~/phd-projects/skpediatrictransplant2023_markdown

2023-05-24

This script reproduces the general results from the manuscript using a subset of the data for faster runtimes. The source script contains the packages the user would have to install to run this script. A markdown file is provided as well.

```
knitr::opts_knit$set(root.dir = "~/phd-projects")
source('~/phd-projects/skpediatrictransplant2023/scripts/skpediatrictransplant2023-functions.R')
```

```
## Loading required package: SnowballC

## Loading required package: permute

## Loading required package: lattice

## This is vegan 2.6-4

## Loading required package: igraph

##
## Attaching package: 'igraph'

## The following object is masked from 'package:vegan':
##
##     diversity

## The following object is masked from 'package:permute':
##
##     permute

## The following objects are masked from 'package:stats':
##
##     decompose, spectrum

## The following object is masked from 'package:base':
##
##     union

## Thanks for using FlowSOM. From version 2.1.4 on, the scale
## parameter in the FlowSOM function defaults to FALSE

## Loading required package: ggplot2
```

```
## Loading required package: limma

## Loading required package: BiocParallel

## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v tibble  3.1.7      v dplyr   1.0.10
## v tidyr   1.2.0      v stringr 1.4.0
## v readr   2.1.2      v forcats 0.5.1
## v purrr   0.3.4

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::as_data_frame() masks tibble::as_data_frame(), igraph::as_data_frame()
## x purrr::compose()       masks igraph::compose()
## x tidyr::crossing()      masks igraph::crossing()
## x dplyr::filter()        masks stats::filter()
## x dplyr::groups()        masks igraph::groups()
## x dplyr::lag()           masks stats::lag()
## x dplyr::select()        masks MASS::select()
## x purrr::simplify()      masks igraph::simplify()
```

```
## data
md = read.csv("skpediatrictransplant2023/data/sample_metadata.txt",sep="\t",  stringsAsFactors = FALSE,
     mutate(sample = as.character(sample))


## this is subsetted dataset sampled to represent every cell type in each subject.
## thus is not an accurate representation of the data and only serves as a subsetted template for examp
df = readRDS('skpediatrictransplant2023/data/sc_data_subset.rds')
```

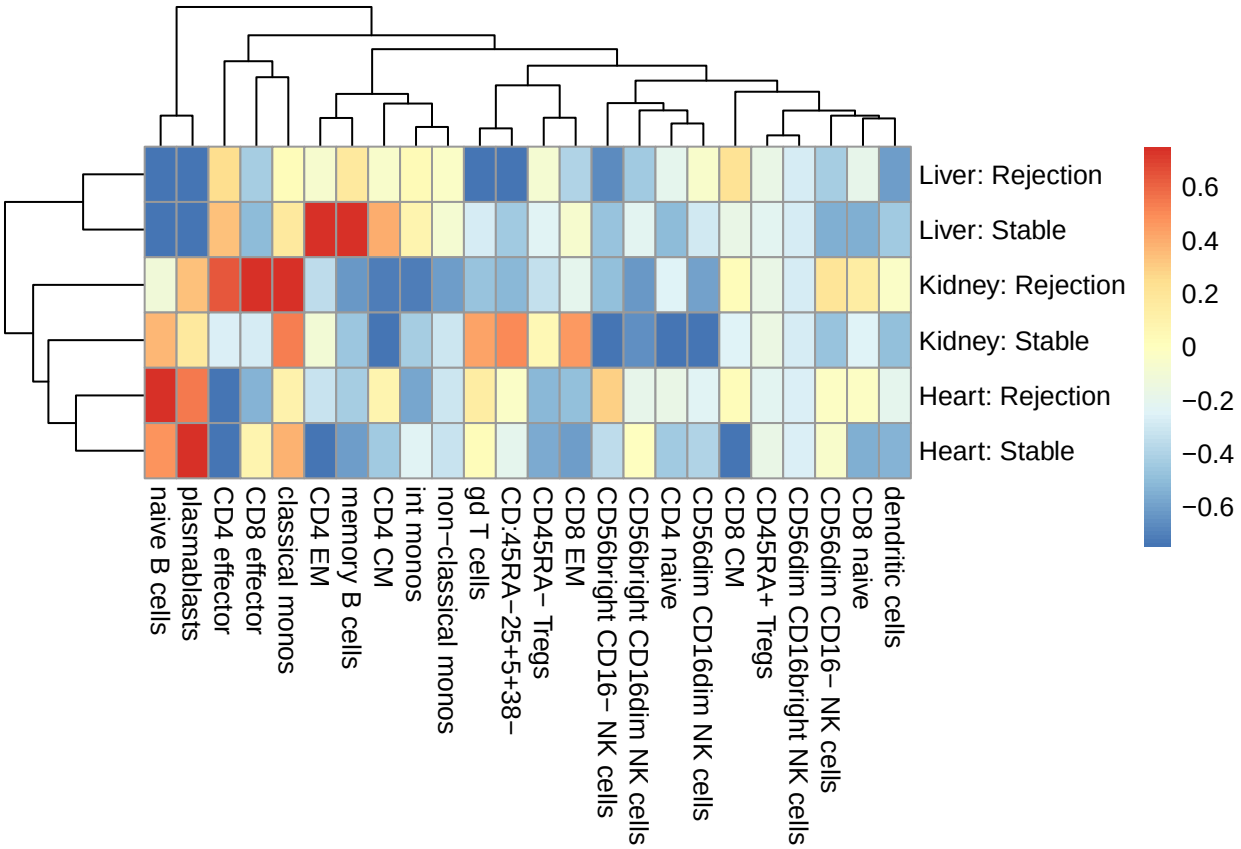
```
## calculate proportions per sample
df_subset_proportions = df %>%
     group_by(celltype, sample) %>%
     summarize(count = n()) %>%
     group_by(sample) %>%
     mutate(percentage = count/sum(count)*100)

## calculate proportions per sample  in each lineage
df_subset_proportions = df %>%
     group_by(sample, lineage, celltype) %>%
     summarize(count = n()) %>%
     group_by(sample, lineage) %>%
     mutate(proportion = count/sum(count)*100) %>%
     ungroup()


## proportions per sample in each lineage, these are manual gating results
data.input_prop =  readRDS( 'skpediatrictransplant2023/data/celltype_proportions.rds')
```

```
## join metadata
data.input_prop = data.input_prop %>%
     left_join(md %>% dplyr::select(sample, Graft, Graft_Health))
```

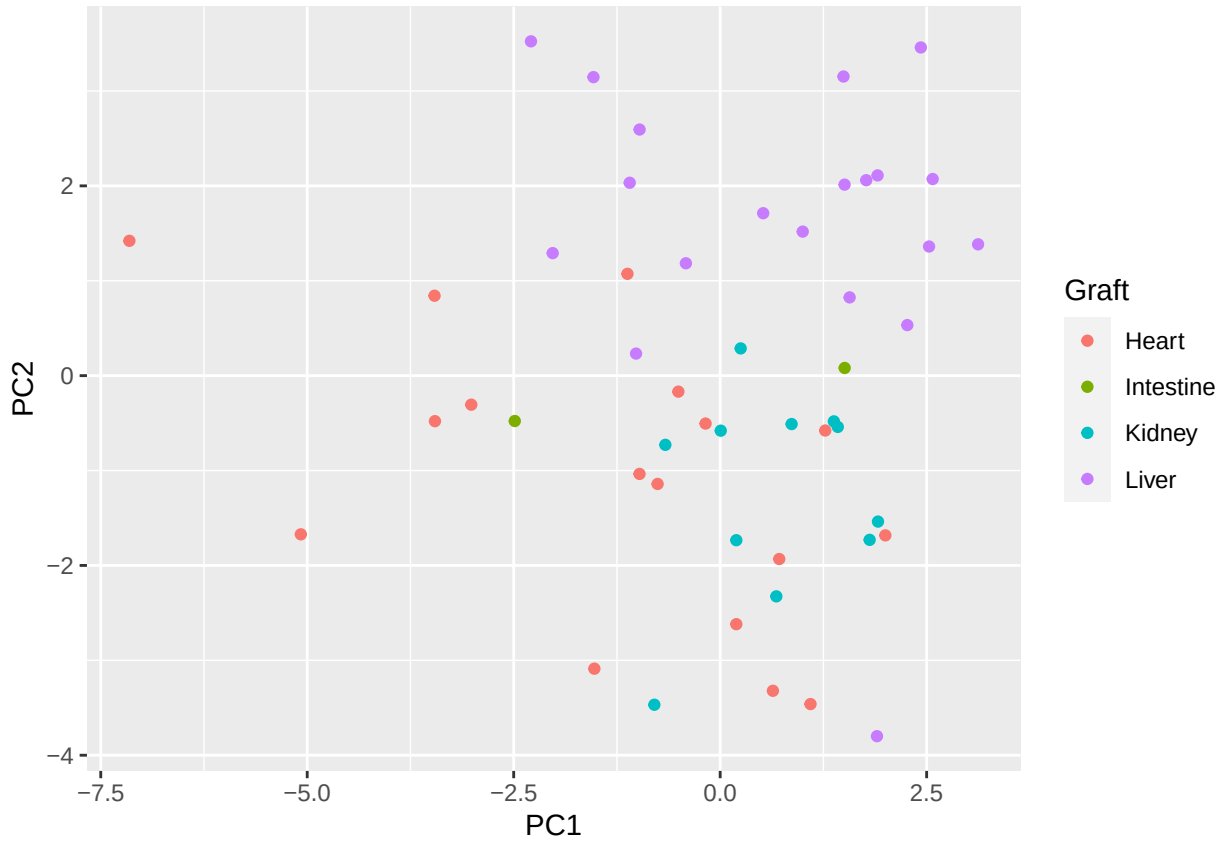
```
## Joining, by = "sample"
```

```
## filter to keep only relevant samples
data.input_prop_filt = data.input_prop %>%
     dplyr::filter(Graft %in% c('Heart','Liver','Kidney','Intestine'), Graft_Health %in% c("Stable","Re


## terminal gated populations only
celltypes_relevant = c('CD4 CM','CD4 effector','CD4 EM','CD4 naive','CD45RA- Tregs','CD45RA+ Tregs','CD
```

```
## summary heatmap
hm.input = data.input_prop_filt %>%
     dplyr::filter(Graft != 'Intestine') %>% ## remove intestine since there are few samples, as done i
     mutate_at(celltypes_relevant, scale) %>%
     group_by(Graft, Graft_Health)%>%
     summarize_at(celltypes_relevant, median)%>%
     ungroup() %>%
     mutate(group = paste(Graft, Graft_Health, sep =': '))

## shunt values to  minimize impact of celltypes with large standard deviations to clarify color gradie
hm.input = lapply(hm.input, function(col){if (is.numeric(col)){
     return(ifelse(abs(col)>= 0.75, 0.75*sign(col), col))
} else {
     return(col)
}
}) %>% bind_cols()
pheatmap::pheatmap(hm.input %>% dplyr::select(-Graft,-Graft_Health) %>% column_to_rownames('group'))
```




```
train.x = data.input_prop_filt %>% dplyr::select(all_of(celltypes_relevant))
## perform PCA
pca.results = pca_function(data.input_prop_filt, celltypes_relevant)
ggplot(pca.results, aes(x = PC1, y = PC2, color = Graft)) + geom_point()
```



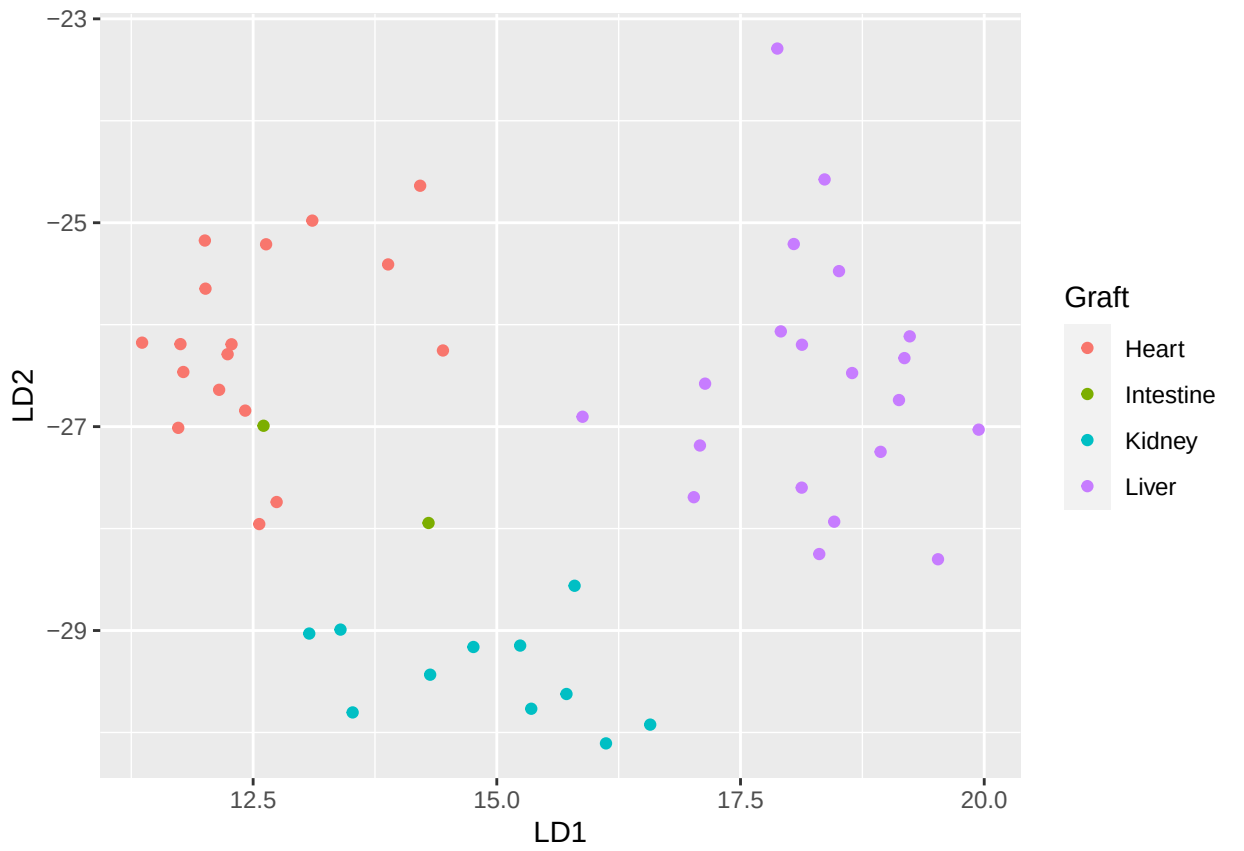

```
## perform permanova analysis
# library(vegan)
dist.res<-vegan::vegdist(pca.results %>% dplyr::select(all_of(paste0('PC',1:10))), method='euclidean')
mds.res = cmdscale(dist.res)
adonis.res<- vegan::adonis2(dist.res~ Graft  + Graft_Health , data=pca.results, permutations = 999, meth
adonis.res
```

```
## Permutation test for adonis under reduced model
## Terms added sequentially (first to last)
## Permutation: free
## Number of permutations: 999
##
## vegan::adonis2(formula = dist.res ~ Graft + Graft_Health, data = pca.results, permutations = 999, me
##              Df SumOfSqs      R2      F Pr(>F)
## Graft         3   188.49 0.18719 3.5258  0.001 ***
## Graft_Health  1    16.52 0.01641 0.9272  0.511
## Residual     45   801.93 0.79640
## Total        49  1006.95 1.00000
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
## perform LDA
## Graft type
train.y = data.input_prop_filt$Graft
lda.results = lda_function(data.input_prop_filt,train.x,train.y)
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

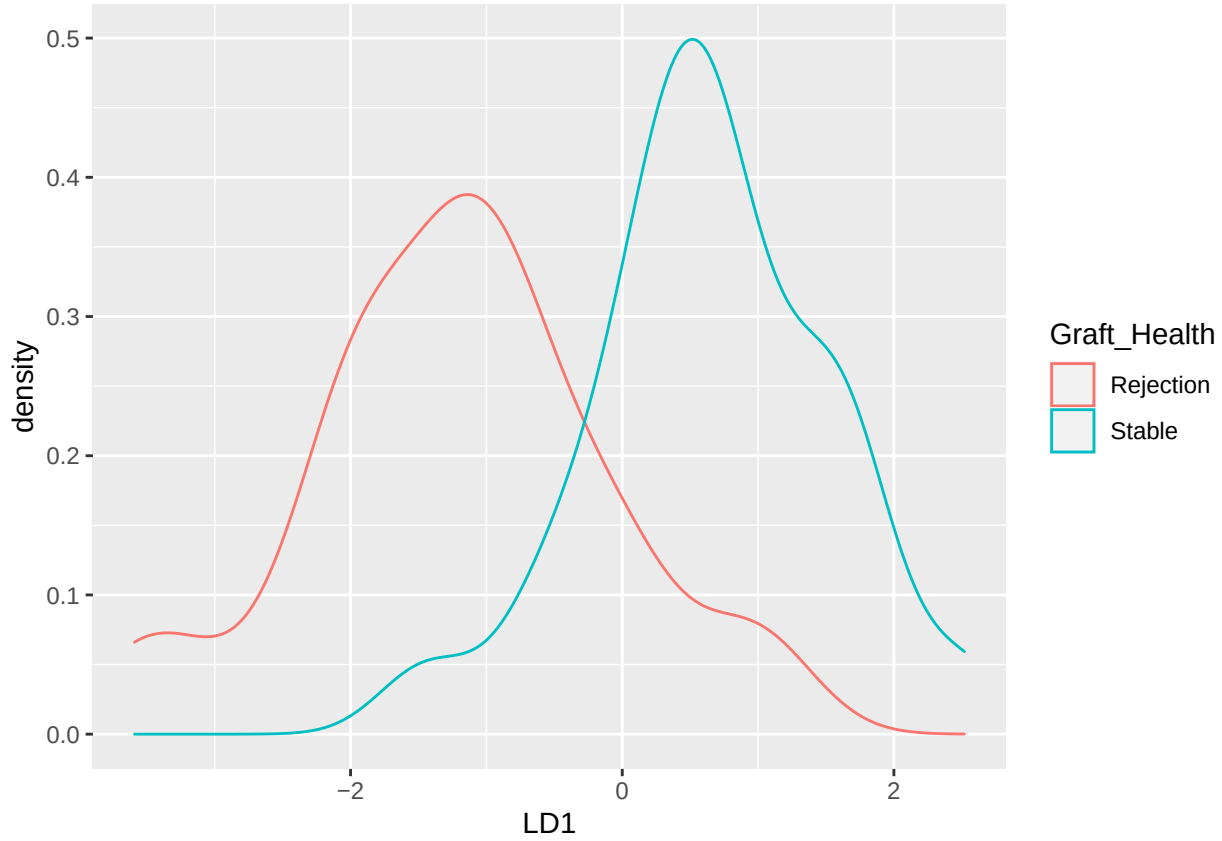
```
ggplot(lda.results$lda.df, aes(x = LD1, y = LD2, color = Graft)) + geom_point()
```




```
## Graft Health
train.y = data.input_prop_filt$Graft_Health
lda.results = lda_function(data.input_prop_filt,train.x,train.y)
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
ggplot(lda.results$lda.df, aes(x = LD1, color = Graft_Health)) + geom_density()
```




```
## perform glm
## without graft as a covariate, CD:45RA-25+5+38 not sigificant
glm.input = data.input_prop_filt %>%
     mutate(Graft_Health = factor(Graft_Health, levels = c('Rejection','Stable')) )
glm.res = glm(Graft_Health ~`CD:45RA-25+5+38-`, family = 'binomial', data = glm.input)
summary(glm.res)
```

```
##
## Call:
## glm(formula = Graft_Health ~ `CD:45RA-25+5+38-`, family = "binomial",
##     data = glm.input)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.601  -1.109  -0.955   1.244   1.399
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)        -0.62006    0.47263  -1.312    0.190
## `CD:45RA-25+5+38-`  0.08218    0.05828   1.410    0.159
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 69.235  on 49  degrees of freedom
## Residual deviance: 67.066  on 48  degrees of freedom
## AIC: 71.066
```

```
##
## Number of Fisher Scoring iterations: 4
```

```
## with graft as a covariate, CD:45RA-25+5+38  significant
glm.res = glm(Graft_Health ~ Graft + `CD:45RA-25+5+38-`, family = 'binomial', data = glm.input)
summary(glm.res)
```

```
##
## Call:
## glm(formula = Graft_Health ~ Graft + `CD:45RA-25+5+38-`, family = "binomial",
##     data = glm.input)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.5690  -1.0341  -0.6285   1.0654   1.7571
##
## Coefficients:
##                     Estimate Std. Error z value Pr(>|z|)
## (Intercept)         -1.56454    0.77766  -2.012   0.0442 *
## GraftIntestine      -1.48929    1.77908  -0.837   0.4025
## GraftKidney         -0.21425    0.86510  -0.248   0.8044
## GraftLiver           1.29431    0.74992   1.726   0.0844 .
## `CD:45RA-25+5+38-`   0.16205    0.08012   2.022   0.0431 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 69.235  on 49  degrees of freedom
## Residual deviance: 62.018  on 45  degrees of freedom
## AIC: 72.018
##
## Number of Fisher Scoring iterations: 4
```

```
confint(glm.res)
```

```
## Waiting for profiling to be done...
```

```
##                         2.5 %     97.5 %
## (Intercept)        -3.22632352 -0.1234874
## GraftIntestine     -5.32959681  2.2131922
## GraftKidney        -1.99215406  1.4744780
## GraftLiver         -0.12832013  2.8470415
## `CD:45RA-25+5+38-`  0.02044791  0.3439311
```

```
#####################################################################
## Unsupervised clsutering: Flowsom on subsetted example data ##
#####################################################################
## flowsom was done in 2 iterations: first to identify the  myeloid and lymphoid lineages, then to iden
## because flowsom is non-deterministic, we show example code below but do not actually label the linea
lineage_markers = c('CD3','CD19','CD4','CD8','CD56','CD14','CD16')
library(FlowSOM)
```

```
flowsom.input = df %>% dplyr::select(all_of(lineage_markers))  %>% as.matrix()
fSOM <- FlowSOM(flowsom.input ,
                # Input options:
                compensate = FALSE,
                transform = FALSE,
                scale = FALSE,
                nClus = 12,
                xdim = 10, ydim = 10,
)
fSOM_clusters <- GetMetaclusters(fSOM)
df$flowsom_general = fSOM_clusters

###  annotate your populations. We use the existing lineage annotations for this example code
myeloid_markers = c('CD56', 'CD16', 'CD14','CD11c')
lymphocyte_markers = c('CD19', 'CD20', 'CD27', 'CD38', 'CD4','CD8', 'CD25', 'CD5', 'CD45RA', 'FoxP3', '

## after your annotation, perform flowsom on 2 separate lineage subsets.
## we use the existing lineage annotations for this example code
df_flowsom_monocyte_NK = df %>% dplyr::filter(lineage %in% c('monocyte lineage','NK-cell')) ## just for
df_flowsom_TcellBcell = df %>% dplyr::filter(lineage %in% c('T-cell','B-cell')) ## just for example

df_flowsom_inputs = list(df_flowsom_monocyte_NK = df_flowsom_monocyte_NK, df_flowsom_TcellBcell = df_fl
fs.results = lapply(names(df_flowsom_inputs), function(fs){
     if(fs == 'df_flowsom_monocyte_NK') {
          channels = myeloid_markers
          print(channels)
          nclust = 15
     } else if (fs == 'df_flowsom_TcellBcell'){
          channels = lymphocyte_markers
          print(channels)
          nclust = 40
     }
     fs.df = df_flowsom_inputs[[fs]]
     fs.input= fs.df %>% ungroup()%>% dplyr::select(all_of(channels))  %>% as.matrix()
     fs.res <- FlowSOM(fs.input ,
                       # Input options:
                       compensate = FALSE,
                       transform = FALSE,
                       scale = FALSE,
                       nClus = nclust,
                       xdim = 10, ydim = 10,
     )
     fs.res_clusters <- GetMetaclusters(fs.res)
     fs_flowsom = fs.df %>%
          mutate(flowsom = fs.res_clusters)
     return(fs_flowsom)
})
```

```
## [1] "CD56"  "CD16"  "CD14"  "CD11c"
```

```
##  [1] "CD19"   "CD20"   "CD27"   "CD38"   "CD4"    "CD8"    "CD25"   "CD5"
##  [9] "CD45RA" "FoxP3"  "CCR7"   "TCRgd"
```

```
names(fs.results) =c('df_flowsom_monocyte_NK','df_flowsom_TcellBcell')
```

```
################################
## variance partition analysis ##
################################
# library(variancePartition)
# library(BiocParallel)
## in the manuscript, we provided all celltypes regardless of how terminally defined they were.
## in this example, we provide only the terminally defined populations as template code for the approac
varPart_data = data.input_prop_filt

form_fixed <- ~  Graft_Health + Graft ## model as fixed effects
form_random <- ~  (1|Graft_Health) + (1|Graft) ## model as random effects if categorical
formula_options = list(fixed = form_fixed, random = form_random)
varpart_res =lapply(formula_options, function(form){
     varPart <- fitExtractVarPartModel(varPart_data %>% ungroup() %>% dplyr::select(-Graft,-Graft_Healt
                                       formula = form,
                                       varPart_data %>% ungroup() %>% dplyr::select(Graft,Graft_Health)
                                       quiet = FALSE)
     varPart_input = varPart_data %>% ungroup() %>% dplyr::select(-Graft,-Graft_Health) %>% t()
     info = varPart_data %>%ungroup%>% dplyr::select(Graft,Graft_Health)

     vp.res = data.frame(varPart, check.names = TRUE) %>% mutate(feature = rownames(.))
     vp.res.gath = vp.res %>% gather(key = "meta", value = "variance_explained", -feature) %>%
          mutate(meta = factor(meta, levels = c("Graft", "Graft_Health","Residuals")))
     return(list(vp.res=vp.res, vp.res.gath=vp.res.gath))
})
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
##
## Total:0.1 s
```

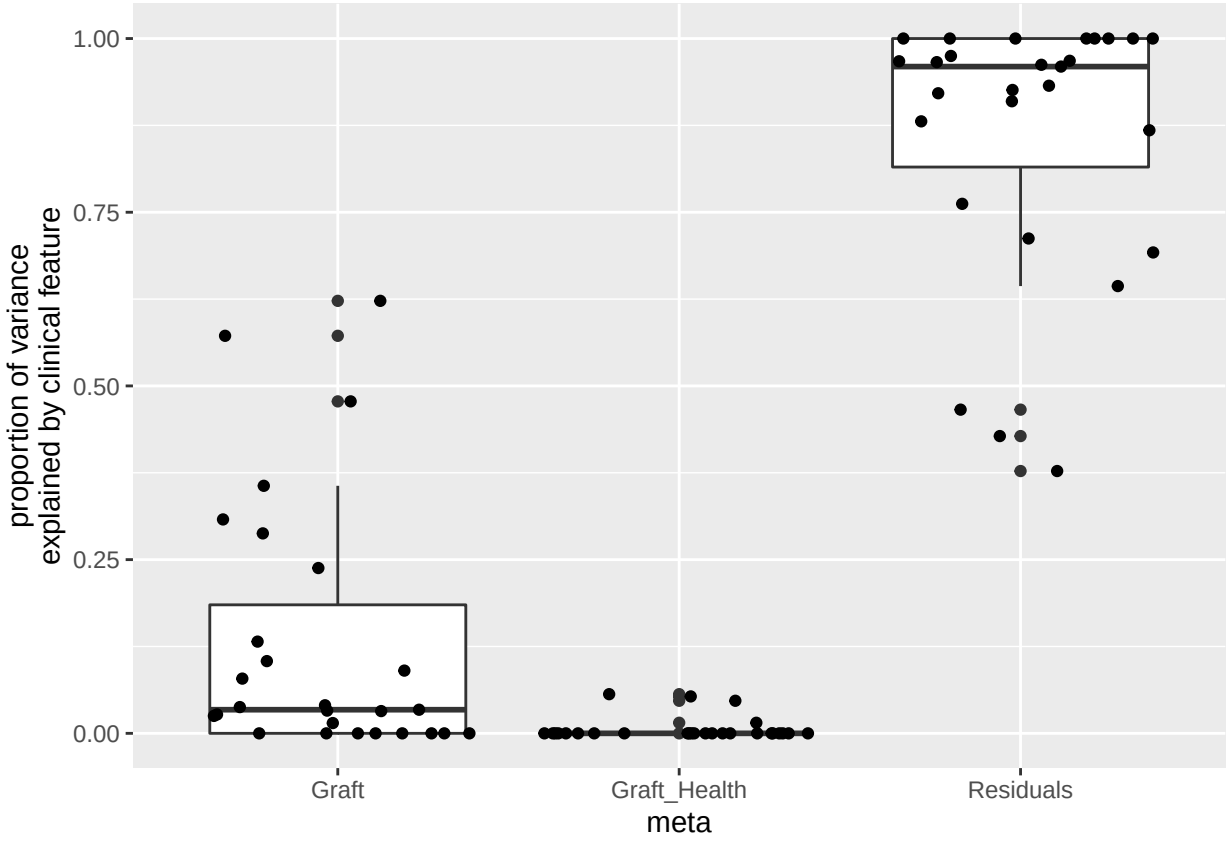
```
## Dividing work into 1 chunks...
```

```
##
## Total:0.5 s
```

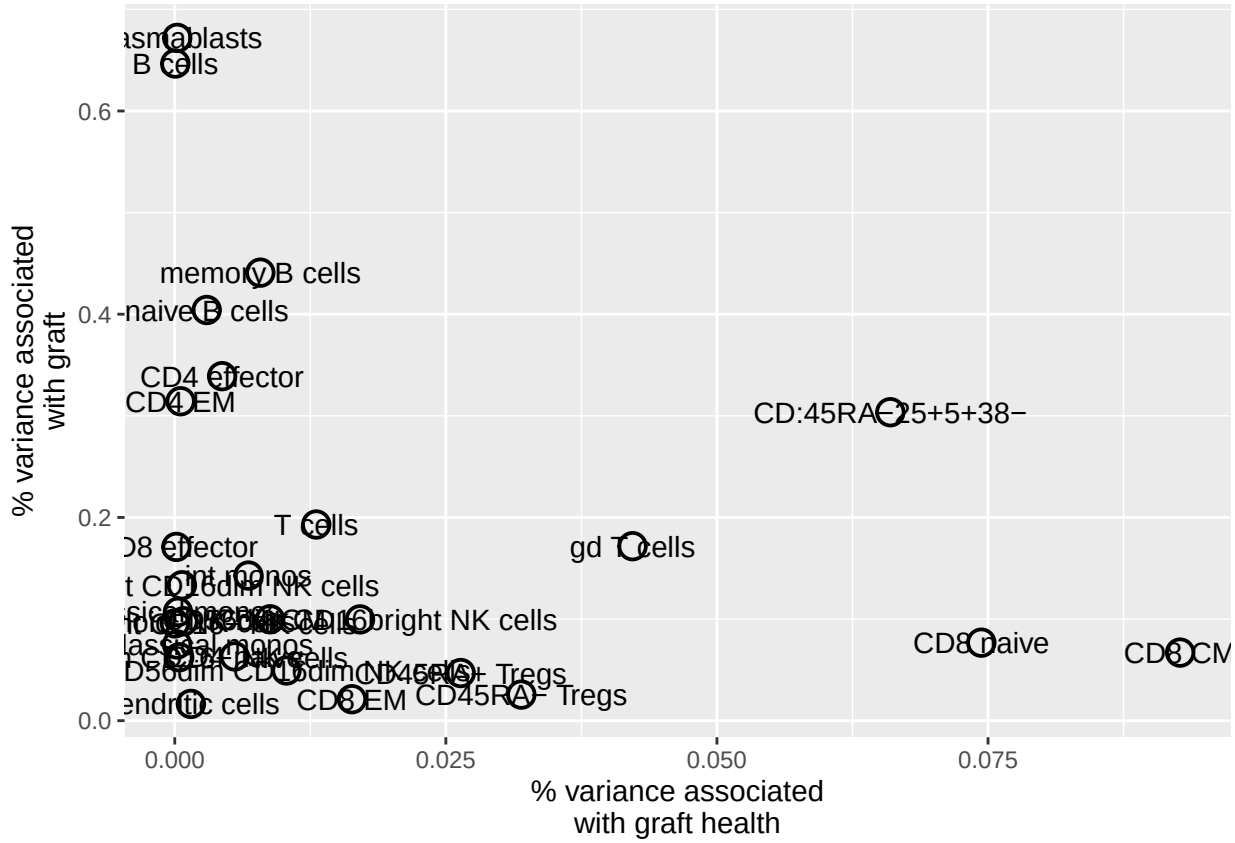
```
# ## example, fixed
# ggplot(varpart_res$fixed$vp.res.gath,
#                             aes(x = meta, y = variance_explained)) +
#      geom_boxplot() +
#      # xlab("") +
#      ylab("proportion of variance\nexplained by clinical feature") +
```

```
#   geom_jitter(size = 1.5)
#
# ggplot(varpart_res$fixed$vp.res  , aes(x = Graft_Health, Graft)) +
#   # theme(panel.background = element_blank()) +
#   geom_point( size = 4, shape = 21, stroke = 1, color = 'black') +
#   xlab('% variance associated\nwith graft health') +
#   ylab('% variance associated\nwith graft') +
#   geom_text(aes(label = feature))
```

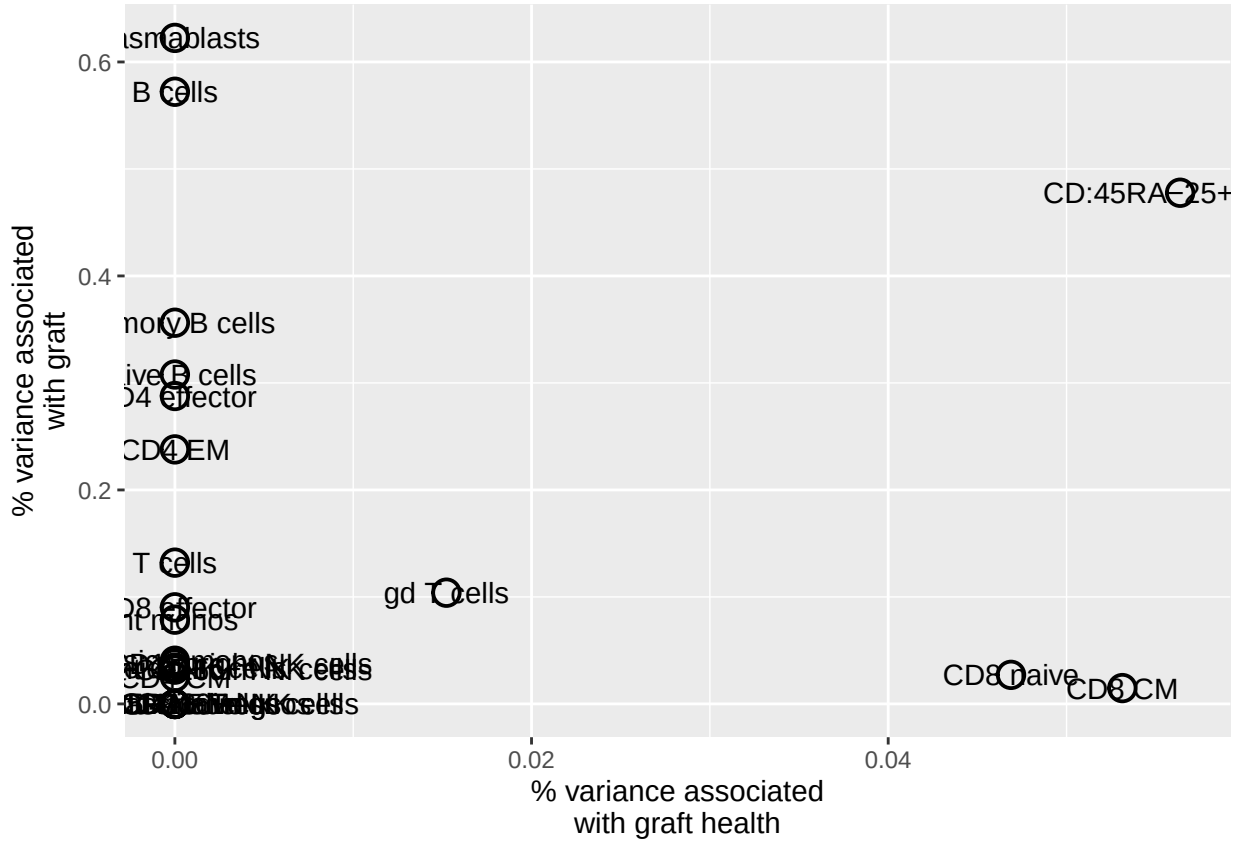
```
## example, random
ggplot(varpart_res$random$vp.res.gath,
       aes(x = meta, y = variance_explained)) +
     geom_boxplot() +
     # xlab("") +
     ylab("proportion of variance\nexplained by clinical feature") +
     geom_jitter(size = 1.5)
```




```
ggplot(varpart_res$fixed$vp.res  , aes(x = Graft_Health, Graft)) +
     # theme(panel.background = element_blank()) +
     geom_point( size = 4, shape = 21, stroke = 1, color = 'black') +
     xlab('% variance associated\nwith graft health') +
     ylab('% variance associated\nwith graft') +
     geom_text(aes(label = feature))
```




```
ggplot(varpart_res$random$vp.res  , aes(x = Graft_Health, Graft)) +
     # theme(panel.background = element_blank()) +
     geom_point( size = 4, shape = 21, stroke = 1, color = 'black') +
     xlab('% variance associated\nwith graft health') +
     ylab('% variance associated\nwith graft') +
     geom_text(aes(label = feature))
```




```
##############################
## cosine distance analysis ## ## using subsetted, example dataset
############################## EXAMPLE CODE ONLY
# library(lsa)
# ## begin code, sample down to 250 cells so the code runs quicker
## not an accurate representation of the celltype proportions, sampling is done within lineages
features = c('CCR4','CCR6','CCR7','CD11b','CD11c','CD25','CD27','CD28','CD38','CD45RA','CD45RO','CD5','
CD4_cells =  df %>%
     dplyr::filter(celltype %in% c('CD4 CM','CD4 effector','CD45RA+ Tregs','CD8 CM','CD45RA- Tregs','CD
     mutate(sample = as.character(sample)) %>%
     dplyr::select(all_of(features ), celltype, sample,cell.id)%>% ungroup()

CD4_cells_sampleSplits = split(CD4_cells, CD4_cells$sample)


message('calculating cosine similarity...')
```

```
## calculating cosine similarity...
```

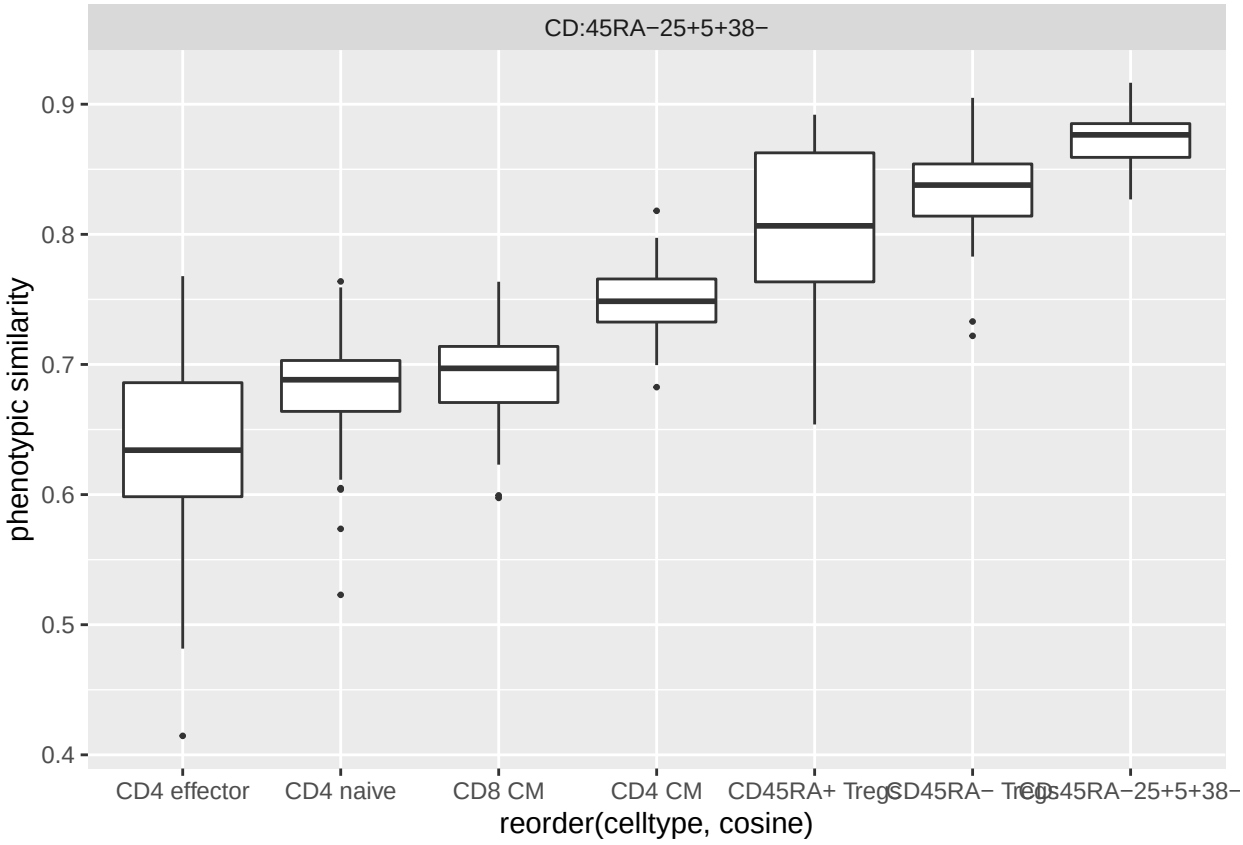
```
CosineSimilarityOfEachSample = calculateCosineSimilarityOfEachSample(CD4_cells_sampleSplits)
```

```
## sample #: 10

## sample #: 11
```

```
## sample #: 12

## sample #: 13

## sample #: 14

## sample #: 15

## sample #: 16

## sample #: 17

## sample #: 2

## sample #: 20

## sample #: 21

## sample #: 22

## sample #: 23

## sample #: 24

## sample #: 25

## sample #: 26

## sample #: 27

## sample #: 29

## sample #: 3

## sample #: 30

## sample #: 31

## sample #: 32

## sample #: 33

## sample #: 34

## sample #: 35

## sample #: 36
```

```
## sample #: 37

## sample #: 4

## sample #: 40

## sample #: 41

## sample #: 42

## sample #: 43

## sample #: 44

## sample #: 45

## sample #: 46

## sample #: 47

## sample #: 48

## sample #: 49

## sample #: 5

## sample #: 50

## sample #: 51

## sample #: 52

## sample #: 53

## sample #: 54

## sample #: 55

## sample #: 56

## sample #: 57

## sample #: 6

## sample #: 7

## sample #: 8
```

```
cosine_results =CosineSimilarityOfEachSample %>% bind_rows()

## plot results
ggplot(cosine_results %>% dplyr::filter(celltype2 == 'CD:45RA-25+5+38-'),
       aes(x = reorder(celltype, cosine), y = cosine)) +
     geom_boxplot(outlier.size = 0.5) +
     # ggpubr::stat_compare_means(comparisons = ) +
     ylab('phenotypic similarity') +
     # theme(axis.text.x = element_text(angle = 90) +
     facet_wrap(~celltype2, scales = 'free_y')
```




```
############################
## example code for umap ##
############################
## using sampled data for fast results
umap.results = umap_function(CD4_cells, features = features)
```

```
## 22:31:27 UMAP embedding parameters a = 1.896 b = 0.8006

## 22:31:27 Read 20342 rows and found 20 numeric columns

## 22:31:27 Using Annoy for neighbor search, n_neighbors = 15

## 22:31:27 Building Annoy index with metric = euclidean, n_trees = 50
```

```
## 0%   10   20   30   40   50   60   70   80   90   100%

## [----|----|----|----|----|----|----|----|----|----|

## **************************************************|
## 22:31:29 Writing NN index file to temp file /var/folders/49/j9v_t5kd1yd04myymzh7zkgm0000gn/T//Rtmpgv
## 22:31:29 Searching Annoy index using 8 threads, search_k = 1500
## 22:31:29 Annoy recall = 100%
## 22:31:30 Commencing smooth kNN distance calibration using 8 threads
## 22:31:30 Initializing from normalized Laplacian + noise
## 22:31:31 Commencing optimization for 200 epochs, with 436890 positive edges
## 22:31:40 Optimization finished
```

```
ggplot(umap.results, aes(x = UMAP1, y  =UMAP2, color = celltype)) +
     geom_point(size = 1)
```