Institute for Brain Development, Johns Hopkins Medical Campus) and Alexis Battle (Department of Computer Science, Whiting School of Engineering at Johns Hopkins University) labs for feedback on the explanatory figures.

Supplementary material
Supplementary File 1: A pre-computed version of the differential expression report.
Click here to access the data.

Supplementary File 2: Session information. This workflow was created using BiocWorkflowTools. The session information is available in this file. The most recent version of this workflow is available via Bioconductor at http://bioconductor.org/help/workflows/.
Click here to access the data.
References


Open Peer Review

Current Referee Status:  

Version 1

Referee Report 12 September 2017
doi: 10.5256/f1000research.13233.r25456

Davide Risso  
Division of Biostatistics and Epidemiology, Department of Healthcare Policy and Research, Weill Cornell Medicine, New York, NY, USA

The authors present a workflow that describes how to analyze the datasets available through the Recount2 project with Bioconductor. Since many of the state-of-the-art methods for the analysis of RNA-seq data are implemented in R and available through the Bioconductor project, this contribution is an important resource for researchers interested in reanalyzing the impressive amount of data that the authors have processed in the Recount2 project.

I have a few comments that hopefully will help improve the workflow.

1. I was a bit confused by the rationale of the scaled coverage counts. And especially on the need for a target library size and the use of scaled counts. Wouldn't it be simpler to divide the coverage by read length (without rescaling)? Wouldn't that result in the actual reads mapped to each region (exon, gene, ...)? I understand that for the derfinder analysis, some rescaling is needed for normalization purposes, but for more "classic" analysis (such as gene- or exon-level differential expression) where the counts are normalized later in the workflow, wouldn't starting from 'coverage/readlength' be a more sensible choice?

2. Sex prediction. This is a really interesting part of the analysis, even though it's not the focus of this workflow. It would be interesting to get the authors' opinion of how to best use this feature on real analyses. For instance, are the 8 misclassified samples likely to be false positives from the classifiers or are they mislabeled samples? What is the recommendation of the authors in such cases? Should these samples be discarded or is there any diagnostics that can be run to make sure that the quality of these samples is not compromised?

3. I think that the authors should give more details on the design matrix. For instance, why did they decide to include RIN and sex? Why is it important to include these variables in the model? More generally, the workflow lacks details on the limma pipeline. I understand that this is not the focus of the authors' work, but it may be confusing for beginners that don't have a direct experience with limma or voom. The authors could for instance refer the reader to the limma workflow for details.

4. Similarly, there is a lack of details on the GO enrichment analysis. Since there are many types of gene-set enrichment analysis, a paragraph could be added with more details and perhaps some references to explain what enrichment analysis is and what types of hypotheses are tested.
5. One important advantage of exon-level differential expression is that it can be used to infer alternative splicing. This can be done with the functions `diffSplice()` and `topSplice()` in limma or with the DEXSeq package. It would be nice to showcase these functions or at least to mention that they exist.

6. Is the annotation in Recount2 stable? Or is it constantly updated when a new version of Gencode is released? If the former, it might make sense to package the `gencode_v25_hg38_txdb` object that the authors create in the workflow so that each user does not have to create it from scratch every time.

Minor comments:

- The authors use throughout the paper `assays(rse)$counts` to access the counts of the `rse` object. Although this is correct, a clearer and more concise way is `assay(rse)` (or `assay(rse, "counts")` if the authors want to explicitly state the name of the assay).
- Section "Coverage counts provided by recount2". The authors say "Although recount can generate count matrices for other annotations using hg38 coordinates." But they never say how this can be done. It would be good to add a paragraph on how to do that (which I presume involves creating an alternative txdb object).

References

Is the rationale for developing the new method (or application) clearly explained?  
Yes

Is the description of the method technically sound?  
Yes

Are sufficient details provided to allow replication of the method development and its use by others?  
Yes

If any results are presented, are all the source data underlying the results available to ensure full reproducibility?  
Yes

Are the conclusions about the method and its performance adequately supported by the findings presented in the article?  
Yes

**Competing Interests:** No competing interests were disclosed.

I have read this submission. I believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.