1. Introduction

Large single-cell RNA sequencing (scRNA-seq) projects usually need to generate data across multiple batches due to logistical constraints. However, the processing of different batches is often subject to uncontrollable differences, e.g., changes in operator, differences in reagent quality. This results in systematic differences in the observed expression in cells from different batches, which we refer to as “batch effects”. Batch effects are problematic as they can be major drivers of heterogeneity in the data, masking the relevant biological differences and complicating interpretation of the results. Computational correction of these effects is critical for eliminating batch-to-batch variation, allowing data across multiple batches to be combined for valid downstream analysis.

In this workflow, we use two relatively simple datasets (Muraro et al. 2016 and Palasantza A et al) to introduce most of the concepts of scRNA-seq data analysis. These two datasets belong to the human pancreas scRNA-seq datasets that have been derived from two different protocols. We will perform the quality control and normalization of two studies separately and then we will merge both these studies to understand how batch differences exist in between two studies due to technical or biological variations. Then we will correct the batch effect from these two studies and will visualize the clusters. Further we will use one of these datasets to identify the differentially expressed genes and build the trajectories.

Trajectories or **Pseudotime analysis** is to study a process where cells change continuously. This includes, for example, many differentiation processes taking place during development: following a stimulus, cells will change from one cell-type to another. Ideally, we would like to monitor the expression levels of an individual cell over time. Unfortunately, such monitoring is not possible with scRNA-seq since the cell is lysed (destroyed) when the RNA is extracted. Instead, we must sample at multiple time-points and obtain snapshots of the gene expression profiles. Since some of the cells will proceed faster along the differentiation than others, each snapshot may contain cells at varying points along the developmental progression. We use statistical methods to order the cells along one or more trajectories which represent the underlying developmental trajectories, this ordering is referred to as “pseudotime”. Here we will consider Diffusion maps for ordering cells according to their pseudotime development. To illustrate the methods, we will be using one of the dataset from pancreatic scRNAseq.

2. Processing of two different datasets

Loading the raw data from two different studies: One from GEO GSE85241 and one from Array-Express E-MTAB-5061.

GSE85241: This dataset was generated by Muraro et al. (2016) using the CEL-seq2 protocol with unique molecular identifiers (UMIs) and ERCC spike-ins. You can read more at https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE85241

E-MTAB-5061: This dataset comprise of the single cells from human pancreas from healthy individuals and type 2 diabetes patients. The dataset is generated by Palasantza A et al. You can read more at https://www.ebi.ac.uk/arrayexpress/experiments/E-MTAB-5061/
The use of the files linked to these studies are in a variable and the files are downloaded.

```
all.urls <- c("ftp://ftp.ncbi.nlm.nih.gov/geo/series/GSE85nnn/GSE85241/suppl/
GSE85241%5Fcellsystems%5Fdataset%5F4donors%5Fupdated%2Ecsv%2Egz",
              "https://www.ebi.ac.uk/arrayexpress/experiments/E-MTAB-5061/fil
es/E-MTAB-5061.processed.1.zip",
              "https://www.ebi.ac.uk/arrayexpress/files/E-MTAB-5061/E-MTAB-50
61.sdrf.txt")

all.basenames <- basename(all.urls)
all.basenames[1] <- "GSE85241_cellsystems_dataset_4donors_updated.csv"

all.modes <- c("wb", "w", "wb", "w")
for (x in seq_along(all.urls)) {
  if (!file.exists(all.basenames[x])) {
    download.file(all.urls[x], all.basenames[x], mode=all.modes[x])
  }
}
```

2.1. Muraro et al, CEL-seq2

2.1.1. Loading the files

```
gse85241.df <- read.table("GSE85241_cellsystems_dataset_4donors_updated.csv",
                          sep='\t', h=TRUE, row.names=1, stringsAsFactors=FAL
SE)
dim(gse85241.df)

## [1] 19140  3072
```

2.1.2. Extraction of Metadata (information about data) for GEO study will be obtained from the column names.

```
donor.names <- sub("^(D[0-9]+).*", "\\1", colnames(gse85241.df))
table(donor.names)
```

```
## donor.names
## D28 D29 D30 D31
## 768 768 768 768
```

```
plate.id <- sub("^D[0-9]+\\.([0-9]+)_.*", "\\1", colnames(gse85241.df))
table(plate.id)
```

```
## plate.id
##   1   2   3   4   5   6   7   8
## 384 384 384 384 384 384 384 384
```

Question 1: *Checkout what are rownames? Are they gene symbols or identifiers (Ensembl or Entrez)?*

Gene symbols were supplied instead of Ensembl or Entrez identifiers. We convert all row names to Ensembl identifiers, removing NA (Absent, Not Available) or duplicated entries (with the exception of spike-in transcripts).

```
gene.symb <- gsub("__chr.*$", "", rownames(gse85241.df))
is.spike <- grepl("^ERCC-", gene.symb)
table(is.spike)
```

```
## is.spike
## FALSE  TRUE
## 19059    81
```

Question 2: *How many spike-ins are there in this data? Why do we need spike-ins for our analysis? Can they be removed?*

We will identify the rows corresponding to ERCC spike-in transcripts from the row names. We store this information in the SingleCellExperiment object for future use. This is necessary as spike-ins require special treatment in downstream steps such as normalization.

```
library(org.Hs.eg.db)
gene.ids <- mapIds(org.Hs.eg.db, keys=gene.symb, keytype="SYMBOL", column="EN
SEMBL")
gene.ids[is.spike] <- gene.symb[is.spike]

keep <- !is.na(gene.ids) & !duplicated(gene.ids)
gse85241.df <- gse85241.df[keep,]
rownames(gse85241.df) <- gene.ids[keep]
summary(keep)
```

```
##    Mode   FALSE    TRUE
## logical    2097   17043
```

We will now create a SingleCellExperiment object to store the counts and metadata together. **SingleCellExperiment** defines a S4 class for storing data from single-cell experiments. This includes specialized methods to store and retrieve spike-in information, dimensionality reduction coordinates and size factors for each cell, along with the usual metadata for genes and libraries.

```
library(SingleCellExperiment)
sce.gse85241 <- SingleCellExperiment(list(counts=as.matrix(gse85241.df)),
                                     colData=DataFrame(Donor=donor.names, Pla
te=plate.id),
                                     rowData=DataFrame(Symbol=gene.symb[keep]
))
isSpike(sce.gse85241, "ERCC") <- grepl("^ERCC-", rownames(gse85241.df))
sce.gse85241
```

```
## class: SingleCellExperiment
## dim: 17043 3072
## metadata(0):
## assays(1): counts
## rownames(17043): ENSG00000268895 ENSG00000121410 ...
##   ENSG00000074755 ENSG00000036549
## rowData names(1): Symbol
## colnames(3072): D28.1_1 D28.1_2 ... D30.8_95 D30.8_96
## colData names(2): Donor Plate
## reducedDimNames(0):
## spikeNames(1): ERCC
```

2.1.3. Quality control and normalization

We compute QC metrics for each cell and identify cells with low library sizes, low numbers of expressed genes, or high ERCC content.
The **calculateQCMetrics** function computes a number of quality control metrics for each cell and feature, stored in the colData and rowData respectively. By default, the QC metrics are computed from the count data, but this can be changed through the exprs_values argument.

```
library(scater)
sce.gse85241 <- calculateQCMetrics(sce.gse85241, compact=TRUE)
all_col_qc <- colnames(colData(sce.gse85241))
all_col_qc <- all_col_qc[grep("ERCC", all_col_qc)]
```

2.1.4. Producing several daignostic Plots for QC

Examining the most expressed features We look at a plot that shows the top 50 (by default) most-expressed features. Each row in the plot below corresponds to a gene, and each bar corresponds to the expression of a gene in a single cell. The circle indicates the median expression of each gene, with which genes are sorted. By default, “expression” is defined using the feature counts (if available), but other expression values can be used instead by changing exprs_values.

```
plotHighestExprs(sce.gse85241, exprs_values = "counts")
```

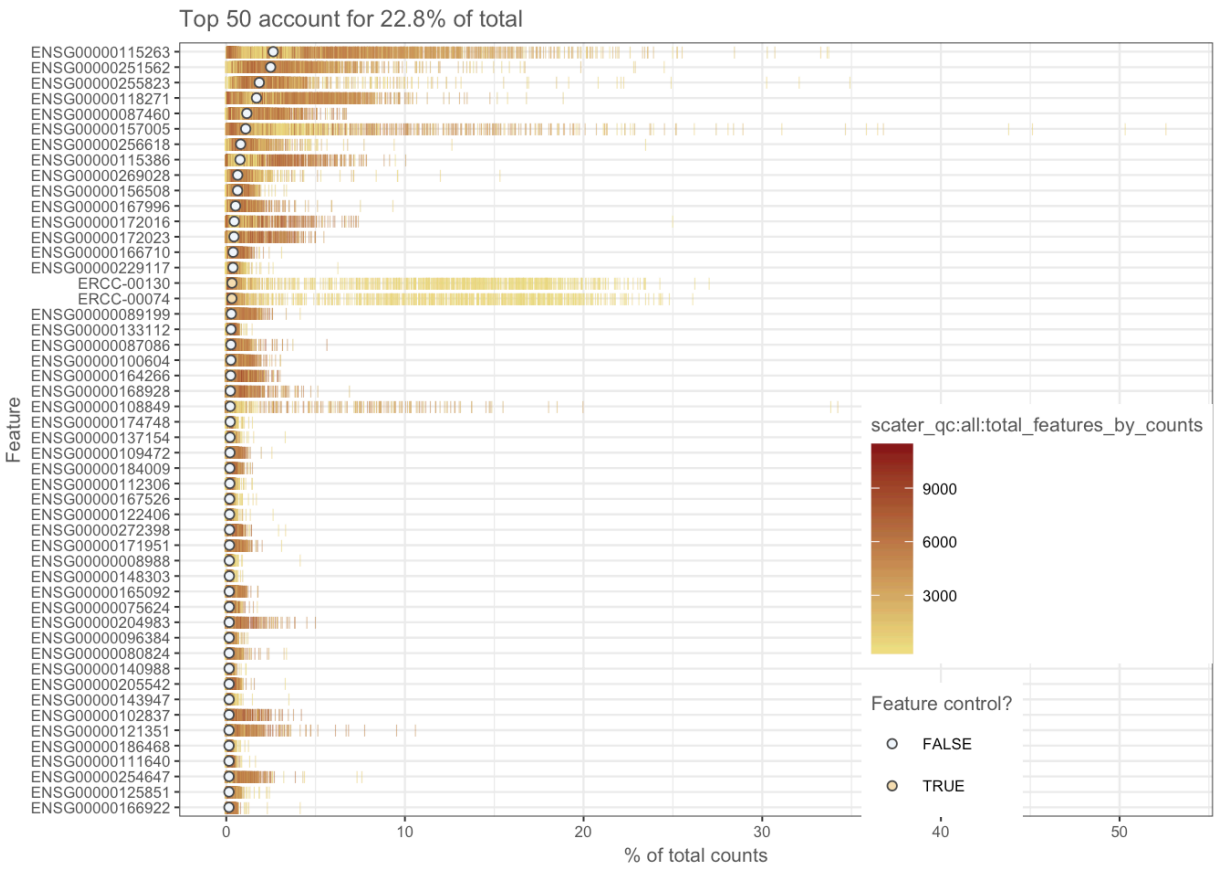

Figure 1: *Percentage of total counts assigned to the top 50 most highly-abundant features in the brain dataset. For each feature, each bar represents the percentage assigned to that feature for a single cell, while the circle represents the average across all cells. Bars are colored by the total number of expressed features in each cell, while circles are colored according to whether the feature is labelled as a control feature.*

Frequency of expression as a function of the mean Another useful plot is that of the frequency of expression (i.e., number of cells with non-zero expression) against the mean. These two metrics should be positively correlated with each other for most genes.

```
plotExprsFreqVsMean(sce.gse85241)
```

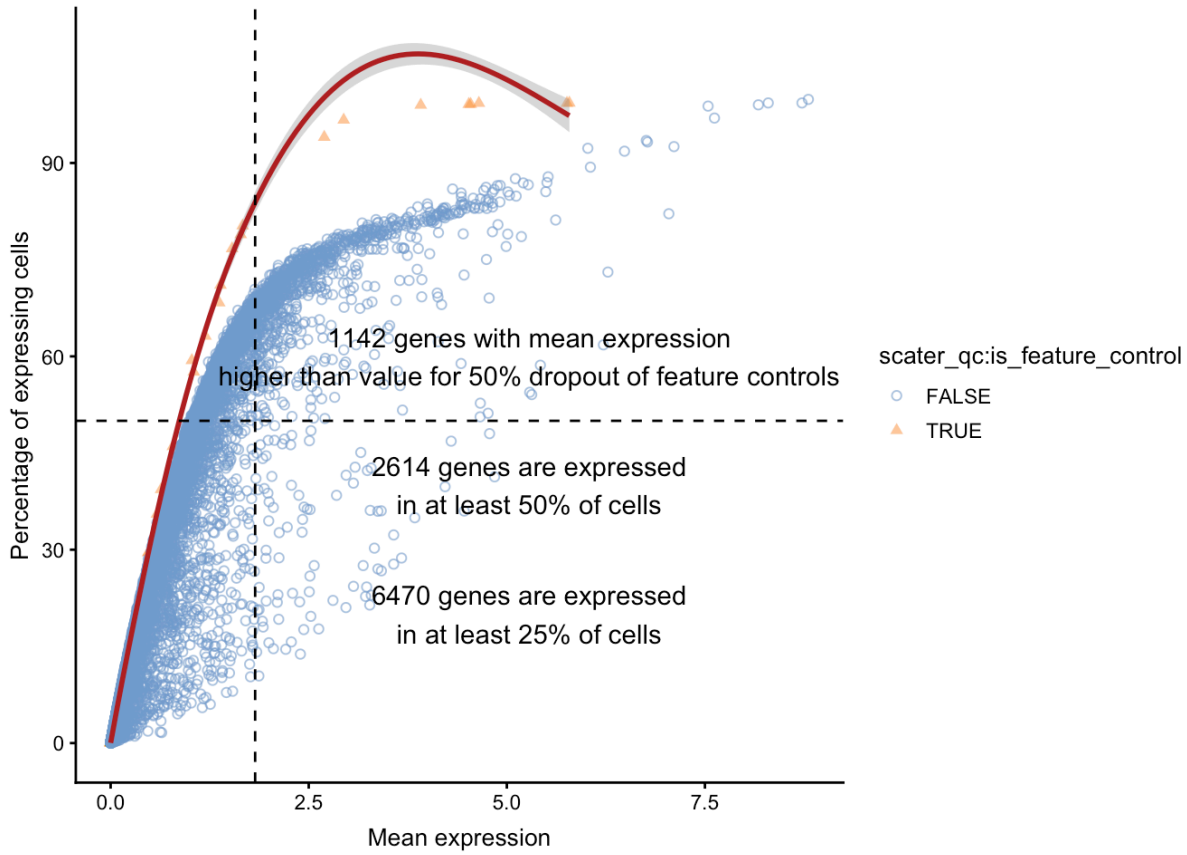

Figure 2: *frequency of expression (i.e., number of cells with non-zero expression) against the mean.*

Note: You can have multiple quality control plots 1) Plotting the percentage of counts assigned to feature controls 2) Cumulative expression plot 3) Plate position plot

2.1.5. Filtering the SingleCellExperiment

isOutlier defines the threshold at a certain number of median absolute deviations (MADs) away from the median expression. Values beyond this threshold are considered outliers and can be filtered out, assuming that they correspond to low-quality cells. Here, we define small outliers (using type="lower") for the log-total counts at 3 MADs from the median.

```
QC <- sce.gse85241$scater_qc
low.lib <- isOutlier(QC$all$log10_total_counts, type="lower", nmad=3)
low.genes <- isOutlier(QC$all$log10_total_features_by_counts, type="lower", n
mad=3)
high.spike <- isOutlier(QC$feature_control_ERCC$pct_counts, type="higher", nm
ad=3)
```

```
data.frame(LowLib=sum(low.lib), LowNgenes=sum(low.genes),
           HighSpike=sum(high.spike, na.rm=TRUE))
```

```
##   LowLib LowNgenes HighSpike
## 1    577       669       696
```

We are looking at three matrices here. These matrices define the quality of cells. Low-quality cells need to be removed to ensure that technical effects do not distort downstream analysis results. 1. LowLib: The library size is defined as the total sum of counts across all features, i.e., genes and spike-in transcripts. Cells with small library sizes are of low quality as the RNA has not been efficiently captured (i.e., converted into cDNA and amplified) during library preparation.

2. LowNgenes: Any cell with very few expressed genes is likely to be of poor quality as the diverse transcript population has not been successfully captured.

3. HighSpike: The proportion of reads mapped to spike-in transcripts is calculated relative to the library size for each cell. High proportions are indicative of poor-quality cells, where endogenous RNA has been lost during processing (e.g., due to cell lysis or RNA degradation). **Note:** In the absence of spike-in transcripts, the proportion of reads mapped to genes in the mitochondrial genome can also be used. *#####Question 3:* ***Reduce the MAD value to 1 and report the summary of Low number of genes and spikes?***

Low-quality cells are defined as those with extreme values for these QC metrics and are removed.

```
discard <- low.lib | low.genes | high.spike
sce.gse85241 <- sce.gse85241[,!discard]
summary(discard)
```

```
##    Mode   FALSE    TRUE
## logical    2346     726
```

We will compute size factors for the endogenous genes and spike-in transcripts, and use them to compute log-normalized expression values. **quickCluster** cluster similar cells based on rank correlations in their gene expression profiles. **Note:** This method is used for the size factor calculations.

```
library(scran)
clusters <- quickCluster(sce.gse85241, min.mean=0.1, method="igraph")
table(clusters)
```

```
## clusters
##   1   2   3   4
## 371 828 536 611
```

Question 4: *Define clusters with minimum size despite minimum mean.*

---

computeSumFactors is a method to normalize single-cell RNA-seq data by deconvolving size factors from cell pools. You can read more here

```
sce.gse85241 <- computeSumFactors(sce.gse85241, min.mean=0.1, clusters=cluste
rs)
summary(sizeFactors(sce.gse85241))
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##  0.07892  0.54127  0.82761  1.00000  1.22184 13.98424
```

Question 5: *Change mean here. How does it affect in computing sum factors.*

ComputeSpikeFactors Compute size factors based on the coverage of spike-in transcripts.

```
sce.gse85241 <- computeSpikeFactors(sce.gse85241, general.use=FALSE)
summary(sizeFactors(sce.gse85241, "ERCC"))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.09295 0.61309 0.88902 1.00000 1.27519 4.04643
```

```
sce.gse85241 <- normalize(sce.gse85241)
```

2.1.6. Identification of highly variable genes

We fit a trend to the spike-in variances as previously described, allowing us to model the technical noise for each gene (Figure 3). Again, we set "**block**" to ensure that uninteresting differences between plates or donors do not inflate the variance.
trendVar computes the biological and technical components of the gene-specific variance in single-cell RNA-seq data.
decomposeVar decomposes the gene-specific variance into biological and technical components for single-cell RNA-seq data.

```
block <- paste0(sce.gse85241$Plate, "_", sce.gse85241$Donor)
fit <- trendVar(sce.gse85241, block=block, parametric=TRUE)
dec <- decomposeVar(sce.gse85241, fit)
plot(dec$mean, dec$total, xlab="Mean log-expression",
     ylab="Variance of log-expression", pch=16)
is.spike <- isSpike(sce.gse85241)
points(dec$mean[is.spike], dec$total[is.spike], col="red", pch=16)
curve(fit$trend(x), col="dodgerblue", add=TRUE)
```

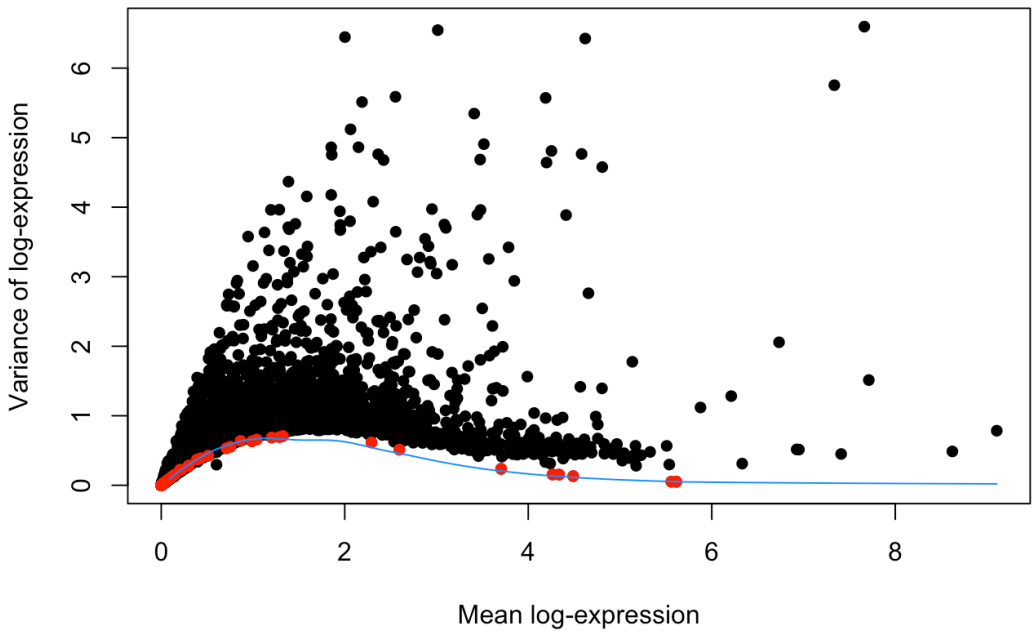

Figure 3: *Variance of normalized log-expression values for each gene in the GSE81076 dataset, plotted against the mean log-expression. The blue line represents the mean-dependent trend fitted to the variances of the spike-in transcripts (red).*

We will order genes by decreasing biological component, as described above.

```
dec.gse85241 <- dec
dec.gse85241$Symbol <- rowData(sce.gse85241)$Symbol
dec.gse85241 <- dec.gse85241[order(dec.gse85241$bio, decreasing=TRUE),]
head(dec.gse85241)
```

```
## DataFrame with 6 rows and 7 columns
##                             mean            total              bio
##                        <numeric>        <numeric>        <numeric>
## ENSG00000115263 7.66267378242402 6.59681396806777   6.568001618991
## ENSG00000089199 4.62064563864715 6.42527492432335 6.30366585830298
## ENSG00000169903 3.01379686443808  6.5465999616288 6.19244340917855
## ENSG00000254647 2.00214393757549 6.44633299225628 5.89008000995735
## ENSG00000118271  7.3354941164374 5.75428324335117 5.72283471884324
## ENSG00000171951  4.1895920246965 5.57157012090427 5.40710655270307
##                             tech   p.value       FDR      Symbol
##                        <numeric> <numeric> <numeric> <character>
## ENSG00000115263 0.0288123490767715         0         0         GCG
## ENSG00000089199  0.121609066020377         0         0        CHGB
## ENSG00000169903  0.354156552450253         0         0      TM4SF4
## ENSG00000254647  0.556252982298933         0         0         INS
## ENSG00000118271 0.0314485245079266         0         0         TTR
## ENSG00000171951  0.164463568201198         0         0        SCG2
```

Question 6: *Plot the expression of top 10 genes based on biological component?*

2.1.7. Clustering cells into putative subpopulations

The reduced dimension coordinates are used to cluster cells into putative subpopulations. We do so by constructing a shared-nearest-neighbour graph (Xu and Su 2015), in which cells are the nodes and edges are formed between cells that share nearest neighbours. Clusters are then defined as highly connected communities of cells within this graph, using methods from the igraph package. This is more efficient than forming a pairwise distance matrix for hierarchical clustering of large numbers of cells.

```
sce1 <- sce.gse85241

geneSymb <- data.frame(row.names=rownames(sce1),
    Symbol=mapIds(org.Hs.eg.db, keytype="ENSEMBL",
        keys=rownames(sce1), column="SYMBOL"))

rowData(sce1)$SYMBOL <- geneSymb
new.names <- rowData(sce1)$SYMBOL
missing.name <- is.na(new.names)
new.names[missing.name] <- rownames(sce1)[missing.name]
dup.name <- new.names %in% new.names$Symbol[duplicated(new.names$Symbol)]
new.names[dup.name] <- paste0(new.names, "_", rowData(sce1)$SYMBOL)[dup.name]
rownames(sce1) = make.names(new.names$Symbol, unique=TRUE)
```

```
snn.gr <- buildSNNGraph(sce1)
clusters <- igraph::cluster_walktrap(snn.gr)
table(clusters$membership)

##
##   1   2   3   4   5   6   7   8   9  10  11  12  13
## 329 252 237 197 525  25 109  45 146 125 249  86  21
```

We will now visualize the cluster assignments for all cells on the t-SNE plot in Figure 4. Adjacent cells are generally assigned to the same cluster, indicating that the clustering procedure was applied correctly.

```
sce1$Cluster <- factor(clusters$membership)
plotTSNE(sce1, colour_by="Cluster")
```

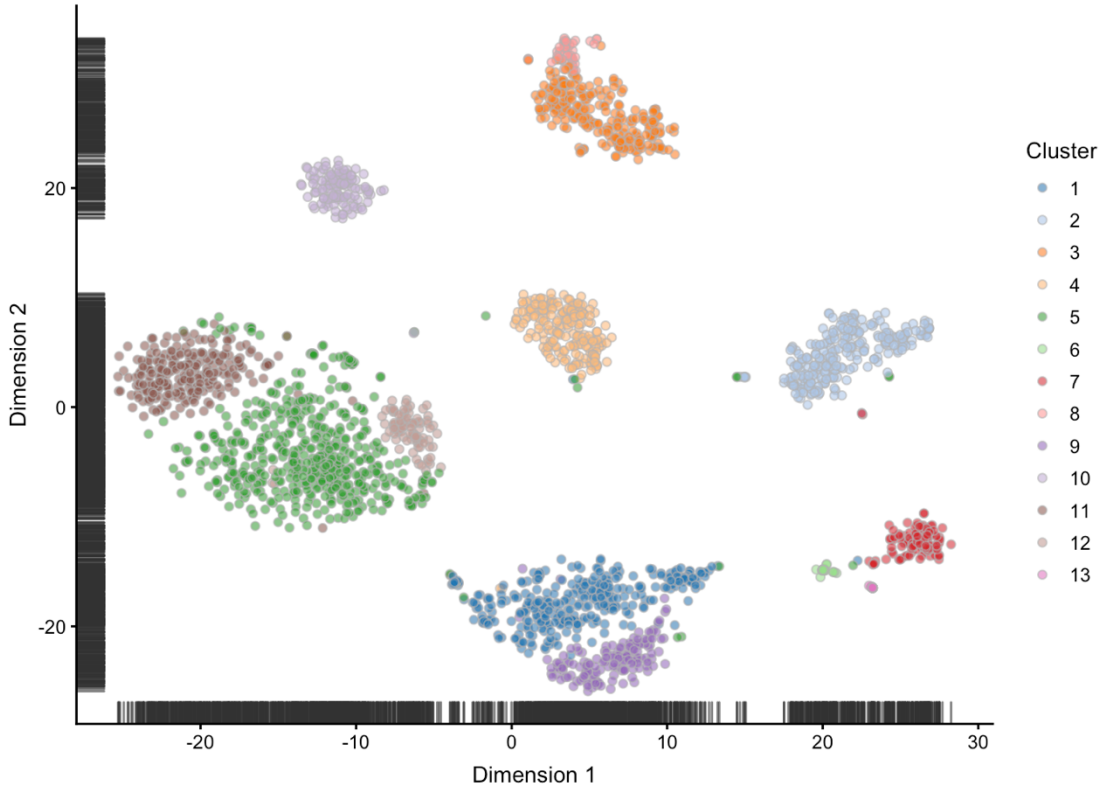

Figure 4: *t-SNE plot of the PCs of the pancraeas dataset. Each point represents a cell and is coloured according to its assigned cluster identity.*

2.1.8. Detecting marker genes between subpopulations

We use the findMarkers function with direction="up" to identify upregulated marker genes for each cluster. We will focus on upregulated genes as these can quickly provide positive identification of cell type in a heterogeneous population. We examine the table for cluster 1, in which log-fold changes are reported between cluster 1 and every other cluster. The same output is provided for each cluster in order to identify genes that discriminate between clusters.

findMarkers in {scran} package performs t-tests to identify differentially expressed genes (DEGs) between pairs of clusters. For each cluster, the log-fold changes and other statistics from all relevant pairwise comparisons are combined into a single table. A list of such tables is returned for all clusters to define a set of potential marker genes.

```
markers <- findMarkers(sce1, clusters$membership, direction="up")
demo <- markers[["5"]] # looking at cluster 5.
demo <- demo[demo$Top <= 3,]
```

Question 7 *Report the top 20 genes from Cluster 1 from above plot?*

```
my.clusters <- clusters$membership
marker.set <- markers[["1"]]
top.markers <- rownames(marker.set)[marker.set$Top <= 10]

plotHeatmap(sce1, features=top.markers, columns=order(my.clusters),
    colour_columns_by="Cluster", cluster_cols=FALSE,
    center=TRUE, symmetric=TRUE, zlim=c(-2, 2), cexRow = 3)
```

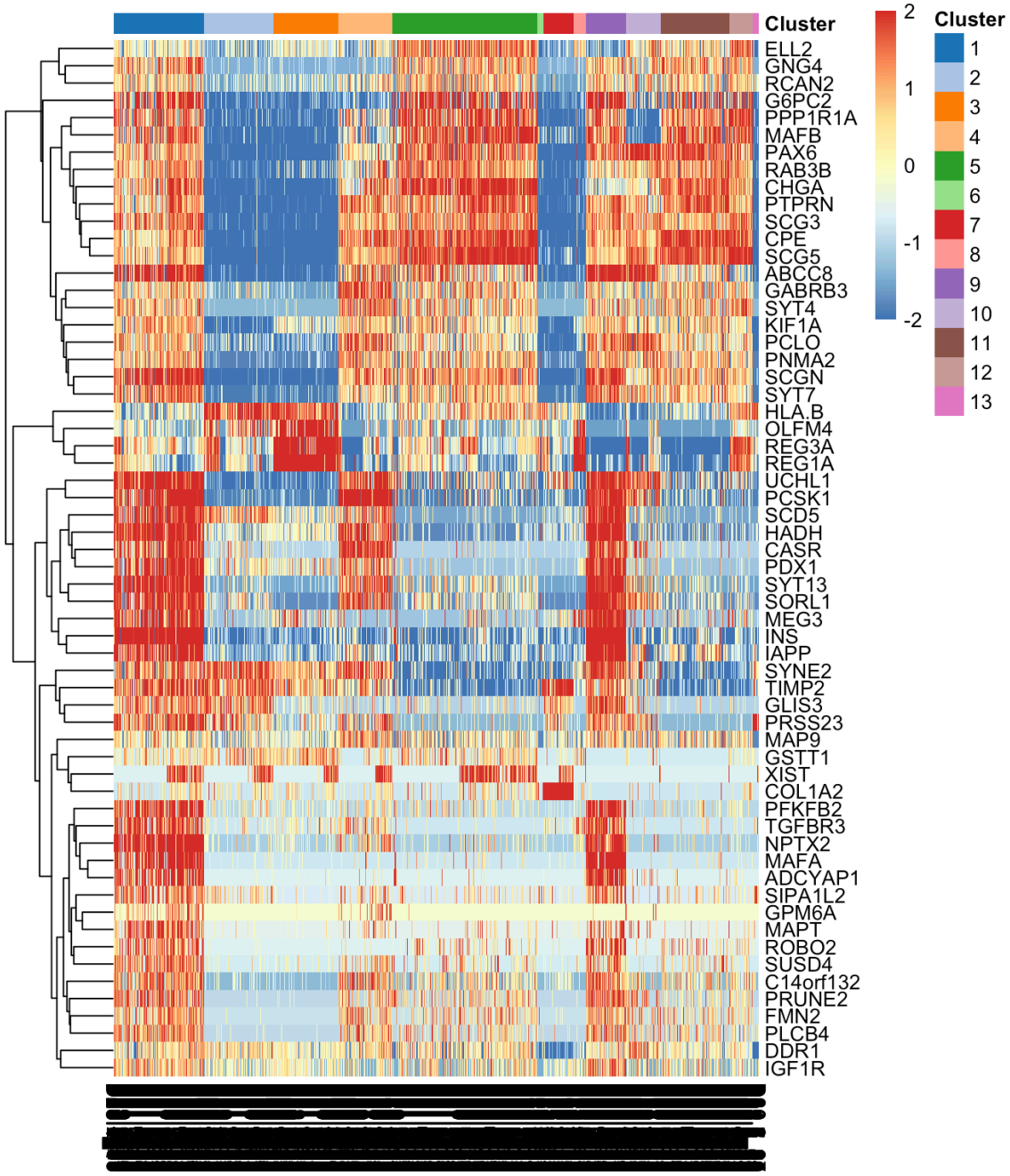

Figure 5: *Heatmap of mean-centered and normalized log-expression values for the top set of markers for cluster 1 in the brain dataset. Column colors represent the cluster to which each cell is assigned, as indicated by the legend.*

```
rm(gse85241.df)
gc()
```

```
##             used   (Mb) gc trigger   (Mb) limit (Mb)  max used   (Mb)
## Ncells   5652870  301.9   10209320  545.3         NA  10209320  545.3
## Vcells 134356774 1025.1  483256848 3687.0      16384 593539421 4528.4
```

2.2. Reading the data from Palasantza A et al study.

Now we will read data from ArrayExpress study (ID: E-MTAB-5061) published by Palasantza A et al. and can perfomm similar analysis as we did for dataset from Muraro et al.

2.2.1. Reading the files

```
unzip("E-MTAB-5061.processed.1.zip")
header <- read.table("pancreas_refseq_rpkms_counts_3514sc.txt",
                     nrow=1, sep="\t", comment.char="", stringsAsFactors=FALS
E)
ncells <- ncol(header) - 1L
```

```
col.types <- vector("list", ncells*2 + 2)
col.types[1:2] <- "character"
col.types[2+ncells + seq_len(ncells)] <- "integer"
e5061.df <- read.table("pancreas_refseq_rpkms_counts_3514sc.txt",
                       sep="\t", colClasses=col.types)
```

Further, we will retreive the gene names and counts in two different dataframes

```
gene.data <- e5061.df[,1:2]
e5061.df <- e5061.df[,-(1:2)]
colnames(e5061.df) <- as.character(header[1,-1])
dim(e5061.df)
```

```
## [1] 26271  3514
```

Spike-in transcripts are kept.

```
is.spike <- grepl("^ERCC-", gene.data[,2])
table(is.spike)
```

```
## is.spike
## FALSE  TRUE
## 26179    92
```

We will convert all row names to Ensembl identifiers.

```
gene.ids <- mapIds(org.Hs.eg.db, keys=gene.data[,1], keytype="SYMBOL", column
="ENSEMBL")
gene.ids[is.spike] <- gene.data[is.spike,2]
```

We will remove the duplicated entries and absent values and will keep the spike-ins.

```
keep <- !is.na(gene.ids) & !duplicated(gene.ids)
e5061.df <- e5061.df[keep,]
rownames(e5061.df) <- gene.ids[keep]
summary(keep)
```

```
##    Mode   FALSE    TRUE
## logical    3736   22535
```

2.2.2. Extraction of Metadata information

```
metadata <- read.table("E-MTAB-5061.sdrf.txt", header=TRUE,
                       sep="\t", check.names=FALSE, stringsAsFactors=FALSE)
m <- match(colnames(e5061.df), metadata[["Assay Name"]])
stopifnot(all(!is.na(m)))
metadata <- metadata[m,]
donor.id <- metadata[["Characteristics[individual]"]]
table(donor.id)
```

```
## donor.id
##           AZ    HP1502401 HP1504101T2D    HP1504901    HP1506401
##           96          352          383          383          383
##    HP1507101 HP1508501T2D    HP1509101 HP1525301T2D HP1526901T2D
##          383          383          383          384          384
```

Single cell Experiment object will be created with all the information.

```
sce.e5061 <- SingleCellExperiment(list(counts=as.matrix(e5061.df)),
                                  colData=DataFrame(Donor=donor.id),
                                 rowData=DataFrame(Symbol=gene.data[keep,1]))
isSpike(sce.e5061, "ERCC") <- grepl("^ERCC-", rownames(e5061.df))
sce.e5061
```

```
## class: SingleCellExperiment
## dim: 22535 3514
## metadata(0):
## assays(1): counts
## rownames(22535): ENSG00000118473 ENSG00000142920 ... ERCC-00061
##   ERCC-00048
## rowData names(1): Symbol
## colnames(3514): HP1502401_N13 HP1502401_D14 ... HP1526901T2D_O11
##   HP1526901T2D_A8
## colData names(1): Donor
## reducedDimNames(0):
## spikeNames(1): ERCC
```

2.2.3. Quality Control and Normalization

```
sce.e5061 <- calculateQCMetrics(sce.e5061, compact=TRUE)
QC <- sce.e5061$scater_qc
low.lib <- isOutlier(QC$all$log10_total_counts, type="lower", nmad=3)
low.genes <- isOutlier(QC$all$log10_total_features_by_counts, type="lower", n
mad=3)
high.spike <- isOutlier(QC$feature_control_ERCC$pct_counts, type="higher", nm
ad=3)
low.spike <- isOutlier(QC$feature_control_ERCC$log10_total_counts, type="lowe
r", nmad=2)
data.frame(LowLib=sum(low.lib), LowNgenes=sum(low.genes),
           HighSpike=sum(high.spike, na.rm=TRUE), LowSpike=sum(low.spike))
```

```
##   LowLib LowNgenes HighSpike LowSpike
## 1    162       572       904      359
```

```
discard <- low.lib | low.genes | high.spike | low.spike
sce.e5061 <- sce.e5061[,!discard]
summary(discard)
```

```
##    Mode   FALSE    TRUE
## logical    2285    1229
```

```
clusters <- quickCluster(sce.e5061, min.mean=1, method="igraph")
table(clusters)
```

```
## clusters
##   1   2   3   4   5   6
## 315 308 460 482 282 438
```

```
sce.e5061 <- computeSumFactors(sce.e5061, min.mean=1, clusters=clusters)
summary(sizeFactors(sce.e5061))
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
##  0.004916  0.377375  0.699866  1.000000  1.315723 12.740545
```

```
sce.e5061 <- computeSpikeFactors(sce.e5061, general.use=FALSE)
summary(sizeFactors(sce.e5061, "ERCC"))
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##  0.01433  0.19416  0.42867  1.00000  1.16641 11.23978
```

```
sce.e5061 <- normalize(sce.e5061)
```

Variance of normalized log-expression values for each gene in the Palasantza A et al. dataset, plotted against the mean log-expression. Each plot corresponds to a donor, where the blue line represents the mean-dependent trend fitted to the variances of the spike-in transcripts (red).

```
donors <- sort(unique(sce.e5061$Donor))
is.spike <- isSpike(sce.e5061)
par(mfrow=c(5, 2),
    mar=c(2.1, 2.1, 1.1, 0.1))
collected <- list()
for (x in unique(sce.e5061$Donor)) {
    current <- sce.e5061[,sce.e5061$Donor==x]
    if (ncol(current)<2L) { next }
    current <- normalize(current)
    fit <- trendVar(current, parametric=TRUE)
    dec <- decomposeVar(current, fit)
    plot(dec$mean, dec$total, xlab="Mean log-expression",
        ylab="Variance of log-expression", pch=16, main=x)
    points(fit$mean, fit$var, col="red", pch=16)
    curve(fit$trend(x), col="dodgerblue", add=TRUE)
    collected[[x]] <- dec
}
```

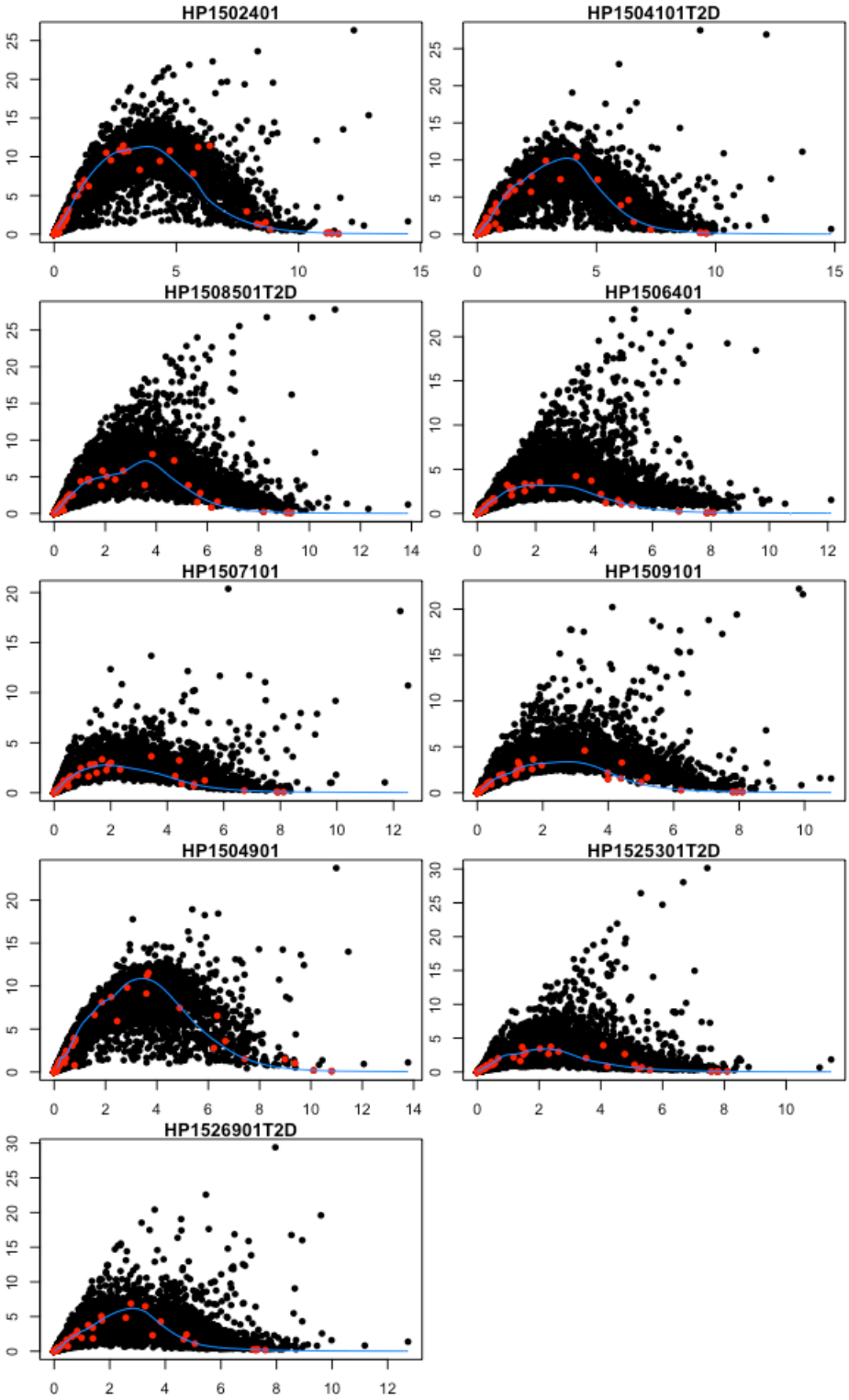

Figure 6: *Variance of normalized log-expression values for each gene in the Palasantza A et al. dataset, plotted against the mean log-expressionEach plot corresponds to a donor, where the blue line represents the mean-dependent trend fitted to the variances of the spike-in transcripts (red).*

2.2.4. Identification of highly variable genes

We combine statistics across donors to consolidate them into a single set of results for this study. We then order genes by decreasing biological component.

```
dec.e5061 <- do.call(combineVar, collected)
dec.e5061$Symbol <- rowData(sce.e5061)$Symbol
dec.e5061 <- dec.e5061[order(dec.e5061$bio, decreasing=TRUE),]
head(dec.e5061)
```

```
## DataFrame with 6 rows and 7 columns
##                              mean            total              bio
##                         <numeric>        <numeric>        <numeric>
## ENSG00000115263 9.79785008603535 24.9554533902531 24.7436081286739
## ENSG00000118271 10.3624735164025 19.1597271291327 19.0677809544342
## ENSG00000089199 8.78894397314284 17.3693554890887 17.1073240459752
## ENSG00000169903 7.11678634372552 15.2550468907374 14.6597366417481
## ENSG00000166922 7.90580761095785 15.0817216478007 14.6353279279495
## ENSG00000254647 4.94929481818886 17.8290056901903 14.5719504604648
##                              tech   p.value       FDR      Symbol
##                         <numeric> <numeric> <numeric> <character>
## ENSG00000115263  0.211845261579168         0         0         GCG
## ENSG00000118271 0.0919461746985641         0         0         TTR
## ENSG00000089199  0.262031443113461         0         0        CHGB
## ENSG00000169903  0.595310248989271         0         0      TM4SF4
## ENSG00000166922  0.446393719851191         0         0        SCG5
## ENSG00000254647   3.25705522972549         0         0         INS
```

Question 8: *Use the single cell experiment object from the ArrayExpress studey the dataset from Palasantza et al and identify the clusters and differentially expressed genes.*

Hint Follow the section 2.1.7 and 2.1.8 for this exercise. ***

```
rm(e5061.df)
gc()
```

```
##             used   (Mb) gc trigger   (Mb) limit (Mb)  max used   (Mb)
## Ncells   5724274  305.8   10209320  545.3         NA  10209320  545.3
## Vcells 191167359 1458.5  483256848 3687.0      16384 593539421 4528.4
```

3. Feature selection across batches

To obtain a single set of features for batch selection, we take the top 1000 genes with the largest biological components from each batch. The intersection of this set across batches is defined as our feature set for MNN correction.

```
top.e5061 <- rownames(sce.e5061)[seq_len(1000)]
top.gse85241 <- rownames(dec.gse85241)[seq_len(1000)]
chosen <- Reduce(intersect, list(top.e5061, top.gse85241))
```

We will now add the gene symbols to the chosen identifiers from the genes intersect from two studies we analayzed.

```
symb <- mapIds(org.Hs.eg.db, keys=chosen, keytype="ENSEMBL", column="SYMBOL")
DataFrame(ID=chosen, Symbol=symb)
```

```
## DataFrame with 51 rows and 2 columns
##                   ID      Symbol
##          <character> <character>
## 1   ENSG00000169504       CLIC4
## 2   ENSG00000069702      TGFBR3
## 3   ENSG00000117632       STMN1
## 4   ENSG00000142949       PTPRF
## 5   ENSG00000169213       RAB3B
## ...             ...         ...
## 47  ENSG00000117318         ID3
## 48  ENSG00000198830       HMGN2
## 49  ENSG00000253368       TRNP1
## 50  ENSG00000175130    MARCKSL1
## 51  ENSG00000162522    KIAA1522
```

Question 9: *Find how many genes each dataset has and how many overlap within these two datasets.*

The use of an intersection is a rather conservative strategy as it requires the same gene to be highly variable in all batches. This may not be possible for marker genes of cell types that are not present in all batches.

```
in.all <- Reduce(intersect, list(rownames(dec.e5061),
                                 rownames(dec.gse85241)))
```

An alternative approach is to use combineVar() to compute the average biological component across batches for each gene. The feature set can then be defined as the top X genes with the largest biological components. Setting weighted=FALSE so each batch contributes equally.

```
combined <- combineVar(dec.e5061[in.all,], dec.gse85241[in.all,],
                       weighted=FALSE)
chosen <- rownames(combined)[head(order(combined$bio, decreasing=TRUE), 1000)
]
```

4. Performing MNN-based correction (Mutual nearest neighbors correction)

mnnCorrect corrects for batch effects in single-cell expression data using the mutual nearest neighbors method. There are several other methods that can be used for batch corrections e.g. canonical correlation analysis (CCA), removeBatchEffect, multiBatchNorm, and fastMNN. We apply the mnnCorrect() function to the three batches to remove the batch effect, using the genes in chosen. This involves correcting their expression values so that all cells are comparable in the coordinate system of the first batch. The function returns a set of matrices containing corrected expression values, which we can use in downstream analyses.

```
original <- list(logcounts(sce.e5061)[chosen,],
                 logcounts(sce.gse85241)[chosen,])
corrected <- do.call(mnnCorrect, c(original, list(k=20, sigma=0.1)))
str(corrected$corrected)
```

```
## List of 2
##  $ : num [1:1000, 1:2285] 0.0828 0.0893 0.0792 0 0.1202 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:1000] "ENSG00000115263" "ENSG00000118271" "ENSG000000891
99" "ENSG00000169903" ...
##   .. ..$ : chr [1:2285] "HP1502401_J13" "HP1502401_H13" "HP1502401_J14" "H
P1502401_B14" ...
##  $ : num [1:1000, 1:2346] 0.093662 0.089142 0.085062 0.069071 -0.000916 ..
.
```

The function also reports the MNN pairs that were identified in each successive batch. This may be useful for trouble-shooting, e.g., to check whether cells independently assigned to the same cell type are correctly identified as MNN pairs.

```
corrected$pairs
```

```
## [[1]]
## DataFrame with 0 rows and 3 columns
##
## [[2]]
## DataFrame with 4369 rows and 3 columns
##      current.cell other.cell other.batch
##         <integer>      <Rle>       <Rle>
## 1              69          5           1
## 2            2250         20           1
## 3            1344         20           1
## 4            1411         20           1
## 5            2339         94           1
## ...           ...        ...         ...
## 4365         1849       2283           1
## 4366         2312       2283           1
```

```
## 4367         2135       2283           1
## 4368          133       2283           1
## 4369         2340       2283           1
```

5. Examining the effect of correction

We create a new SingleCellExperiment object containing these corrected counts for each cell, along with information regarding the batch of origin.

```
omat <- do.call(cbind, original)
mat <- do.call(cbind, corrected$corrected)
colnames(mat) <- NULL
sce <- SingleCellExperiment(list(original=omat, corrected=mat))
colData(sce)$Batch <- rep(c("e5061", "gse85241"),
                          lapply(corrected$corrected, ncol))
sce
```

```
## class: SingleCellExperiment
## dim: 1000 4631
## metadata(0):
## assays(2): original corrected
## rownames(1000): ENSG00000115263 ENSG00000118271 ...
##   ENSG00000173230 ENSG00000126247
## rowData names(0):
## colnames(4631): HP1502401_J13 HP1502401_H13 ... D30.8_93 D30.8_94
## colData names(1): Batch
## reducedDimNames(0):
## spikeNames(0):
```

We examine the batch correction with some t-SNE plots. Figure 7 demonstrates how the cells separate by batch of origin in the uncorrected data. After correction, more intermingling between batches is observed, consistent with the removal of batch effects.

```
osce <- runTSNE(sce, exprs_values="original", rand_seed=100)
ot <- plotTSNE(osce, colour_by="Batch") + ggtitle("Original")
csce <- runTSNE(sce, exprs_values="corrected", rand_seed=100)
ct <- plotTSNE(csce, colour_by="Batch") + ggtitle("Corrected")
multiplot(ot, ct, cols=2)
```

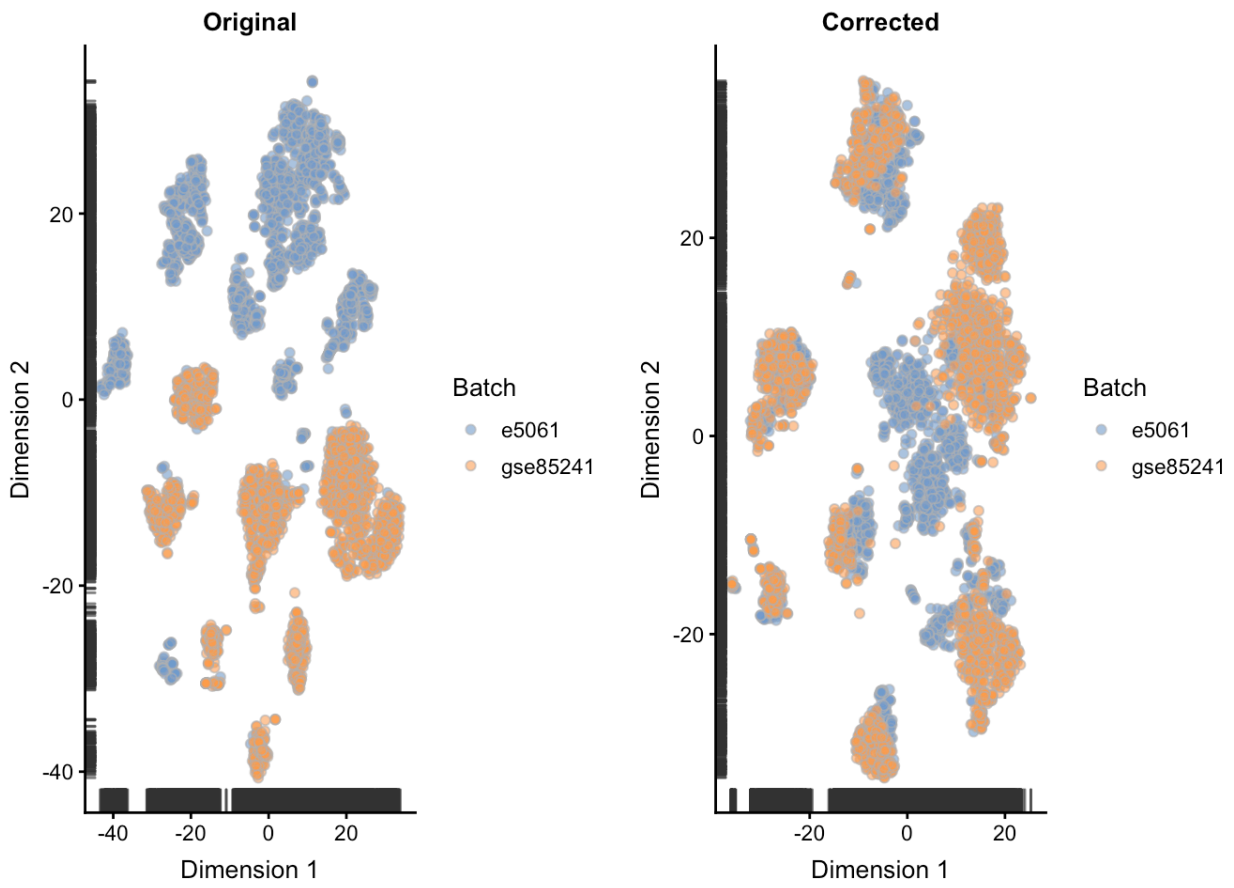

Figure 7: *t-SNE plots of the pancreas datasets, before and after MNN correction. Each point represents a cell and is coloured by the batch of origin.*

We color by the expression of marker genes for known pancreas cell types to determine whether the correction is biologically sensible. Cells in the same visual cluster express the same marker genes (Figure 8), indicating that the correction maintains separation of cell types.

```
ct.gcg <- plotTSNE(csce, by_exprs_values="corrected",
                   colour_by="ENSG00000115263") + ggtitle("Alpha cells (GCG)"
)
ct.ins <- plotTSNE(csce, by_exprs_values="corrected",
                   colour_by="ENSG00000254647") + ggtitle("Beta cells (INS)")
ct.sst <- plotTSNE(csce, by_exprs_values="corrected",
                   colour_by="ENSG00000157005") + ggtitle("Delta cells (SST)"
)
ct.ppy <- plotTSNE(csce, by_exprs_values="corrected",
                   colour_by="ENSG00000108849") + ggtitle("PP cells (PPY)")
multiplot(ct.gcg, ct.ins, ct.sst, ct.ppy, cols=2)
```

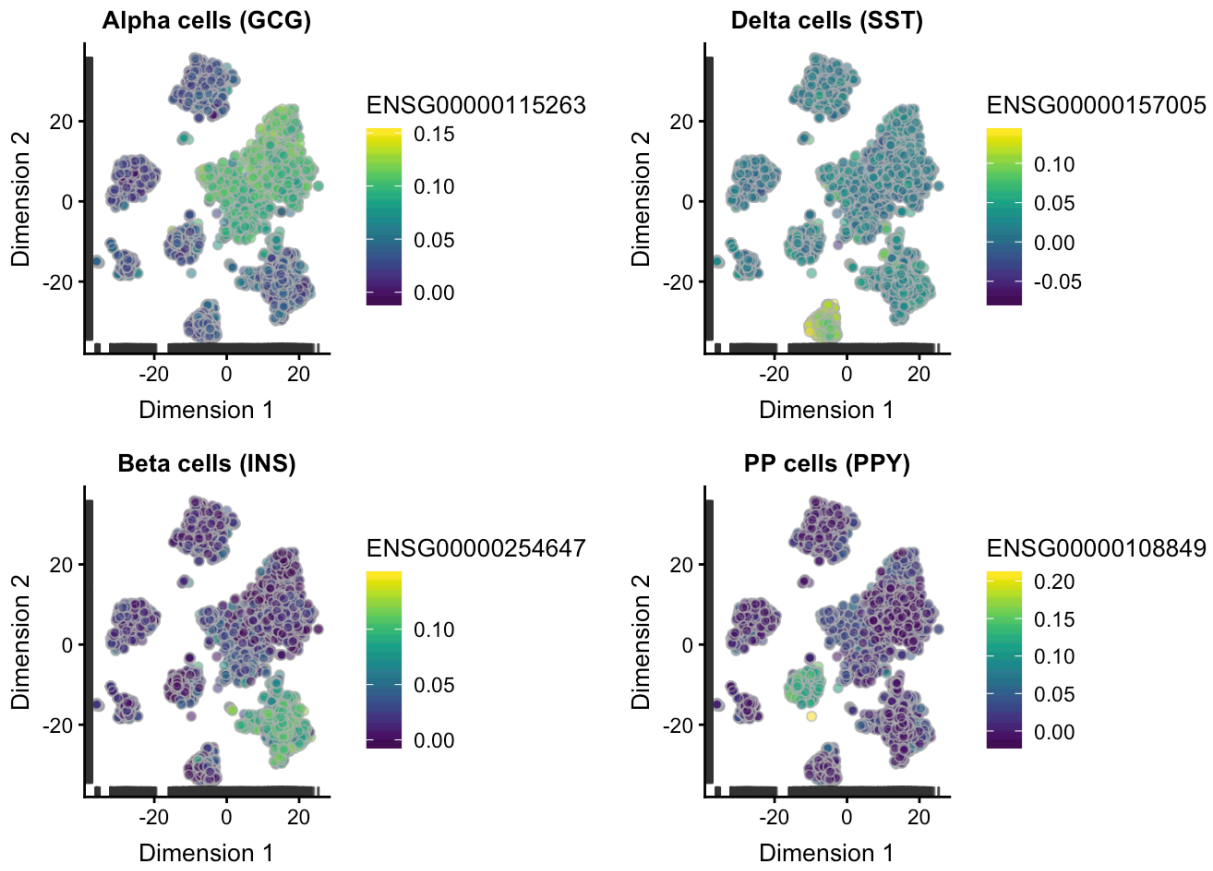

Figure 8: *t-SNE plots after MNN correction, where each point represents a cell and is coloured by its corrected expression of key marker genes for known cell types in the pancreas*

6. Using the corrected values in downstream analyses

MNN correction places all cells from all batches within the same coordinate system. This means that the corrected values can be freely used to define distances between cells for dimensionality reduction or clustering. However, the correction does not preserve the mean-variance relationship. As such, we do not recommend using the corrected values for studying heterogeneity. MNN-corrected values are generally not suitable for differential expression (DE) analyses, for several reasons: 1. The default parameters of mnnCorrect() do not return corrected values on the log-scale, but rather a cosine-normalized log-scale. This makes it difficult to interpret the effect size of DE analyses based on the corrected values.

2. It is usually inappropriate to perform DE analyses on batch-corrected values, due to the failure to model the uncertainty of the correction. This usually results in loss of type I error control, i.e., more false positives than expected.

7. Building Trajectories of dataset from by Muraro et al. (2016) using Diffusion Map

---

There are several methods by which one can build trajectories (pseudotime or real time) namely monocle, SCATER, SCUBA, kbranches, densityPath, destiny and scdiff. We will be using destiny to build the diffusion map for one of our dataset. **Blocking on the cell cycle phase** Cell cycle phase is usually uninteresting in studies focusing on other aspects of biology. However, the effects of cell cycle on the expression profile can mask other effects and interfere with the interpretation of the results. This cannot be avoided by simply removing cell cycle marker genes, as the cell cycle can affect a substantial number of other transcripts (Buettner et al. 2015). We will empirically identify the cell cycle phase using the pair-based classifier in cyclone. assignment of cell-cycle phase thorugh cyclone takes 20-25 minutes depending on the CPU and memory limits.

```
library(readxl)
sce1 <- sce.gse85241
ensembl <- rownames(sce1)
set.seed(100)
hs.pairs <- readRDS(system.file("exdata", "human_cycle_markers.rds",
    package="scran"))
assignments <- cyclone(sce1, hs.pairs, gene.names=ensembl, assay.type="logcou
nts")
plot(assignments$score$G1, assignments$score$G2M,
    xlab="G1 score", ylab="G2/M score", pch=16)
```

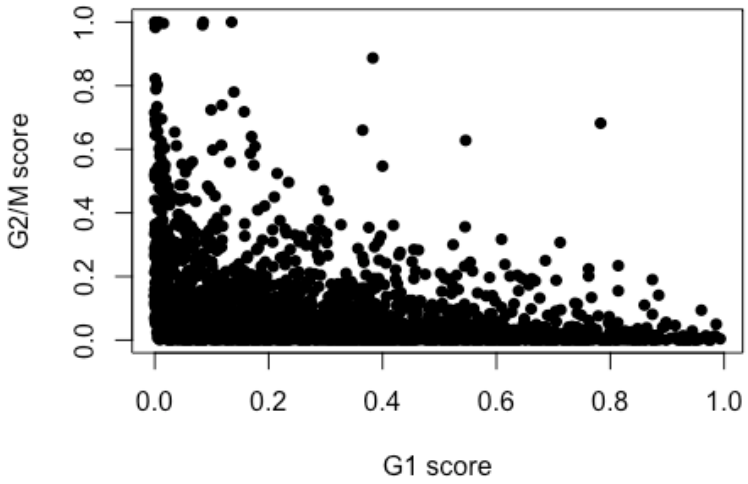

Figure 9: *Cell cycle phase scores from applying the pair-based classifier on the TH2 dataset, where each point represents a cell.*

We can block directly on the phase scores in downstream analyses. This is more graduated than using a strict assignment of each cell to a specific phase, as the magnitude of the score considers the uncertainty of the assignment. The phase covariates in the design matrix will absorb any phase-related effects on expression such that they will not affect estimation of the effects of other experimental factors. Users should also ensure that the phase score is not confounded with other factors of interest. For example, model fitting is not possible if all cells in one experimental condition are in one phase, and all cells in another condition are in a different phase.

```
design <- model.matrix(~ G1 + G2M, assignments$score)
fit.block <- trendVar(sce1, design=design, parametric=TRUE, use.spikes=NA)
dec.block <- decomposeVar(sce1, fit.block)

library(limma)
sce1.block <- sce1
assay(sce1.block, "corrected") <- removeBatchEffect(
    logcounts(sce1), covariates=design[,-1])

sce1.block <- denoisePCA(sce1.block, technical=dec.block,
    assay.type="corrected")
dim(reducedDim(sce1.block, "PCA"))

## [1] 2346    5
```

The result of blocking on design is visualized with some PCA plots in Figure 10. Before removal, the distribution of cells along the first two principal components is strongly associated with their G1 and G2/M scores. This is no longer the case after removal, which suggests that the cell cycle effect has been mitigated.

```
sce1$G1score <- sce1.block$G1score <- assignments$score$G1
sce1$G2Mscore <- sce1.block$G2Mscore <- assignments$score$G2M


# Without blocking on phase score.
fit <- trendVar(sce1, parametric=TRUE, use.spikes=NA)
sce1 <- denoisePCA(sce1, technical=fit$trend)
fontsize <- theme(axis.text=element_text(size=12), axis.title=element_text(si
ze=16))
out <- plotReducedDim(sce1, use_dimred="PCA", ncomponents=2, colour_by="G1sco
re",
    size_by="G2Mscore") + fontsize + ggtitle("Before removal")

# After blocking on the phase score.
out2 <- plotReducedDim(sce1.block, use_dimred="PCA", ncomponents=2,
    colour_by="G1score", size_by="G2Mscore") + fontsize +
    ggtitle("After removal")
multiplot(out, out2, cols=2)
```

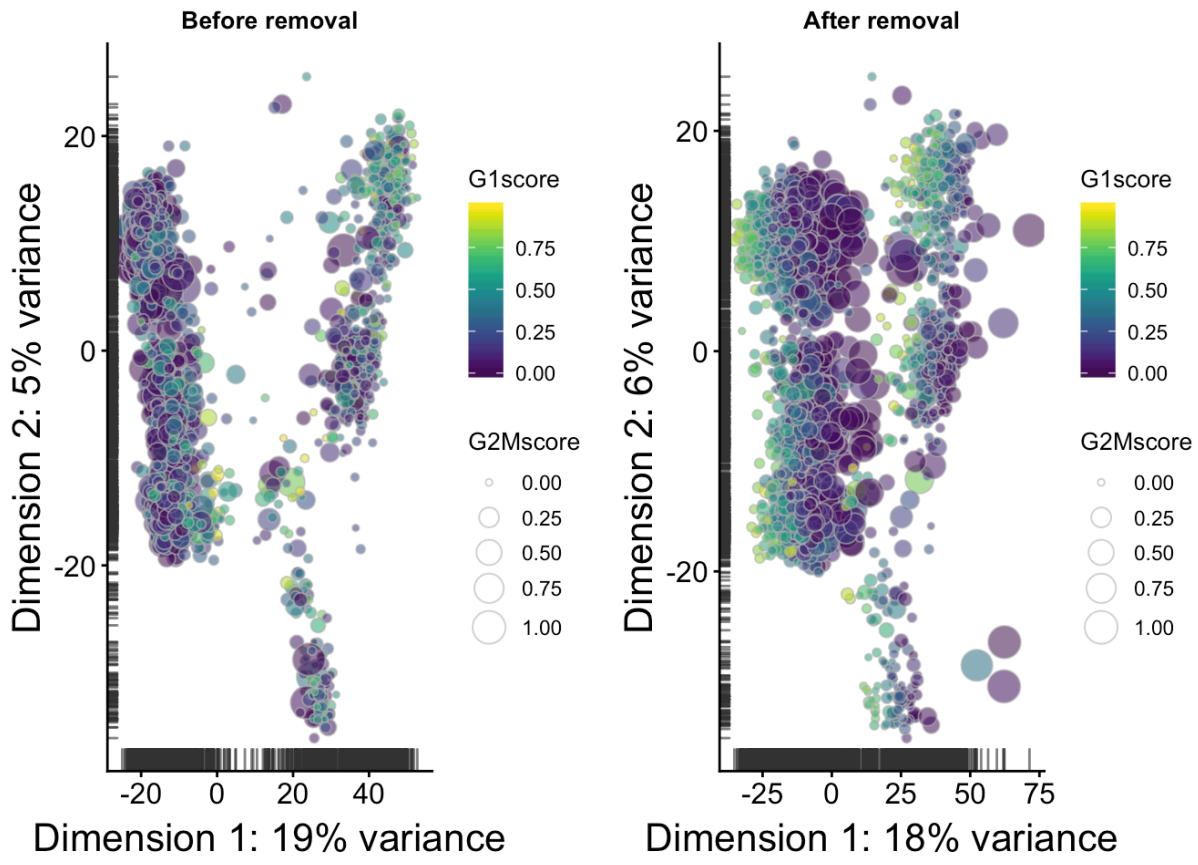

Figure 10: *PCA plots before (left) and after (right) removal of the cell cycle effect in the Pancreas dataset. Each cell is represented by a point with colour and size determined by the G1 and G2/M scores, respectively.*

Cells are arranged along a trajectory in the low-dimensional space. The change in expression of "GCG" gene from left to right.

```
plotDiffusionMap(sce1.block, colour_by="ENSG00000115263",
    run_args=list(use_dimred="PCA", sigma=25)) + fontsize
```

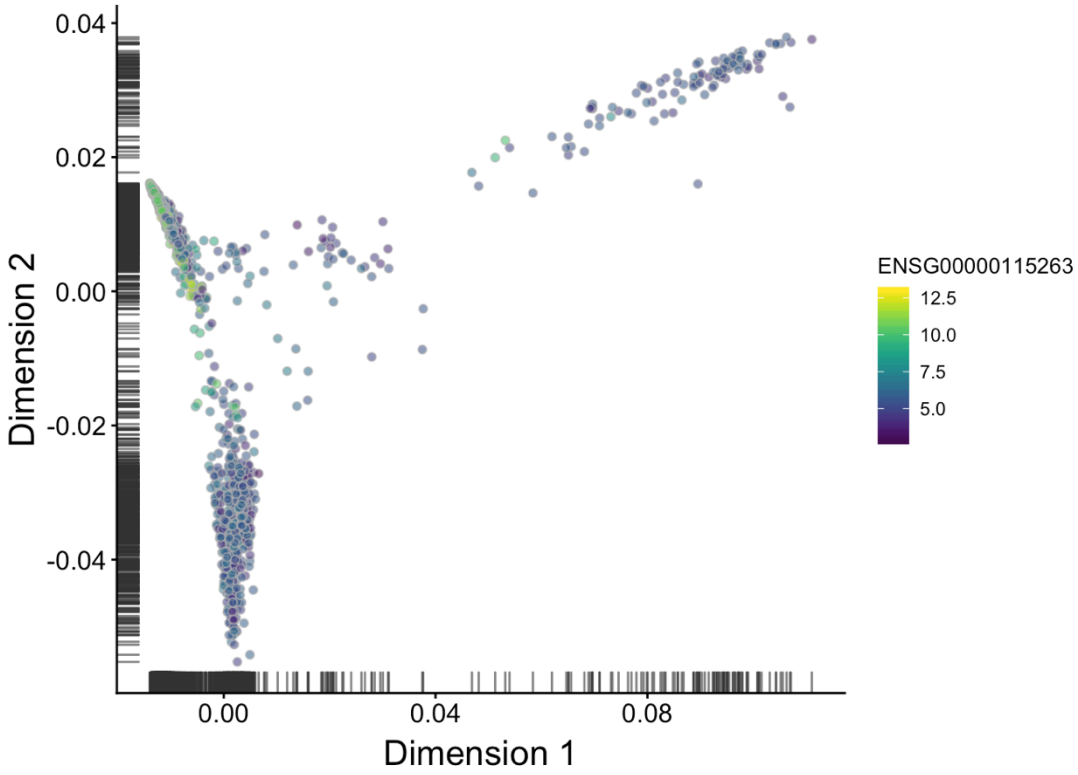

Figure 11: *diffusion map for the TH2 dataset, where each cell is coloured by its expression of GCG A larger sigma is used compared to the default value to obtain a smoother plot.*

Question 10: *We plotted the colour for gene GCG in the above diffusion map. Now, generate the same Colour diffusion map with genes "INS: ENSG00000254647", "SST: ENSG00000157005" and "PPY: ENSG00000108849".*

---

8. Resources

---

1. Analyzing single-cell RNA-seq data containing UMI counts
2. Correcting batch effects in single-cell RNA-seq data
3. Further strategies for analyzing single-cell RNA-seq data
4. Analysis of single cell RNA-seq data