Seurat single cell pipeline

After sequencing and running cellranger in the server we are going to focus in analyzing the data using Seurat R package.

Seurat is an R package designed for QC, analysis, and exploration of single-cell RNA-seq data. It enables users to identify and interpret sources of heterogeneity from single-cell transcriptomic measurements, and to integrate diverse types of single-cell data.
https://satijalab.org/seurat/index.html

Seurat has many vignettes and applications such as analysis, visualization, and integration of spatial datasets or integration of scRNA-seq and scATAC-seq data.

For this occasion, we will follow these vignettes and make a custom script from them.
https://satijalab.org/seurat/articles/pbmc3k_tutorial.html
https://satijalab.org/seurat/articles/integration_rpca.html

In the first script called scRNASeq_Integration.R we start by reading the data from cellranger output and create a Seurat object. The object serves as a container that contains both data (like the count matrix) and analysis (like PCA, or clustering results) for a single-cell dataset.
Depending on which data we are receiving from Celrranger multiplexed still or demultiplexed the script will run an additional demultiplex step or not.
Then we perform a QC trimming. We filter cells that have unique feature counts over 9000 or less than 50, cells with more than 60000 and we filter cells that have >5% mitochondrial counts.
As an output, after trimming violin plots of the distribution of number of genes, number of counts and number of mitochondrial percentage will be printed for each sample.

Then we normalize data a global-scaling normalization method “LogNormalize” that normalizes the feature expression measurements for each cell by the total expression, multiplies this by a scale factor (10,000 by default), and log-transforms the result.

Then we calculate a subset of features that exhibit high cell-to-cell variation in the dataset (i.e, they are highly expressed in some cells, and lowly expressed in others). This helps to highlight biological signal in single-cell datasets.

The usage of this first script will be:

```
Rscript scRNASeq_Integration.R
```

If running in the server you need to create a sh script (Integration_R.sh) similar to this one:

```
#!/bin/bash
#
#$
#$ -cwd
#$ -j y
#$ -N single_cell
#$ -S /bin/bash
#$ -l h_vmem=10g


Rscript ~/pipelines/scRNAseq/scRNASeq_Integration.R

#done
Integration_R.sh (END)
```

And run:

```
qsub scRNASeq_Integration_TotalSeq.sh
```

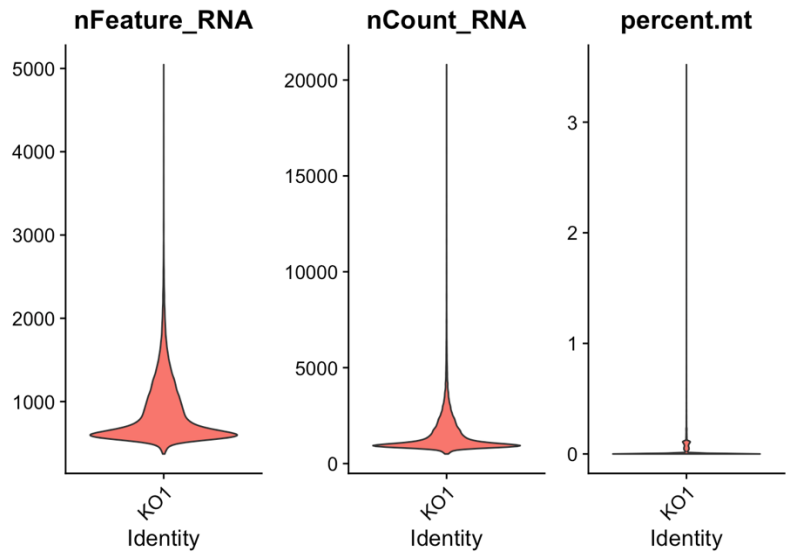
If COUNT option is selected the demultiplex step will be performed.

```r
#FUNCTION_1: Create demultiplexed objects for each multiplexed files. This coming from Cellranger COUNT mode
if (config$cellranger_option=="COUNT"){
create_object <- function(sample_list,data_dir){

  Sample_dataset <- Read10X(data_dir)
  Sample_expression <- CreateSeuratObject(counts = Sample_dataset[["Gene Expression"]], project = "Sample_test")
  Sample_antibody <- CreateSeuratObject(counts = Sample_dataset[["Antibody Capture"]], project = "Sample_test")

  # filtered the cells for you, but perform this step for clarity.
  joint.bcs <- intersect(colnames(Sample_expression), colnames(Sample_antibody))

  # Subset RNA and HTO counts by joint cell barcodes
  Sample_expression <- Sample_expression[, joint.bcs]
  Sample_antibody <- as.matrix(Sample_antibody@assays$RNA@counts[, joint.bcs])

  # Normalize RNA data with log normalization
  Sample.hashtag <- NormalizeData(Sample_expression)
  # Find and scale variable features
  Sample.hashtag <- FindVariableFeatures(Sample.hashtag, selection.method = "mean.var.plot")
  Sample.hashtag <- ScaleData(Sample.hashtag, features = VariableFeatures(Sample.hashtag))

  # Add HTO data as a new assay independent from RNA
  Sample.hashtag[["HTO"]] <- CreateAssayObject(counts = Sample_antibody, project = "Sample_test")


  #Demultiplex cells based on HTO enrichment
  # Normalize HTO data, here we use centered log-ratio (CLR) transformation
  Sample.hashtag <- NormalizeData(Sample.hashtag, assay = "HTO", normalization.method = "CLR")

  # If you have a very large dataset we suggest using k_function = 'clara'. This is a k-medoid
  # clustering function for large applications You can also play with additional parameters (see
  # documentation for HTODemux()) to adjust the threshold for classification Here we are using the
  # default settings
  Sample.hashtag <- HTODemux(Sample.hashtag, assay = "HTO", positive.quantile = 0.99)
  #The sample assigment is saved in the metadata dataframe inside the object Sample.hahstag.
```

```
    #Visualize demultiplexing results
    # Global classification results
    #table(Sample.hashtag$HTO_classification.global)

    # Group cells based on the max HTO signal
    Idents(Sample.hashtag) <- "HTO_maxID"
    RidgePlot(Sample.hashtag, assay = "HTO", features = rownames(Sample.hashtag[["HTO"]])[1:2], ncol = 2)


    Idents(Sample.hashtag) <- "HTO_classification.global"

    png(file ="nCount_violin.png",
      width = 1000,
      height = 1200)

    nCount_violin <- VlnPlot(Sample.hashtag, features = "nCount_RNA", pt.size = 0.1, log = TRUE)
    print(nCount_violin)
    dev.off()


    # First, we will remove negative cells from the object
    Sample.hashtag.subset <- subset(Sample.hashtag, idents = "Negative", invert = TRUE)

    # Calculate a distance matrix using HTO
    hto.dist.mtx <- as.matrix(dist(t(GetAssayData(object = Sample.hashtag.subset, assay = "HTO"))))

    # Calculate tSNE embeddings with a distance matrix
    Sample.hashtag.subset <- RunTSNE(Sample.hashtag.subset, distance.matrix = hto.dist.mtx, perplexity = 100)
    #DimPlot(Sample.hashtag.subset)

    # Extract the singlets
    Sample.singlet <- subset(Sample.hashtag, idents = "Singlet")
    #Normalize data and find variable genes.
    Sample.singlet <- NormalizeData(Sample.singlet)
    Sample.singlet <- FindVariableFeatures(Sample.singlet, selection.method = "vst", nfeatures = config$HVF_selection)
}

Object_list <- mapply(FUN=create_object,sample_names, cell_ranger_output)

}
```

If MULTI option is selected it will go to QC, trimming and integration steps.

```r
#These directory have to have inside folder for each samples and inside the three necessary files.
cell_ranger_output <- list.dirs(config$data_location,recursive = F)
print(cell_ranger_output)
#Now I get the names to call the objects and that have to be the same as the folder.
sample_names <- sub('.*\\/', '', cell_ranger_output)

#Load metadata information and get the samples.
#FUNCTION_1
create_object <- function(sample_list,data_dir){
  Sample_dataset <- Read10X(data_dir)
  # Initialize the Sample_object object with the raw (non-normalized data).
  Sample_object <- CreateSeuratObject(counts = Sample_dataset, project = paste0(sample_list), min.cells = 3, min.features = 200)
  Sample_object[["percent.mt"]] <- PercentageFeatureSet(Sample_object, pattern = "^mt-")
  #Filtering
  Sample_object <- subset(Sample_object, subset = nFeature_RNA > 50 & nFeature_RNA < 9000 & nCount_RNA < 60000 & percent.mt < 5)

  #png(file =paste0(config$quality_location,sample_list,"_Violin_data_quality.png"),
  #width = 1000,
  #height = 1200)

  quality_plot <- VlnPlot(Sample_object, features = c("nFeature_RNA", "nCount_RNA", "percent.mt"), ncol = 3, pt.size = 0)
  print(quality_plot)
  #dev.off()

  #Normalize data and find variable genes.
  Sample_object <- NormalizeData(Sample_object)
  Sample_object <- FindVariableFeatures(Sample_object, selection.method = "vst", nfeatures = 2000)
}
#Run the function through all samples and get one object per sample.
Object_list <- mapply(FUN=create_object,sample_names, cell_ranger_output)
```



We jump then to integrate the data. If the object has more than one sample it will integrate all the samples in one object. If we only have one sample it will skip this point.

```
if (length(Object_list) > 1) {
  #In order to integrate data select integration features through the objects.
  features <- SelectIntegrationFeatures(object.list = Object_list, nfeatures = 2000)

  anchors <- FindIntegrationAnchors(object.list = Object_list, anchor.features = features)

  #Create 'integrated' data assay through all the samples.
  Integrated_data <- IntegrateData(anchorset =anchors)

  DefaultAssay(Integrated_data) <- "integrated"
} else {
  Integrated_data <- Object_list[[1]]
}
```

Once we have the integrated object (or the single sample object) we will apply a linear transformation ('scaling'). Scaling the data will shifts the expression of each gene, so that the mean expression across cells is 0 and the variance across cells is 1.

```
message("Load metadata and add information to obejct")
#Add to metadata condition information about each sample.
metadata <- read.csv(config$metadata_file, header = T)
cell_names <- rownames(Integrated_data@meta.data)
Integrated_data@meta.data <- base::merge(Integrated_data@meta.data, metadata, by="orig.ident"
rownames(Integrated_data@meta.data) <- cell_names
Integrated_data@meta.data$Condition <- relevel(factor(Integrated_data@meta.data$Condition), r


#Scale data for visualization.
Integrated_data <- ScaleData(Integrated_data, verbose = FALSE)

message(paste0("Saving integrated data in", config$seurat_object))
save(Integrated_data, file=paste0(config$seurat_object,config$seurat_object_name))
```

Integrated data will be saved so we don't have to run the integration again.

After this we will jump to the second script scRNASEq_dim_reduction.R.

Rscript scRNAseq_dim_reduction.R

The usage of this script in the server will be:

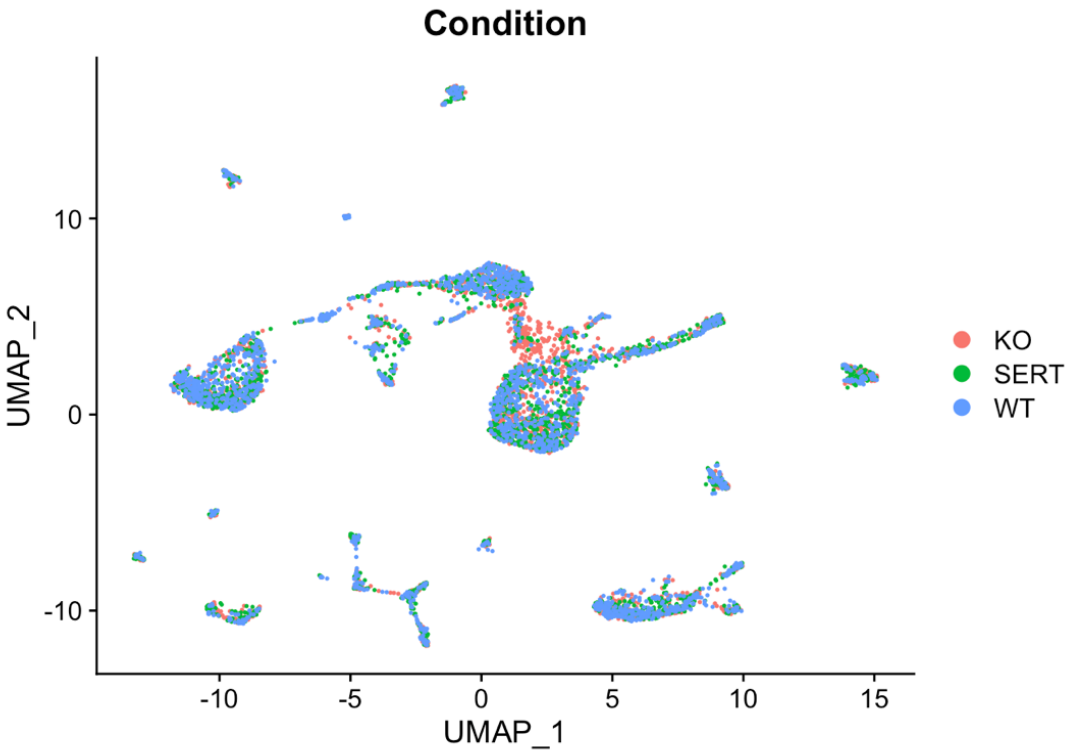
Next we perform PCA on the scaled data and to clusters the cells Seurat will first construct a KNN graph based on the Euclidean distance in PCA space and refine the edge weights between any two cells based on the shared overlap in their local neighborhoods. To cluster the cells, Seurat next apply modularity optimization techniques such as the Louvain algorithm (default) or SLM to iteratively group cells together, with the goal of optimizing the standard modularity function.

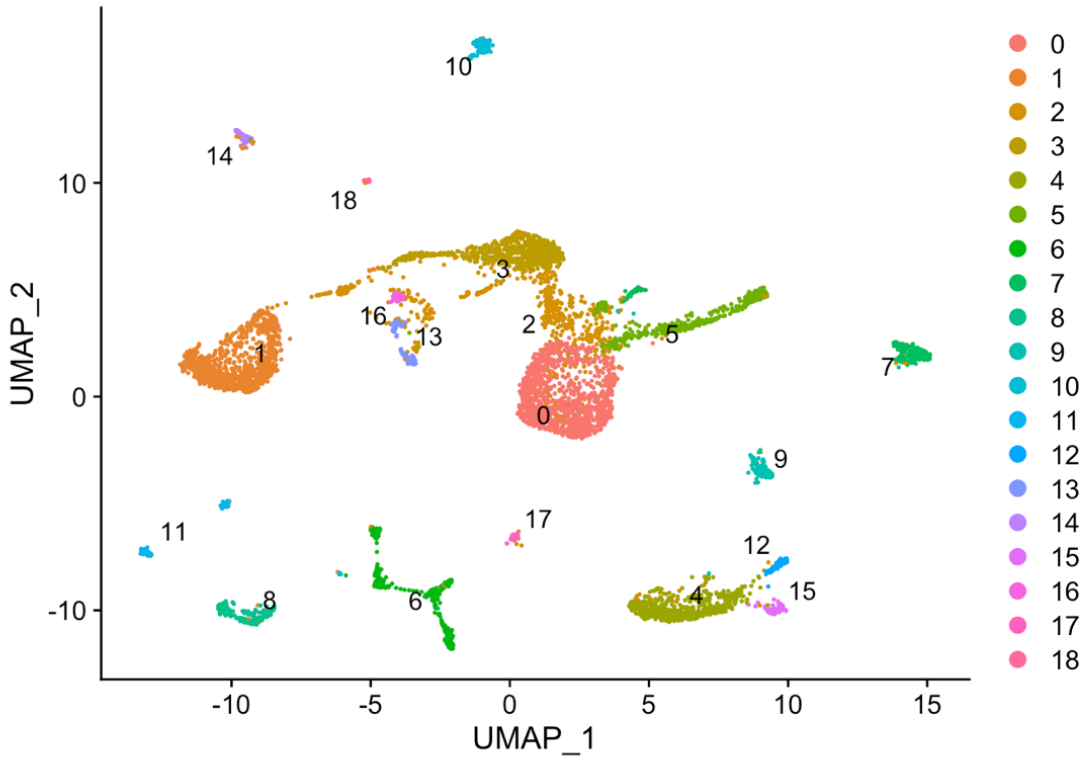
Finally, to place similar cells together in low-dimensional space we apply non-linear dimensional reduction (UMAP or tSNE) and we plot our results based on Condition of the samples and found clusters.

For this step, we will add to the Seurat object metadata information in which the condition of each sample is provided. Here we set from config.yml the reference condition so the further downstream works well. We could add more information to the metadata such as gender, age...

```
#Add to metadata condition information about each sample.
metadata <- read.csv(config$metadata_file, header = T)
cell_names <- rownames(Integrated_data@meta.data)
Integrated_data@meta.data <- base::merge(Integrated_data@meta.data, metadata, by="orig.ident", all=T)
rownames(Integrated_data@meta.data) <- cell_names


#Scale data for visualization.
Integrated_data <- ScaleData(Integrated_data, verbose = FALSE)

#Dimensionality reduction
Integrated_data <- RunPCA(Integrated_data, npcs = 30, verbose = FALSE)
Integrated_data <- RunUMAP(Integrated_data, reduction = "pca", dims = 1:30)
Integrated_data <- FindNeighbors(Integrated_data, reduction = "pca", dims = 1:30)
Integrated_data <- FindClusters(Integrated_data, resolution = 0.2)

#Save the integrated object so we can play with it.
save(Integrated_data, file=paste0(config$seurat_object,config$seurat_object_name))
```

```r
#Plot integration UMAP and clustering UMAP.

#png(file=paste0(config$integration_folderation_folder,"UMAP_integration_samples.png"),
#width = 1200,
#height = 1200)
plot1 <- DimPlot(Integrated_data, reduction = "umap", group.by = "Condition")
print(plot1)
#dev.off()

#png(file =paste0(config$integration_folder,"UMAP_clustering.png"),
#width = 1200,
#height = 1200)
p2 <- DimPlot(Integrated_data, reduction = "umap", label = TRUE, repel = TRUE)

print(p2)
#dev.off()
```





In the third script called scRNAseq_downstream_analysis.R we will work in the downstream analysis.

The usage of this first script will be:

```
Rscript scRNAseq_downstream_analysis.R
```

If running in the server you need to create a sh script (Downstream_Seurat.sh) similar to this one:

```
[maider@hdx2-d scRNAseq_test]$ vim Downstream_Seurat.sh

#!/bin/bash
#
#$
#$ -cwd
#$ -j y
#$ -N single_cell_DE
#$ -S /bin/bash
#$ -l h_vmem=10g


Rscript ~/pipelines/scRNAseq/scRNAseq_downstream_analysis.R

#done
~
~
```

And run:

```
qsub Downstream_Seurat.sh
```

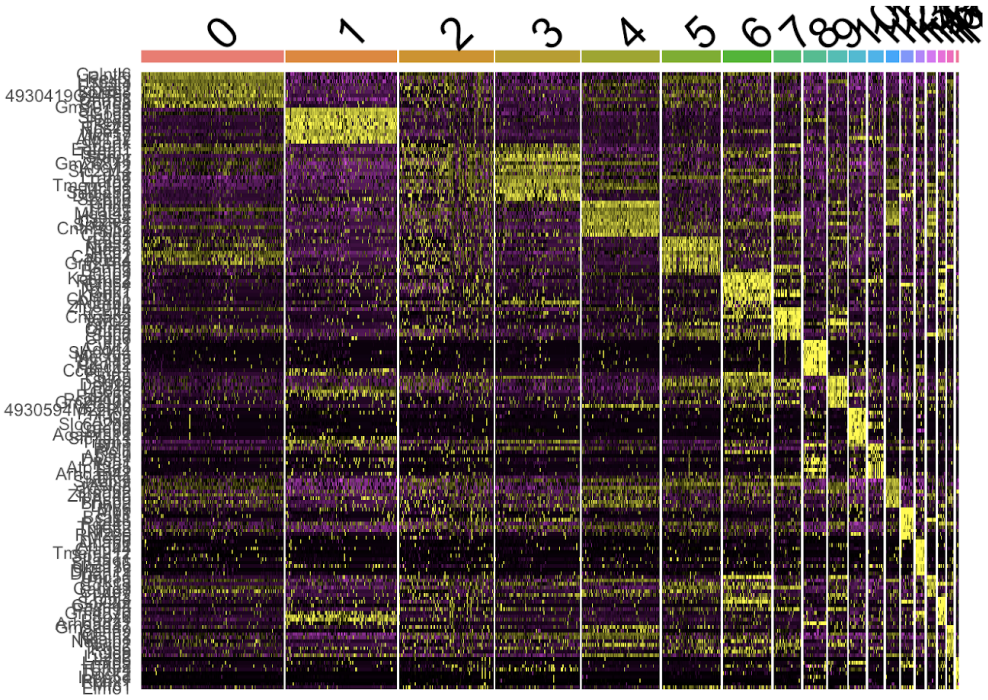
```
#!/bin/bash
#
#$
#$ -cwd
#$ -j y
#$ -N dimensionality_reduction
#$ -S /bin/bash
#$ -l h_vmem=10g


Rscript scRNASEq_dim_reduction.R

#done
~
~
```

Once clusters are defined, we will try to annotate the different clusters (cell types) by the expression of markers. These markers are differentially expressed (DE) genes between clusters.

So in order to identify markers for each of the clusters the DE analysis will be Cluster_1 vs Rest_of_all_Clusters.

The output of this function will be a dataframe with DE genes for each cluster. Also we plot a heatmap in which the top markers for each of the clusters are shown and we can see how specific are those markers for each cluster.

```
Markers <- FindAllMarkers(Integrated_data, min.pct = 0.25, logfc.threshold = 0.25)

#Save results
write.table(Markers, paste0(config$integration_folder,"Markers.csv"), quote = F, row.names = F,

#Plot top markers in a heatmap
Markers %>%
  group_by(cluster) %>%
  top_n(n = 10, wt = avg_log2FC) -> top10

#png(file =paste0(config$integration_folder,"Top_markers_heatmap.png"),
#width = 1600,
#height = 1600)
Heatmap <- DoHeatmap(Integrated_data, features = top10$gene, size = 8) + NoLegend()
print(Heatmap)
#dev.off()
```



Once we have annotated the cell types we can proceed to the Differential Analysis between the different conditions. Since we are expecting samples from three different conditions, we set our reference condition in the config.yml file.

We also create in this step files of ranked genes based on fold change for each cluster to use as input in GSEA.

```
if(length(unique(Integrated_data@meta.data$orig.ident)) >1) {
  #Get DE for each cluster between conditions. Here we can select which conditions to compare.
  DE0 <- function(cluster0, Reference, KO){
    cell1 = rownames(Integrated_data@meta.data)[Integrated_data@meta.data$seurat_clusters == cluster0 & Integrated_data@meta.data$Condition %in
    cell2 = rownames(Integrated_data@meta.data)[Integrated_data@meta.data$seurat_clusters == cluster0 & Integrated_data@meta.data$Condition %in
    DE0 = FindMarkers(Integrated_data, ident.1 = cell1, ident.2 = cell2,logfc.threshold=0.1)
    DE0$cluster = cluster0
    return(DE0)
  }


  cluster_list <- as.numeric(levels(Integrated_data@meta.data$seurat_clusters))
  #Select which comparison you want to do.
  conditions <- unique(metadata$Condition)

  #Here a create a df with all possible comparison depending of how many conditions we have.
  possible_conditions <- t(combn(conditions,2))

  #I run the DE analysis plus I create the files for enrichment analysis.
  for (row in 1:nrow(possible_conditions)) {
    print(paste0("Calculating DEG for ", possible_conditions[row, ][1], "_vs_", possible_conditions[row, ][2]))
    DE_list1 <- lapply(FUN=DE0,cluster_list, possible_conditions[row, ][1], possible_conditions[row, ][2])
    DE_results1 = do.call(rbind,DE_list1)
    write.table(DE_results1,
                file = paste0(config$DE_folder,possible_conditions[row, ][1], "_vs_", possible_conditions[row, ][2], "_DE_results.tsv"),
                sep = ",",
                col.names = T,
                row.names = F,
                quote = F)
```

```
    #Here I start to create the files for the enrichment analysis.
    dir.create(paste0(config$enrichment_folder,possible_conditions[row, ][1],"_",possible_conditions[row, ][2]))
    print(paste0("Creating directory for ",possible_conditions[row, ][1],"_",possible_conditions[row, ][2]))
    cluster_names <- unique(DE_results1$cluster)
    DE_results1$Symbol <- rownames(DE_results1)
    for (i in cluster_names){
      DE_cluster <- DE_results1 %>% filter(cluster==i)
      DE_cluster$value <- as.numeric(DE_cluster$avg_log2FC > 0)
      DE_cluster2 <- DE_cluster %>%
        dplyr::arrange(p_val) %>%
        dplyr::mutate(log_pvalue = ifelse(value == 0, (-log10(DE_cluster$p_val)*-1), (-log10(DE_cluster$p_val)*1))) %>% dplyr::select(Symbol, l
      write.table(DE_cluster2,
                  paste0(config$enrichment_folder,possible_conditions[row, ][1],"_",possible_conditions[row, ][2],"/Cluster_",i, "_rank.rnk"),
                  sep = "\t",
                  row.names = F,
                  col.names = F,
                  +quote = F)
    }
  }
}
```