Integration: A single-cell comparison of adult and foetal human epicardium defines the age-associated changes in epicardial activity

Vincent Knight-Schrijver

03 February 2023

Introduction

We are now in a position to integrate the adult and foetal datsets. In this script we will initially annotate our data, followed by combining the raw counts from both stages, and aligning the features. Then we will process all samples, split foetal and adult stages out, cluster and sub-sample foetal datasets (for stratified cell type sampling), and combined both datasets again using the raw counts across the sampled datasets using our custom integration tree. We will continue the analysis here by loading in the foetal dataset generated in the previous report. This step is focussed on aligning foetal data characteristics with those found in the adult data ready for integrating our human heart stages by going through some quality control steps.

As usual, let's pick up our functions and packages.

```
source("config.r")
library(rmarkdown)
    GetPackages=F
func.dir="~/Documents/Rlib/functions"
source("scripts/packages.r")
```

Loading the objects prior to integration

We shall take the raw counts from the adult sampled dataset, and the foetal dataset after QC. We will load in the Seurat objects that we made earlier in this analysis.

```
######################################################################
# old foetal integrated dataset (Seurat Object)
  foetal.integrated = readRDS(file.path("output/v_3/foetal_and_results", "foetal.integrated.rds"))

# Adult data sampled from HCA (Seurat Object)
  adult.ss.so = readRDS("RDS_files_v3/adult.ss.so.rds")
######################################################################
```

Feature standardisation

Between our datasets, we have potentially several cellranger genome annotation versions. In this analysis we are using the HGNC gene symbol and not EnsemblID. Therefore, we should update the features of old versions in an attempt to standardise our datasets. This involves reading in the old cellranger output features.tsv files, and mapping them across using the EnsemblID. There is an issue of duplicated feature names that we will address by aggregating these duplicate features together. These are HGNC gene symbols that are the same in two or more different EnsemblIDs.

```
######################################################################
### Gene / feature mapping and updating
# We need to normalise features - we will pull up our previous features list in the foetal datasets
# Cellranger (pre 3)
  genes1 = read.table("data/gene_features/Alex10X_features.tsv")
# Cellranger (6)
```

```
  genes2 = read.table("data/gene_features/Semih_cellranger6_features.tsv.gz")
# Cellranger (3.02)
  genes4 = read.table("data/gene_features/Vince_cellranger3_features.tsv.gz")

# where genes4[,2] == rownames of the adult integrated dataset
# Thus to bring our adult features up to the cellranger 6 version we need to
# First map to EnsemblID:
  ens.map = match(genes4[,1], genes2[,1])
  ens.na = is.na(match(genes4[,1], genes2[,1]))

# Second, new column names as mapped to ENSID in the genes2 object (cellranger 6)
  genes4[,5] = genes2[ens.map,2]
  genes4[ens.na,5] = genes4[ens.na,2]

# Where genes4[,5] is now the correct and updated row by row mapping for previous
# cellranger gene names

# Simply match across for each object now:
# integrated dataset (2000 features)
  genes4[match(rownames(adult.ss.so),genes4[,2]),5]

  # Annoying issues of duplicated genes...
      genes5 = genes4[,5]
      dup.genes = genes5[duplicated(genes5)]
      duplicates2remove = genes5 %in% dup.genes & (1:length(genes5) %in% which(duplicated(genes5)))

  # remove genes from genes list:
     genes5 = genes5[!duplicates2remove]

  # This should now the the updated rownames after aggregating duplicates
    rownames(adult.matrix) = genes5
    gc()

######################################################################
```

Constructing an initial merged matrix.

After this step we now have the standardised gene list between samples. We can construct the adult and foetal matrices prior to merging.

```
######################################################################
# ### Adult matrix construction:
   adult.matrix = as.matrix(GetAssayData(adult.ss.so, assay = "RNA", slot = "counts"))

  # aggregate adult gene duplicates:
    for(i in 1:length(dup.genes)){
      # Sum together genes of same name into one row to keep
        adult.matrix[which(genes5 == dup.genes[i])[1],] = colSums(adult.matrix[genes5 == dup.genes[i],])
    }
  # Then we subset the matrix removing the redundant duplicates
     adult.matrix = adult.matrix[!duplicates2remove,]

# Foetal matrix
  foetal.matrix = as.matrix(GetAssayData(foetal.integrated, assay = "RNA", slot = "counts"))
  gc()

# Quick threshold of features in data:
# Removing dud genes (minimise the object size)
  adult.matrix = adult.matrix[rowSums(adult.matrix) > 4,]
  foetal.matrix = foetal.matrix[rowSums(foetal.matrix) > 4,]
  gc()

######################################################################
```

Then we can create a new matrix for the merged data. We do this by constructing an empty matrix of the dimensions of [genes,cells] and then fill the matrix iteratively taking the standardised rownames and matching them against rownames in foetal and adult matrices. Although a little lengthy, this ensures that we are filling the matrix correctly, putting the data from the same feature from foetal and adult matrices together.

```
######################################################################
# Create new empty matrix for all feature names and columns
  all.matrix = matrix(0, length(unique(c(rownames(foetal.matrix), rownames(adult.matrix)))), ncol(adult.matrix)+ncol(foetal.matrix))
  rownames(all.matrix) = unique(c(rownames(foetal.matrix), rownames(adult.matrix)))
  colnames(all.matrix) = c(colnames(foetal.matrix),colnames(adult.matrix))
  gc()
```

```
  # It must be noted that the next function takes a fair bit of memory...
  # if lacking, remove the adult matrix and free up ram. We can build again after.
  # # # # optional # # # # #
            # clear ram... (we will rebuild adult matrix)
              rm(foetal.integrated)
              rm(adult.matrix)
              gc()
  # # # # # # # # # # # # # #

# Then fill in the spaces for foetal
  all.matrix[rownames(foetal.matrix),colnames(foetal.matrix)] = foetal.matrix[rownames(foetal.matrix),colnames(foetal.matrix)]

      ## rebuild adult matrix - remove duplicated genes...
        adult.matrix = as.matrix(GetAssayData(adult.ss.so, assay = "RNA", slot = "counts"))

        # aggregate adult gene duplicates:
          for(i in 1:length(dup.genes)){
            # Sum together genes of same name into one row to keep
              adult.matrix[which(genes5 == dup.genes[i])[1],] = colSums(adult.matrix[genes5 == dup.genes[i],])
          }
        # Then we subset the matrix removing the redundant duplicates
           adult.matrix = adult.matrix[!duplicates2remove,]

        # This should now the the updated rownames after aggregating duplicates
          rownames(adult.matrix) = genes5
          gc()

        # again remove the genes
          adult.matrix = adult.matrix[rowSums(adult.matrix) > 4,]

# Then fill in the spaces for adult data
  all.matrix[rownames(adult.matrix),colnames(adult.matrix)] = adult.matrix[rownames(adult.matrix),colnames(adult.matrix)]


# ### Finally, we want to generate a sparse matrix and save as 10x type data in order to store with low space usage
  all.matrix.sparse <- Matrix(all.matrix, sparse = T)

  write10xCounts(
    "all_matrix/",
    all.matrix.sparse,
    barcodes = colnames(all.matrix),
    gene.id = rownames(all.matrix),
    gene.symbol = rownames(all.matrix),
    gene.type = "Gene Expression",
    overwrite = FALSE,
    type = c("auto", "sparse", "HDF5"),
    genome = "unknown",
    version = c("3"),
    chemistry = "Single Cell 3' v3",
    original.gem.groups = 1L,
    library.ids = "custom"
  )
#########################################################################
```

Creating a Seurat Object of merged data

We have merged our datasets together across a set of standardised features and saved it as a sparse matrix. Now we will load it and organise it into Seurat's handy object. To begin with we will add annotations.

```
#########################################################################
### Seurat Object and Annotations
# Now we can create a Seurat Object
  all.so = CreateSeuratObject(all.matrix.sparse, Assay="RNA")

# Adult annotations
  # We will select only a few of the annotations for transferring across the whole dataset.
  # We have a number of foetal ones too to add in here
    annotations = c("age_group", "cell_source", "cell_states", "cell_type", "donor", "gender", "region", "sample")
    for(i in 1:length(annotations)){
      annotation = annotations[i]
      all.so@meta.data[,annotation] = vector(mode=class(as.vector(adult.ss.so@meta.data[,annotation])))
      all.so@meta.data[rownames(adult.ss.so@meta.data),annotation] = as.vector(adult.ss.so@meta.data[,annotation])

    }

# Foetal annotations - we now collect our previous foetal object
```

```
  foetal.integrated = readRDS("output/v_3/foetal_and_results/foetal.integrated.rds")

# Integrating annotations:
# SEX
  colnames(all.so@meta.data)[9] = "sex"
  all.so@meta.data[all.so@meta.data[,9] %in% "Male",9] = "M"
  all.so@meta.data[all.so@meta.data[,9] %in% "Female",9] = "F"

  all.so@meta.data[rownames(foetal.integrated@meta.data),"sex"] = foetal.integrated@meta.data[,"sex"]
  all.so@meta.data[,"sex"] = factor(all.so@meta.data[,"sex"])

# Donor Identifier
  all.so@meta.data[rownames(foetal.integrated@meta.data),"donor"] = as.character(foetal.integrated@meta.data[,"BRC"])
  all.so@meta.data[,"donor"] = factor(all.so@meta.data[,"donor"])

# Sample Identifier
  all.so@meta.data[rownames(foetal.integrated@meta.data),"sample"] = as.character(foetal.integrated@meta.data[,"sample"])
  all.so@meta.data[,"sample"] = factor(all.so@meta.data[,"sample"])

# Region code (AX = apex, RV = right ventricle, LV = left ventricle, SP = Septum, RA = Right atrium, LV = Left Atrium, BA = Base / Aorta)
  all.so@meta.data[rownames(foetal.integrated@meta.data),"region"] = as.character(foetal.integrated@meta.data[,"location"])
  all.so@meta.data[all.so@meta.data[,"region"] == "apex","region"] = "AX"
  all.so@meta.data[all.so@meta.data[,"region"] == "base","region"] = "BA"
  all.so@meta.data[,"region"] = factor(all.so@meta.data[,"region"])

   # also change cell_source
     all.so@meta.data[rownames(foetal.integrated@meta.data),"cell_source"] = as.character(foetal.integrated@meta.data[,"location"])

# Age and sample (ordering)
  all.so@meta.data[,"age.order"] = character()
  all.so@meta.data[rownames(foetal.integrated@meta.data),"age.order"] = as.character(foetal.integrated@meta.data[,"age.order"])
    a.order = gsub("^.*##", "", sort(names(table(paste(all.ss.so@meta.data[,"age_group"], all.ss.so@meta.data[,"donor"], sep="##")))))
    A.names =  c("A1",
                 "A2",
                 "A3",
                 "A4",
                 "A5",
                 "A6")
  all.so@meta.data[rownames(all.ss.so@meta.data),"age.order"] = A.names[match(all.ss.so@meta.data[,"donor"], a.order)]
  all.so@meta.data[,"age.order"] = factor(
    all.so@meta.data[,"age.order"], levels=c("F1", "F2", "F3", "F4", "F5", "F6", "F7", "A1", "A2", "A3", "A4", "A5", "A6")
  )

# Age (days post-conception)
  all.so@meta.data[,"age.daysPC"] = numeric()
  all.so@meta.data[rownames(foetal.integrated@meta.data),"age.daysPC"] = foetal.integrated@meta.data[,"age.daysPC"]

# CRL
  all.so@meta.data[,"size.CRL"] = numeric()
  all.so@meta.data[rownames(foetal.integrated@meta.data),"size.CRL"] = foetal.integrated@meta.data[,"size.CRL"]

# NRL
  all.so@meta.data[,"size.NRL"] = numeric()
  all.so@meta.data[rownames(foetal.integrated@meta.data),"size.NRL"] = foetal.integrated@meta.data[,"size.NRL"]

# HCC
  all.so@meta.data[,"HCC"] = character()
  all.so@meta.data[rownames(foetal.integrated@meta.data),"HCC"] = as.character(foetal.integrated@meta.data[,"HCC"])

  # fetch adult HCC results:: optional - these are annotations from a heart cell classifier I was building at the time
  HCA.predictions = read.csv('HCA2020/HCC_output.version_1/all_ss_counts/all_ss_counts_prediction.tsv',sep="\t")
   rownames(HCA.predictions) = HCA.predictions[,1]
  all.so@meta.data[rownames(HCA.predictions),"HCC"] = as.character(HCA.predictions[,2])

  # And re-level... finally
    all.so@meta.data[,"HCC"] = factor(all.so@meta.data[,"HCC"], levels=levels(foetal.integrated$HCC))

# STAGE
  all.so@meta.data[,"stage"] = factor(c("foetal", "adult")[(gsub(".*-1-|Heart.*","",colnames(all.so)) == "HCA")+1])

# Cell_state
  all.so@meta.data[rownames(foetal.integrated@meta.data),"cell_states"] = "foetal"
  all.so@meta.data[,"cell_states"] = factor(all.so@meta.data[,"cell_states"])

# Integration order
  # This particular annotation details the order in which to integrated the objects.
  #  If we split into groups we're left with 24 groups to integrate
    all.so@meta.data[,"integration.groups"] = paste(all.so@meta.data[,"donor"], all.so@meta.data[,"cell_source"], sep="_")
    all.so@meta.data[,"integration.groups"] = factor(all.so@meta.data[,"integration.groups"])
```

```
# # # # # # annotations done # # # # # #
# Save file for future reference
  saveRDS(all.so,  "RDS_files_v3/all.so.rds")

# Split the object ready for integration
  all.list = SplitObject(all.so, split.by="integration.groups")
  gc()

# Normalise and calculate variable features
  for (i in 1:length(all.list)) {
    all.list[[i]] <- NormalizeData(all.list[[i]], verbose = FALSE)
    all.list[[i]] <- FindVariableFeatures(all.list[[i]])
  }
  gc()
  saveRDS(all.list, "RDS_files_v3/all.list.rds")
######################################################################
```

As you can see we are building a lot of objects for this. Most of these I store in the "RDS_files_v3" directory and we read in when needed for the analysis. We should restart R because it will be using a lot of RAM right now.

Foetal dataset stratified sampling and balancing with adult data

At this stage we are ready to balance our cell types and donors within the foetal dataset and balance the groups with the adult dataset. We will read back in our list, take only our foetal samples, and process them using Seurat's integration pipeline.

```
######################################################################
### Foetal dataset subsampling (stratified subsampling)
# Recommend to restart R and run
  all.list = readRDS("all.list.rds")
  all.list.foetal = lapply(1:13, function(i){all.list[[i]]})
    names(all.list.foetal) = names(all.list)[1:13]

  # Variable genes
    variable.genes = table(unlist(lapply(1:length(all.list), function(i){VariableFeatures(all.list[[i]])})))
  # Take genes that are variable in at least 25 % of all datasets here (This is just for integration purposes...)
    variable.genes = names(variable.genes)[variable.genes >= 6]

  # Free up memory
    rm(all.list)
    gc()

   sample.tree =
      rbind(
        c( -1, -2),               # 1
        c( -3, -4),               # 2
        c( -5, -6),               # 3     Foetal Base + Apex
        c( -7, -8),               # 4
        c( -9,-10),               # 5
        c(-11,-12),               # 6

        c( -13, 1),               # 7
        c(  2,  3),               # 8
        c(  4,  5),               # 9     Foetal Donors
        c(  6,  7),               # 10
        c(  8,  9),               # 11
        c( 10, 11)                # 12
      )

  # Run anchor finding process
    all.anchors.foetal <- FindIntegrationAnchors(object.list = all.list.foetal, dims = 1:30, anchor.features=variable.genes)
    gc()
  # save output
    saveRDS(all.anchors.foetal, "RDS_files_v3/all.anchors.foetal.rds")
  # read anchors
    all.anchors.foetal = readRDS("RDS_files_v3/all.anchors.foetal.rds")

  # Integrate data
    all.foetal.integrated <- IntegrateData(
      anchorset = all.anchors.foetal, sample.tree=sample.tree, dims = 1:30, features.to.integrate = variable.genes
    )
    gc()

#  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #
# After integration
```

```
  # Scale data
    all.foetal.integrated <- ScaleData(all.foetal.integrated, verbose = FALSE)

  # Run PCA
    all.foetal.integrated <- RunPCA(all.foetal.integrated, verbose = FALSE, npcs = 50)
        pca.eig = (all.foetal.integrated[["pca"]]@stdev^2)/sum(all.foetal.integrated[["pca"]]@stdev^2)*100
            plot(pca.eig); abline(h=1, lty=2); which(pca.eig > 1)

  # Run UMAP
    all.foetal.integrated <- RunUMAP(
      all.foetal.integrated, dims = 1:30, umap.method = "uwot", n.components=2, min.dist=0.3, spread=1, n.neighbors=50
    )

  # Edit factors and factor levels...
    all.list = readRDS("RDS_files_v3/all.list.rds")
      factor.annotations = colnames(all.list[[1]]@meta.data)[
        sapply(colnames(all.list[[1]]@meta.data), function(i){class(all.list[[1]]@meta.data[,i])}) == "factor"
      ]
    for(i in factor.annotations){
      all.foetal.integrated@meta.data[,i] = factor(
        all.foetal.integrated@meta.data[,i], levels=levels(all.list[[1]]@meta.data[,i])
      )
    }

# Clustering foetal integrated dataset
# Find neighbours for k == 20
  all.foetal.integrated = FindNeighbors(all.foetal.integrated, k = 20)

# Cluster using Louvain at different resolutions
  all.foetal.integrated = FindClusters(all.foetal.integrated, res=0.1)
  all.foetal.integrated = FindClusters(all.foetal.integrated, res=0.3)
  all.foetal.integrated = FindClusters(all.foetal.integrated, res=0.5)
  all.foetal.integrated = FindClusters(all.foetal.integrated, res=1)
  all.foetal.integrated = FindClusters(all.foetal.integrated, res=2)
gc()


# Looking at this, the resolution 0.5 looks decent enough to begin sampling from.
# The epicardial cluster 14 contains nearly 1600 cells, and there are 21 clusters.
# Approximately: 21 * 1600 = 33,600 (we will lose some as some of these clusters are small)
# Adult dataset has around 30,000 cells

# We'll see how many we get from subsampling - few foetal clusters are also below 1600
   saveRDS(all.foetal.integrated, "RDS_files_v3/all.foetal.integrated.rds")

# We can look at a plot of these clustering resolutions using this code
  par(mfrow=c(2,3))
  par(oma=c(4,4,4,12), mar=c(0,0,0,0), cex=4)
    plot.so(all.foetal.integrated, "integrated_snn_res.0.1", col=rep(Discrete.U,3), label=T, leg=F, main="", xaxt="n")
      mtext("res 0.1", 3, padj=2, adj=0.01, cex=4)
    plot.so(all.foetal.integrated, "integrated_snn_res.0.3", col=rep(Discrete.U,3), label=T, leg=F, main="", xaxt="n", yaxt="n")
       mtext("res 0.3", 3, padj=2, adj=0.01, cex=4)
    plot.so(all.foetal.integrated, "integrated_snn_res.0.5", col=rep(Discrete.U,3), label=T, leg=T, main="", xaxt="n", yaxt="n")
       mtext("res 0.5", 3, padj=2, adj=0.01, cex=4)
    plot.so(all.foetal.integrated, "integrated_snn_res.1", col=rep(Discrete.U,3), label=T, leg=F, main="")
         mtext("res 1", 3, padj=2, adj=0.01, cex=4)
    plot.so(all.foetal.integrated, "integrated_snn_res.2", col=rep(Discrete.U,3), label=T, leg=F, main="", yaxt="n")
         mtext("res 2", 3, padj=2, adj=0.01, cex=4)
    plot.so(all.foetal.integrated, "UPK3B", col=alt.cols, cex=0.6, leg=T, main="", yaxt="n")
         mtext("UPK3B expression", 3, padj=2, adj=0.01, cex=4)

######################################################################
```

Approaching our first visualisation of the foetal dataset after sample integration, we can see the cluster resolutions compared with the expression of UPK3B. This is to positively identify the epicardial cell cluster.

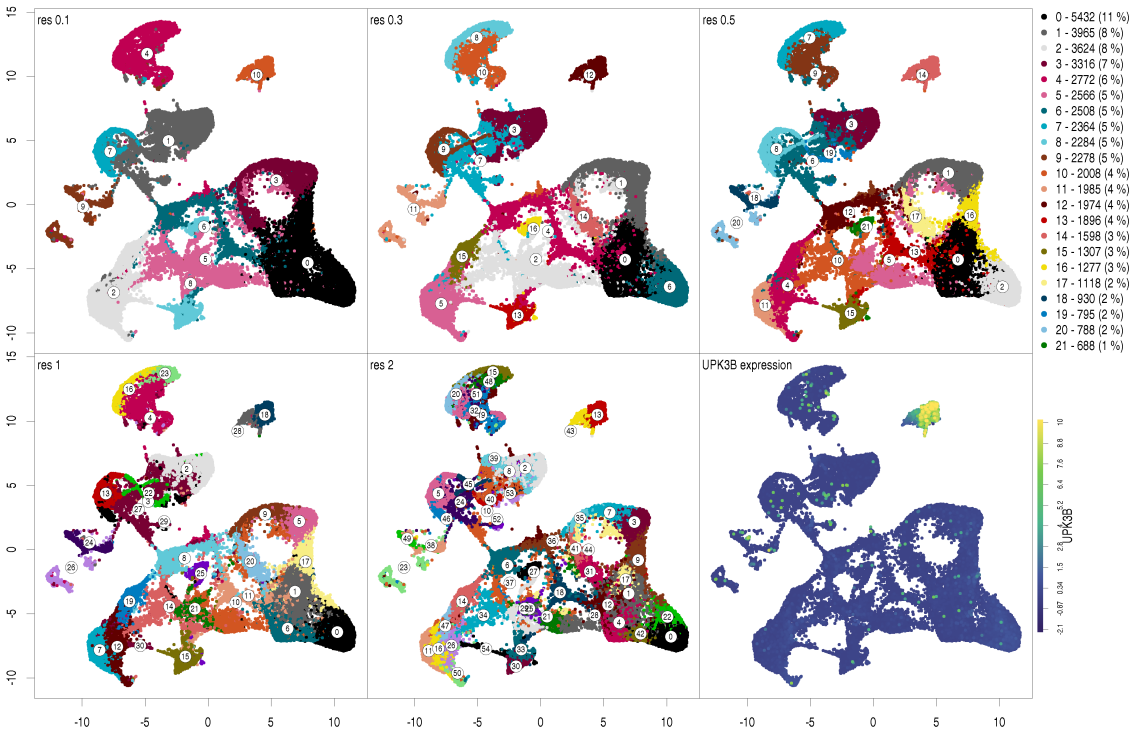

Figure 1: A UMAP plot of the foetal data after intermediate integration in order to identify sub-sampling stratifications based on epicardial cluster size. Clustering is observed over 5 different resolutions and UPK3B expression identifies the epicardial cluster.

Subsampling

We will use cluster resolution 0.5 for foetal cell type stratifications as this is the highest resolution here with an intact epicardial cluster. This leaves us with 1,598 epicardial cells and will be the N for our sampling. This stratified subsampling function was previously used for the adult dataset, here we apply it across the donor and clusters here. It must be said, that there appears to be a sampling bias of fibroblast-like cells (not shown here). The large conglomerate of cells in the bottom right of the UMAPs in Figure 1. Because this group is split into many clusters, each of which we will sample from, we expect to carry this bias into the sub-sampled dataset. With the number of foetal cells in our dataset currently we are unable to parse the distinct types within this group too effectively.

```
######################################################################
# ### Subsampling
  cells2sample = colnames(all.foetal.integrated)
  N = 1598
# Sample 100 samples...
  foetal.ss = sapply(1:100,function(i){stratified_subsample(
    f1 = all.foetal.integrated$integrated_snn_res.0.5[cells2sample],
    f2 = all.foetal.integrated$sample[cells2sample],
    cell_IDs = cells2sample,
    N = N,
    seed = i
    )}
  )

# save output matrix
  write.table(
    foetal.ss,
    file=file.path("output/v_3/adult_foetal_integration/foetal_stratified_subsample_IDs.csv"),
```

```
    sep=",", quote=F, col.names=F, row.names=F
  )
######################################################################
```

Integration

We begin stage integration by removing further unnecessary cells. At this point I realise that the adult dataset still contains doublets so we begin by removing these. Then we take only the foetal subsampled cells and split our object into our defined integration groups (based on donor + cell source / region). You will notice that we are saving objects often here in case of memory-related crashes, we could pick up where we left off at the last stage.

```
######################################################################
# ### Preparation for subsample integration using Reciprocal PCA
# Read that large annotated Seurat object of adult and foetal cells
  all.so = readRDS("RDS_files_v3/all.so.rds")

# Remove those doublets... come on now...
  all.so = subset(all.so, cells=colnames(all.so)[all.so$cell_states != "doublets"])

# Now we subset to include only the stratified sample of foetal cells and adult cells
  all.so = subset(all.so, cells=c(foetal.ss[,1][order(all.so$donor[foetal.ss[,1]])], colnames(all.so)[grep("HCAHeart", colnames(all.so))]))

# We begin by splitting our object by the relevant factors:
# Here we defined a previous factor of integration groups.
  all.subsample.list = SplitObject(all.so, split.by="integration.groups")

# And save the output
  saveRDS(all.subsample.list, "RDS_files_v3/all.subsample.list.rds")
  gc()

# For each part in subsample.list, normalise and find variable features
    for (i in 1:length(all.subsample.list)) {
        all.subsample.list[[i]] <- NormalizeData(all.subsample.list[[i]], verbose = FALSE)
        all.subsample.list[[i]] <- FindVariableFeatures(all.subsample.list[[i]])
      }
   gc()

# Create a variable genes object
  # Skim all variable genes in datasets
    variable.genes = table(unlist(lapply(1:length(all.subsample.list), function(i){VariableFeatures(all.subsample.list[[i]])})))
  # Take genes that are variable in at least 1/4 of all datasets here (This is just for integration purposes...)
    variable.genes = names(variable.genes)[variable.genes >= 6]
    saveRDS(variable.genes, "RDS_files_v3/variable.genes.rds"))
######################################################################
```

Identify integration anchors:

```
######################################################################
# ### Integration via RPCA:
  # Reciprocal PCA integration
    # we need to apply a PCA to the objects:
      all.subsample.list <- lapply(X = all.subsample.list, FUN = function(x) {
        x <- ScaleData(x, features = variable.genes, verbose = FALSE)
        x <- RunPCA(x, features = variable.genes, verbose = FALSE)
      })

  # Run anchor finding process
    all.subsample.anchors <- FindIntegrationAnchors(object.list = all.subsample.list, dims = 1:30,
      anchor.features=variable.genes,
      reduction="rpca",
      k.anchor = 5
    )
    gc()

  # save anchors object
    saveRDS(all.subsample.anchors, "RDS_files_v3/all.subsample.anchors_RPCA_30dims.rds")
    gc()
######################################################################
```

Run the integration steps. At this point we will restart R and open a new session because of memory. The following code should suffice for integration of the stages. Following the Seurat vignette we apply some configuration settings, set up a custom sample tree for integration (see Ext Fig 1 on our manuscript)

```
######################################################################
# ### Close R and restart --- lots of RAM used here...
```

```
# Load our objects
 library(Seurat)
 variable.genes = readRDS("RDS_files_v3/variable.genes.rds")
 all.subsample.anchors = readRDS("RDS_files_v3/all.subsample.anchors_RPCA_30dims.rds") ## k == 5

 # Memory options
   options(future.globals.maxSize = 20000 * 1024^2)

 # Custom integration order - integers here refer to list indice of samples in our all.subsample.anchors list object
       # We begin with merging adult datasets - nuclei to cells
         # 1: 15 + 20
         # 2: 16 + 21
         # 3: 17 + 22
         # 4: 18 + 23
         # 5: 19 + 24

     # Then we merge adult donors
         # 6: 14 + 15:20
         # 7: 16:21 + 17:22
         # 8: 18:23 + 19:24
         # 9: 14:15:20 + 16:17:21:22
         # 10: 14:15:16:17:20:21:22  18:19:23:24

       # We then merge foetal data source (base vs apex)
         # 11: 2 + 3
         # 12: 4 + 5
         # 13: 6 + 7
         # 14: 8 + 9
         # 15: 10 + 11
         # 16: 12 + 13

     # Then we merge by foetal donors
         # 17: 1 + 2:3
         # 18: 4:5 + 6:7
         # 19: 8:9 + 10:11
         # 20: 12:13 + 1:2:3
         # 21: 4:5:6:7 + 8:9:10:11
         # 22: 4:5:6:7:8:9:10:11 + 1:2:3:12:13

 # Lastly we merge by stages
         # 23: 1:2:3:4:5:6:7:8:9:10:11:12:13 + 14:15:16:17:18:19:20:21:22:23:24

 # END

 # Thus we need to create a custom sample tree:
 sample.tree =
     rbind(
       c(-15,-20),               # 1
       c(-16,-21),               # 2
       c(-17,-22),               # 3      Adult Nuclei + Cells
       c(-18,-23),               # 4
       c(-19,-24),               # 5

       c(-14,  1),               # 6
       c(  2,  3),               # 7
       c(  4,  5),               # 8      Adult Donors
       c(  6,  7),               # 9
       c(  8,  9),               # 10

       c( -2, -3),               # 11
       c( -4, -5),               # 12
       c( -6, -7),               # 13     Foetal Base + Apex
       c( -8, -9),               # 14
       c(-10,-11),               # 15
       c(-12,-13),               # 16

       c( -1, 11),               # 17
       c( 12, 13),               # 18
       c( 14, 15),               # 19     Foetal Donors
       c( 16, 17),               # 20
       c( 18, 19),               # 21
       c( 20, 21),               # 22

       c( 10, 22)                # 23     Both stages
     )

 # Integrated data
   all.subsample.integrated <- IntegrateData(
     anchorset = all.subsample.anchors, sample.tree=sample.tree, dims = 1:30, features.to.integrate = variable.genes
   )
```

```
    gc()

# Edit factors and factor levels...
  # load in our annotated list
  all.subsample.list = readRDS("RDS_files_v3/all.subsample.list.rds")
    factor.annotations = colnames(all.subsample.list[[14]]@meta.data)[
      sapply(colnames(all.subsample.list[[14]]@meta.data), function(i){class(all.subsample.list[[14]]@meta.data[,i])}) == "factor"
    ]
  for(i in factor.annotations){
    all.subsample.integrated@meta.data[,i] = factor(
      all.subsample.integrated@meta.data[,i], levels=levels(all.subsample.list[[1]]@meta.data[,i])
    )
  }
  all.subsample.integrated$cell_states = factor(all.subsample.integrated$cell_states)

# dims 1:30 and variable genes which are variable in at least 6 samples (1/4 of dataset)
  saveRDS(all.subsample.integrated, "RDS_files_v3/all.subsample.integrated_RPCA_dims30_vargenes6samples.rds")
#####################################################################
```

Summary

This report ends the initial sample and object preparation. The next report will go through the clustering and annotation of the dataset which is where the results truly begin. We will use the object **all.subsample.integrated** a lot going forward, so keep track of it! In this stage we have a lot to add during the analysis steps. However, there is a complete and up to date version of the **all.subsample.integrated** object stored online on our submission to GEO which one can use throughout the rest of the analysis.