

# A framework for pipeline benchmarking and its application to scRNAseq clustering

Pierre-Luc Germain<sup>1,2</sup>, Anthony Sonrel<sup>1</sup>, Mark D. Robinson<sup>1</sup>

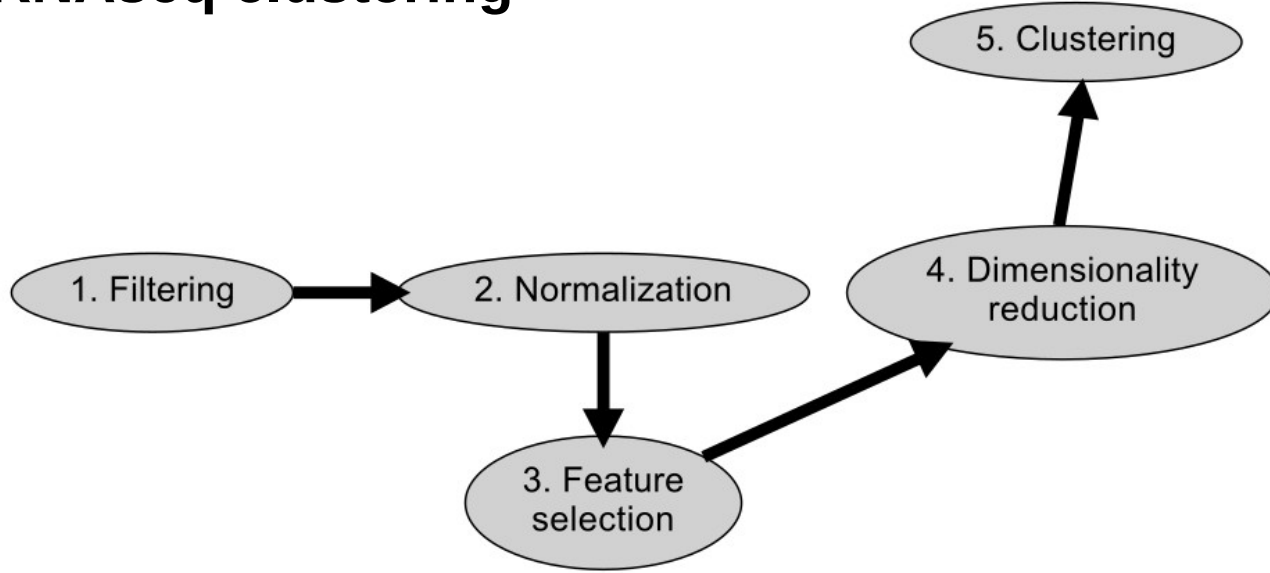
1. Lab of Statistical Bioinformatics (Robinson group), University of Zürich
2. D-HEST Institute for Neuroscience, ETH Zürich

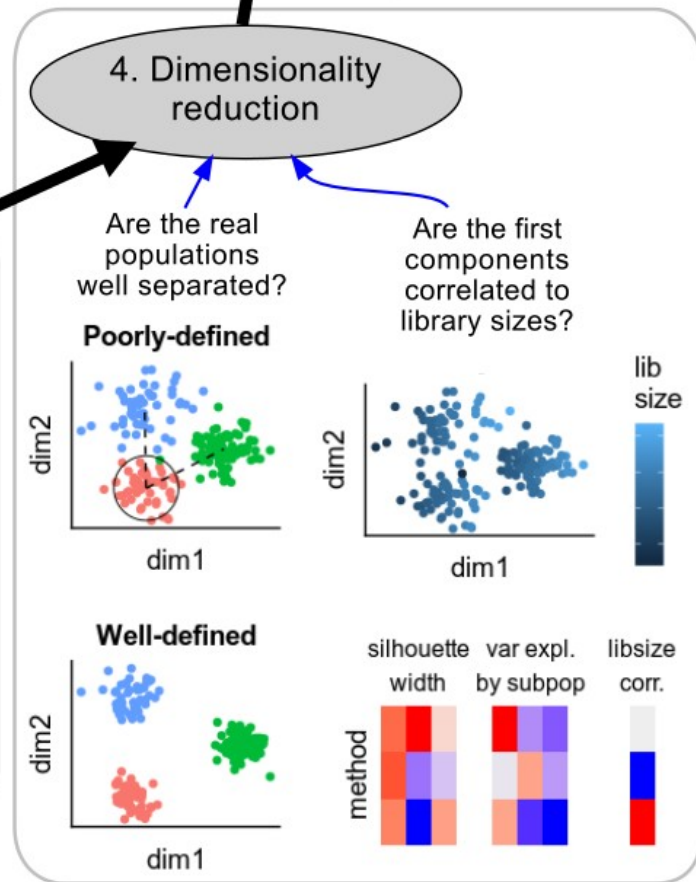
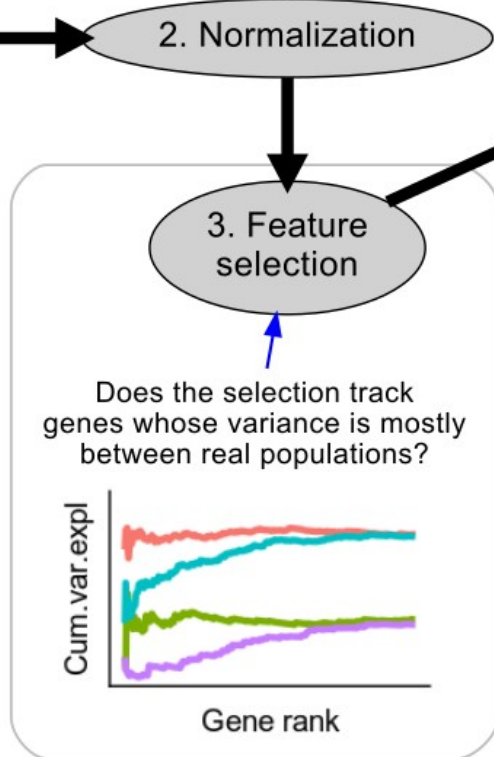
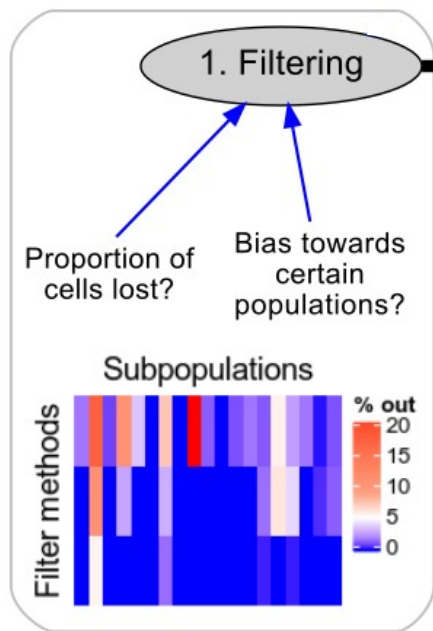
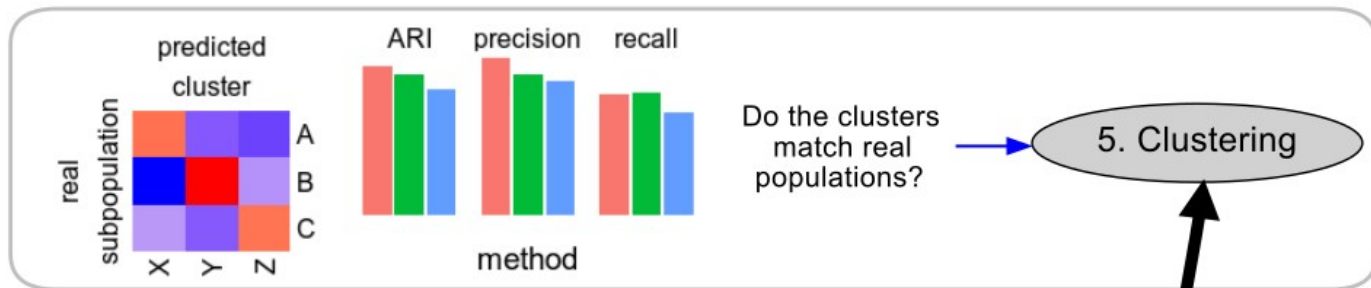


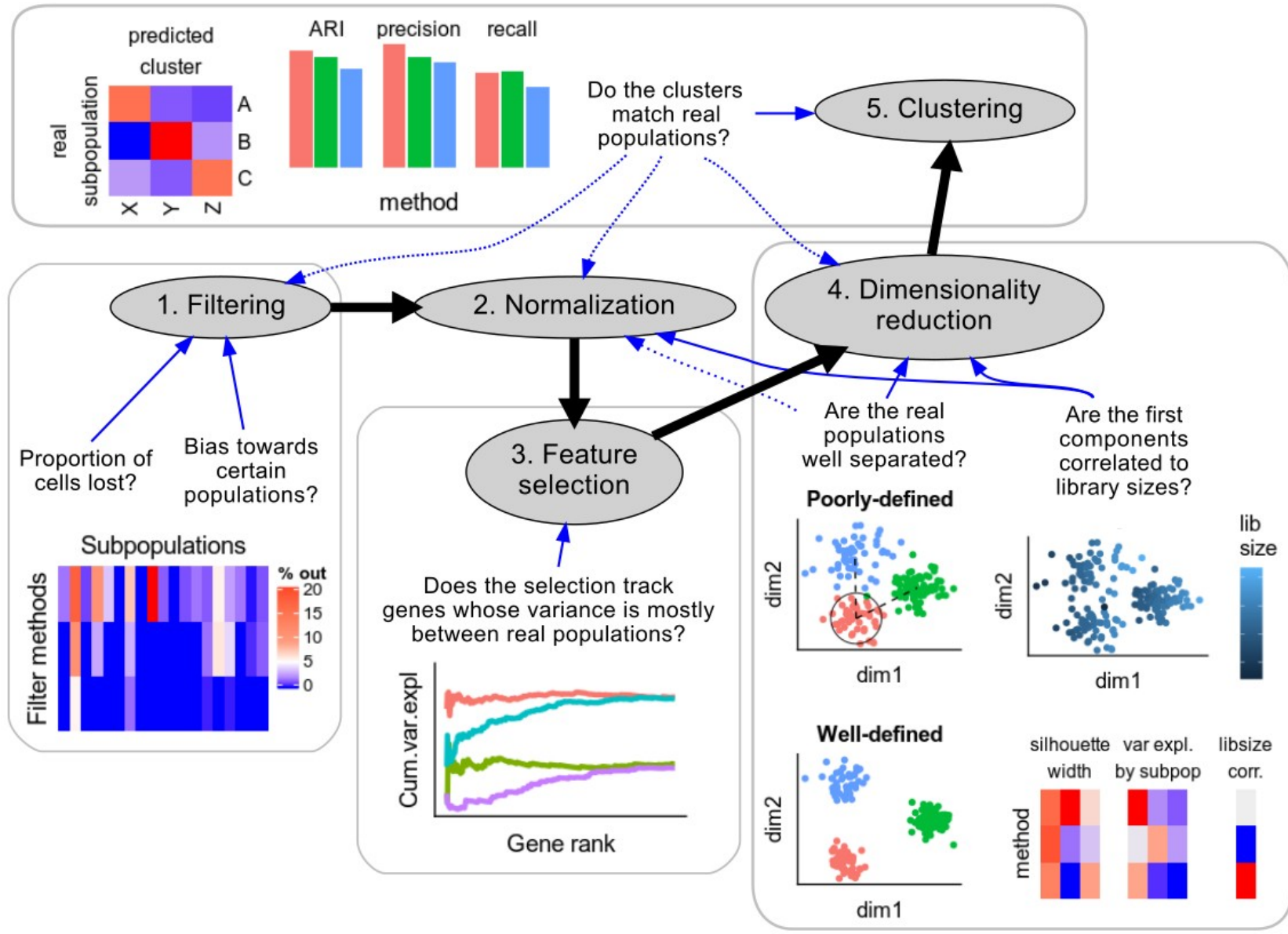
University of  
Zurich<sup>UZH</sup>

**ETH** zürich

# A simple scRNAseq clustering pipeline







**We define a pipeline as, minimally, a list of functions to be successively applied on the output of the previous one**

```
pipeline <- PipelineDefinition(  
  list(  
    step1=function(x, s1_method, param1){  
      ## processing, e.g.:  
      x <- some_function(x, s1_method, param1)  
      return(x)  
    },  
    step2=function(x, s2_method, param2, param3){  
      ## processing, e.g.:  
      get(s2_method)(x, param2, param3)  
    },  
    ...  
    stepN=function(x, param4){  
      ## processing  
    }  
  )  
)
```



**We define a pipeline as, minimally, a list of functions to be successively applied on the output of the previous one**

```
pipeline <- PipelineDefinition(  
  list(  
    step1=function(x, s1_method, param1){  
      ## processing, e.g.:  
      x <- some_function(x, s1_method, param1)  
      return(x)  
    },  
    step2=function(x, s2_method, param2, param3){  
      ## processing, e.g.:  
      get(s2_method)(x, param2, param3)  
    },  
    ...  
    stepN=function(x, param4){  
      ## processing  
    }  
  )  
)
```

Optionally, the *PipelineDefinition* can include *evaluation* functions for some steps

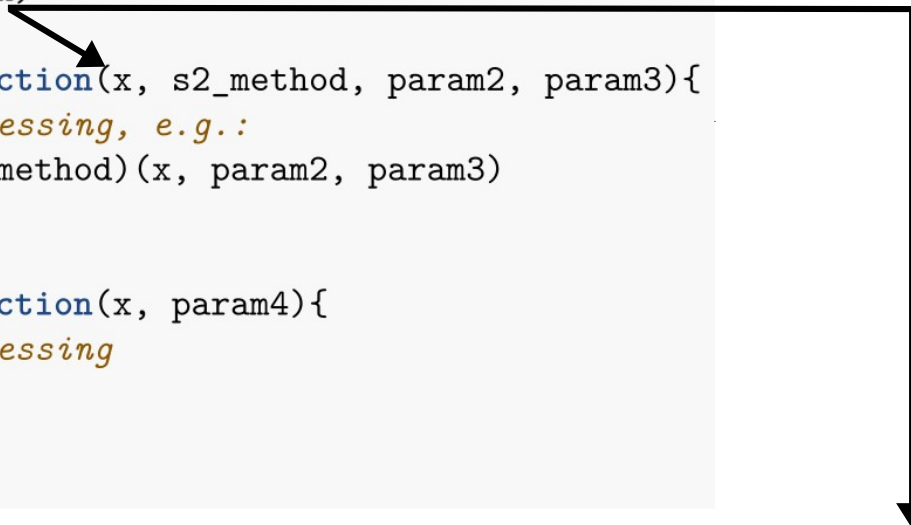
```
stepFn(pipeline, step="step1", type="evaluation") <- function(x){  
  # produce some evaluation metric based on 'x'  
}
```

We define a pipeline as, minimally, a list of functions to be successively applied on the output of the previous one

```
pipeline <- PipelineDefinition(  
  list(  
    step1=function(x, s1_method, param1){  
      ## processing, e.g.:  
      x <- some_function(x, s1_method, param1)  
      return(x)  
    },  
    step2=function(x, s2_method, param2, param3){  
      ## processing, e.g.:  
      get(s2_method)(x, param2, param3)  
    },  
    ...  
    stepN=function(x, param4){  
      ## processing  
    }  
  )  
)
```

Optionally, the *PipelineDefinition* can include *evaluation* functions for some steps

```
stepFn(pipeline, step="step1", type="evaluation") <- function(x){  
  # produce some evaluation metric based on 'x'  
}
```



dataset	s1_method	param1	metric1
dataset 1	function_A	5	...
dataset 1	function_B	5	...



**We define a pipeline as, minimally, a list of functions to be successively applied on the output of the previous one**

```
pipeline <- PipelineDefinition(  
  list(  
    step1=function(x, s1_method, param1){  
      ## processing, e.g.:  
      x <- some_function(x, s1_method, param1)  
      return(x)  
    },  
    step2=function(x, s2_method, param2, param3){  
      ## processing, e.g.:  
      get(s2_method)(x, param2, param3)  
    },  
    ...  
    stepN=function(x, param4){  
      ## processing  
    }  
  )  
)
```

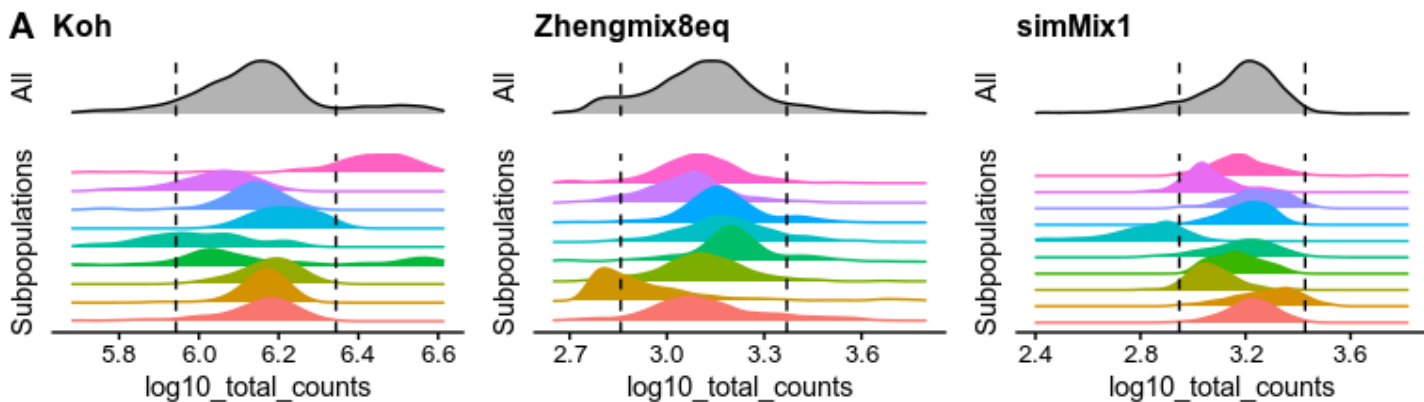
```
alternatives = list(  
  s1_method=c("function_A", "function=B"),  
  param1=5,  
  s2_method=c("function_C", "function_D"),  
  param2=FALSE,  
  param3=50,  
  param4=FALSE  
)  
runPipeline(datasets, alternatives, pipDef=pipeline)
```



## Basic Seurat-based *PipelineDefinition*:

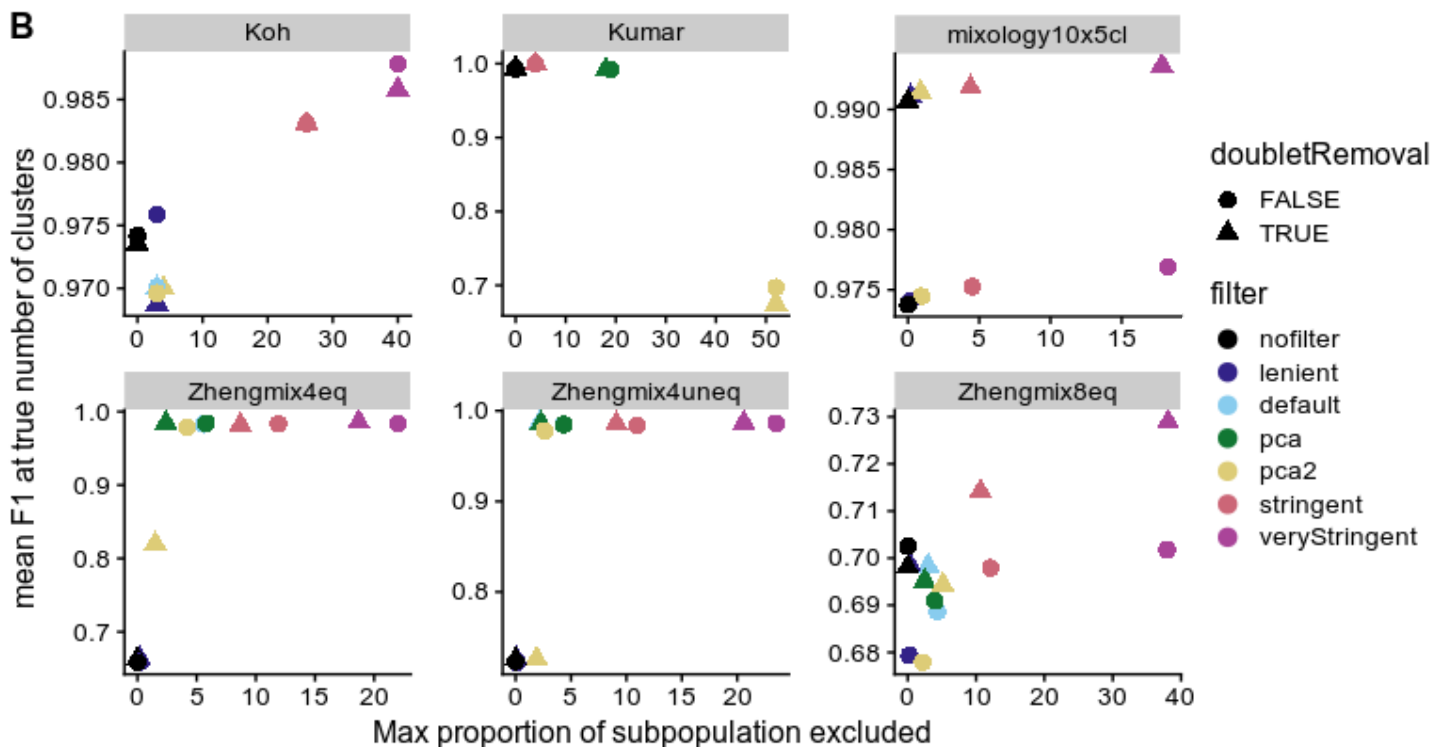
```
> pipeline <- scrna_seurat_pipeline()
> pipeline
A PipelineDefinition object with the following steps:
- doublet(x, doubletmethod) *
Takes a SCE object with the `phenoid` colData column, passes it through the function `doubletmethod`, and outputs a filtered SCE.
- filtering(x, filt) *
Takes a SCE object, passes it through the function `filt`, and outputs a filtered Seurat object.
- normalization(x, norm) *
Passes the object through function `norm` to return the object with the normalized and scale data slots filled.
- selection(x, sel, selnb)
Returns a seurat object with the VariableFeatures filled with `selnb` features using the function `sel`.
- dimreduction(x, dr, maxdim) *
Returns a seurat object with the PCA reduction with up to `maxdim` components using the `dr` function.
- clustering(x, clustmethod, dims, k, steps, resolution, min.size) *
Uses function `clustmethod` to return a named vector of cell clusters.
```

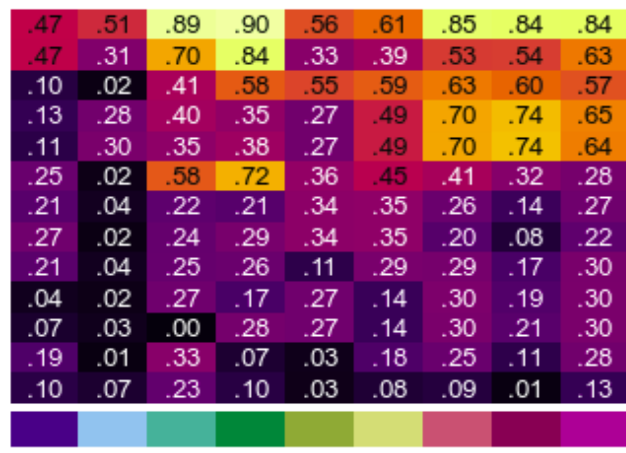
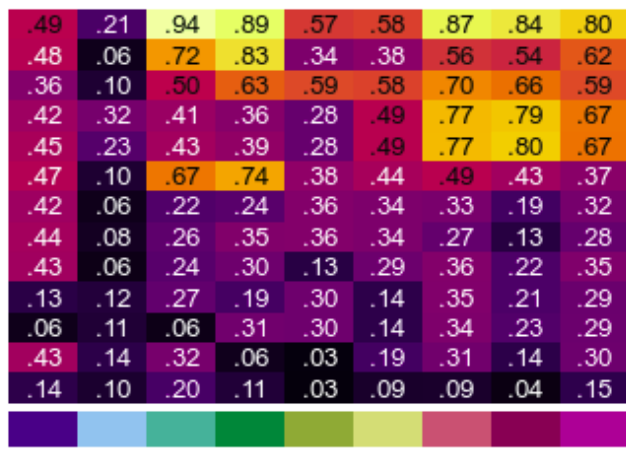
and its application to 9 datasets with true cell labels...



## Filtering:

Tradeoff between classification accuracy and the proportion/bias of the excluded cells

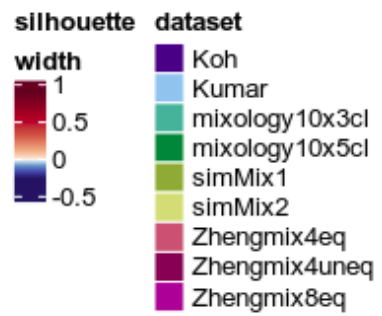
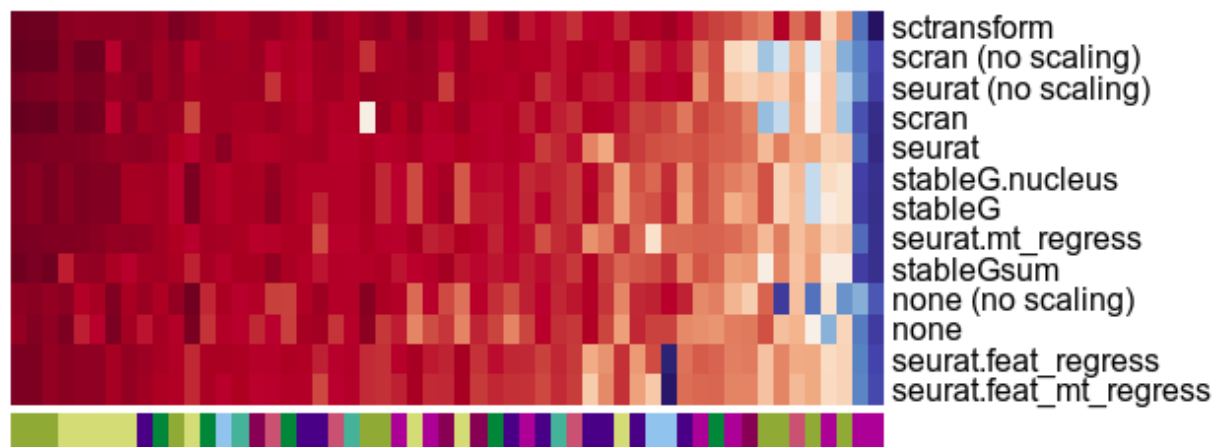


**A**residual corr with  
log10\_total\_countsresidual corr with  
log10\_total\_features

none (no scaling)  
 none  
 seurat (no scaling)  
 seurat.feats\_mt\_regress  
 seurat.feats\_regress  
 scran (no scaling)  
 stableG  
 stableG.nucleus  
 stableGsum  
 seurat  
 seurat.mt\_regress  
 scran  
 sctransform

# Normalization & scaling:

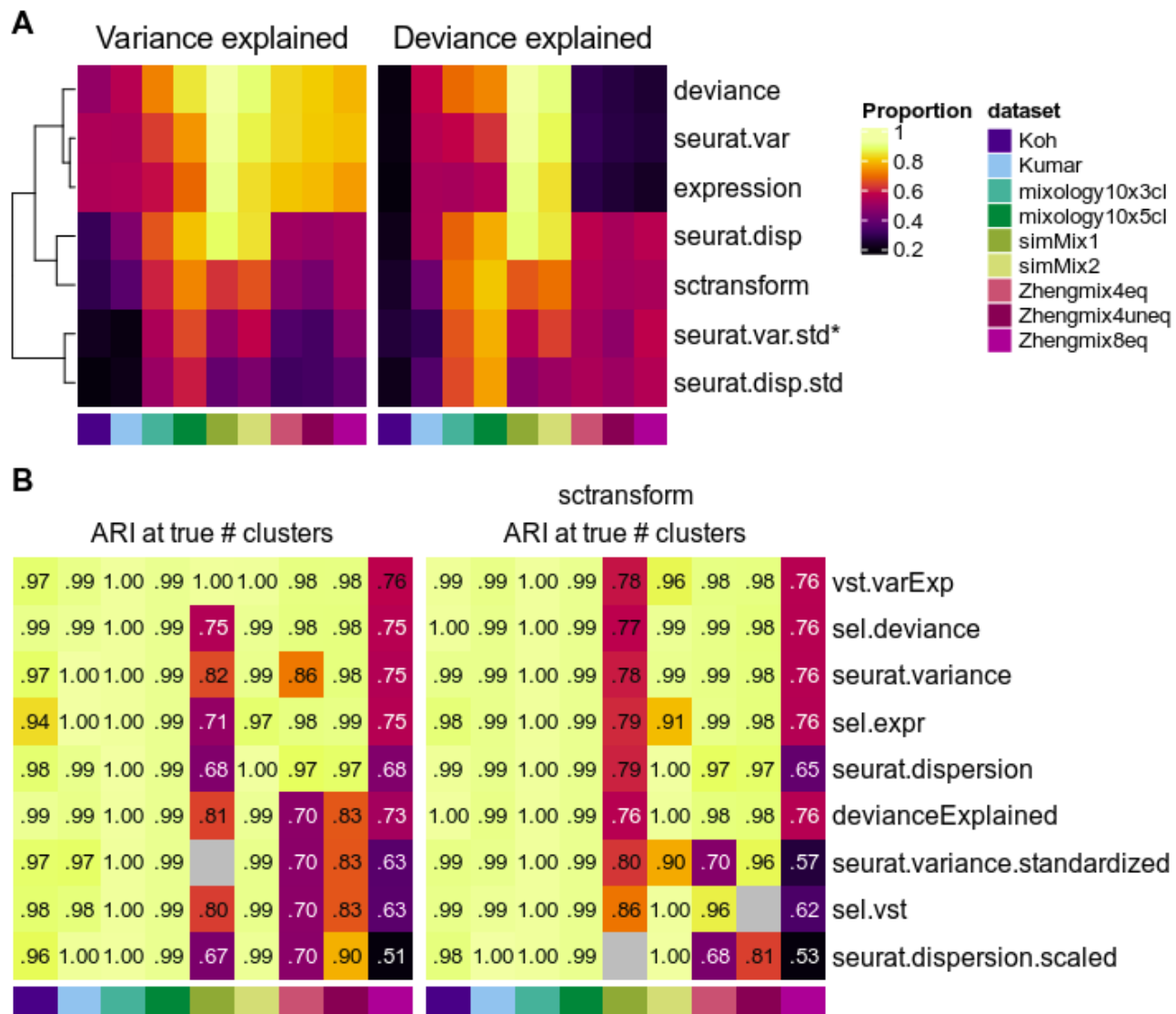
Residual correlation with technical covariates, separability of the subpopulations (silhouette)

**B**average silhouette width  
per subpopulation

sctransform wins...

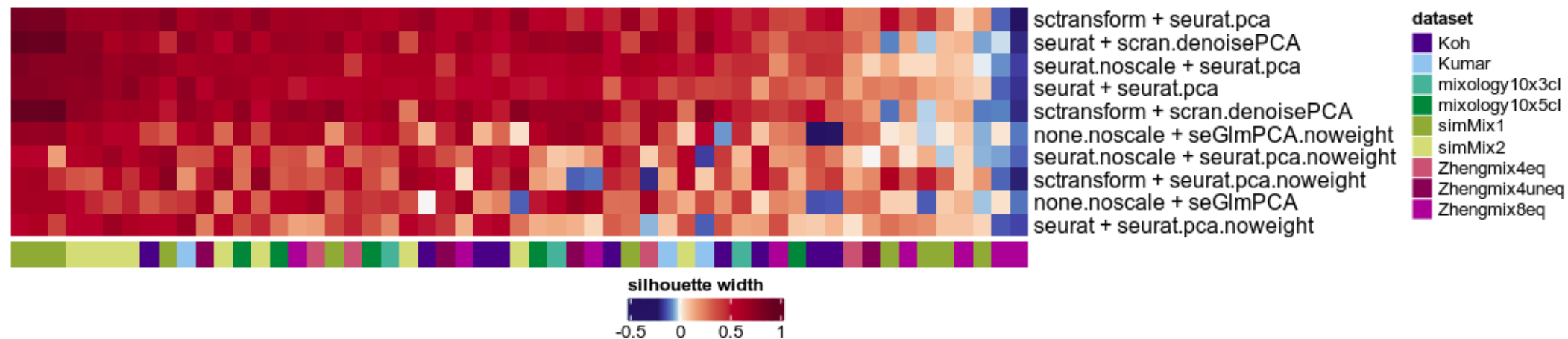
## Feature selection:

Does the ranking / selection track genes whose variance is between subpopulations?

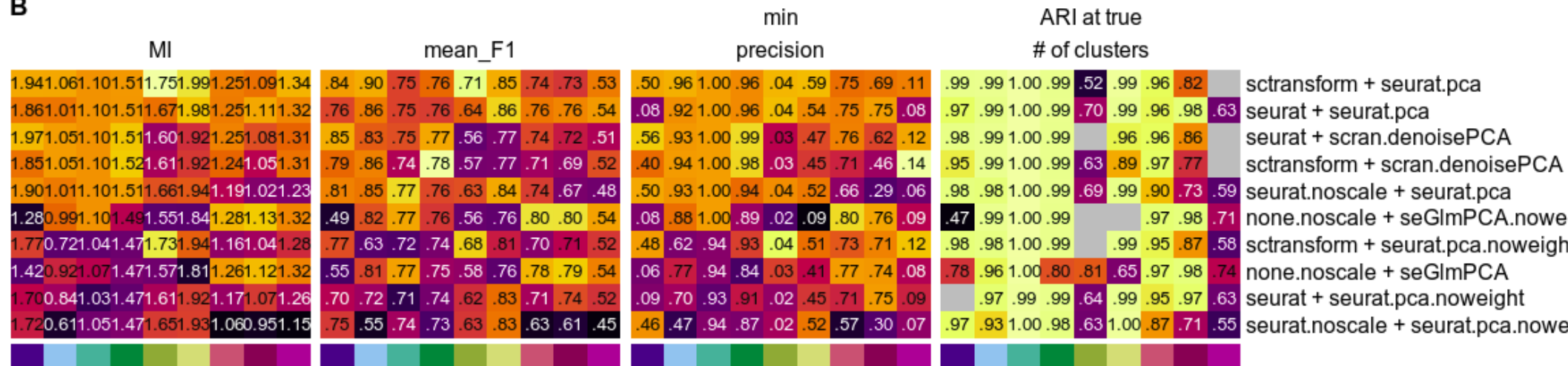


A

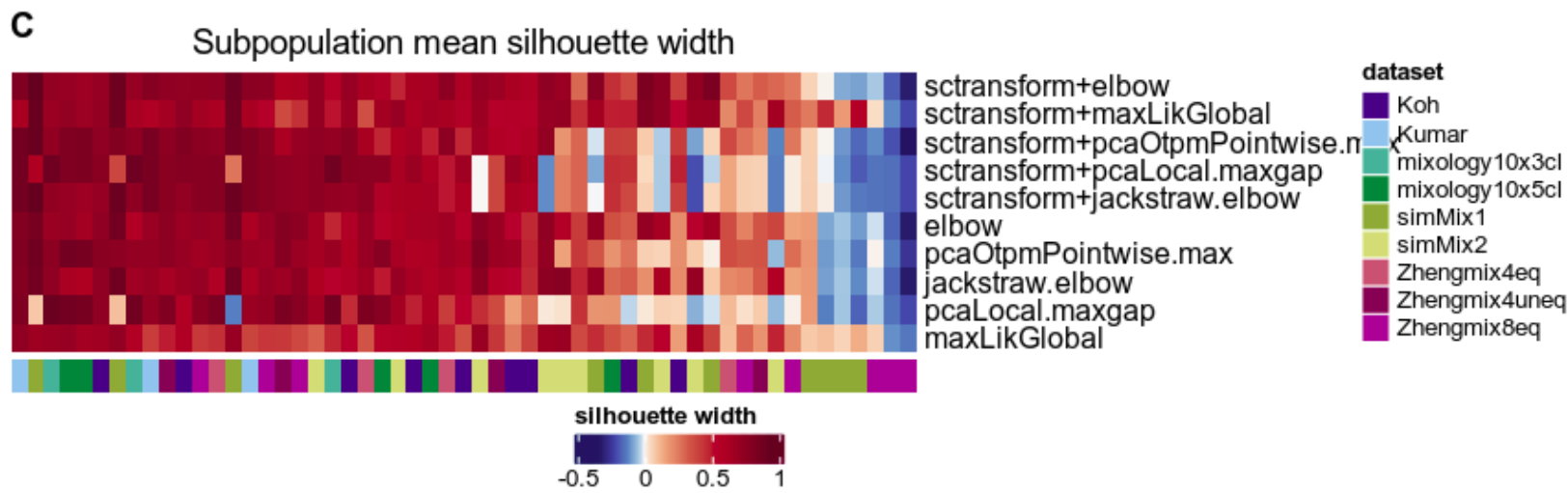
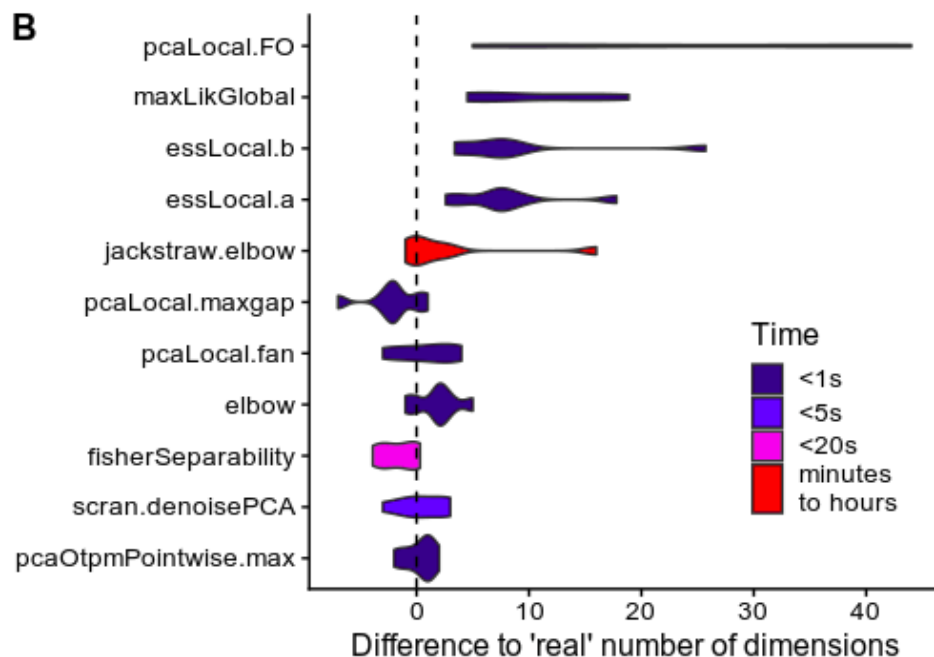
## Subpopulation mean silhouette width



B



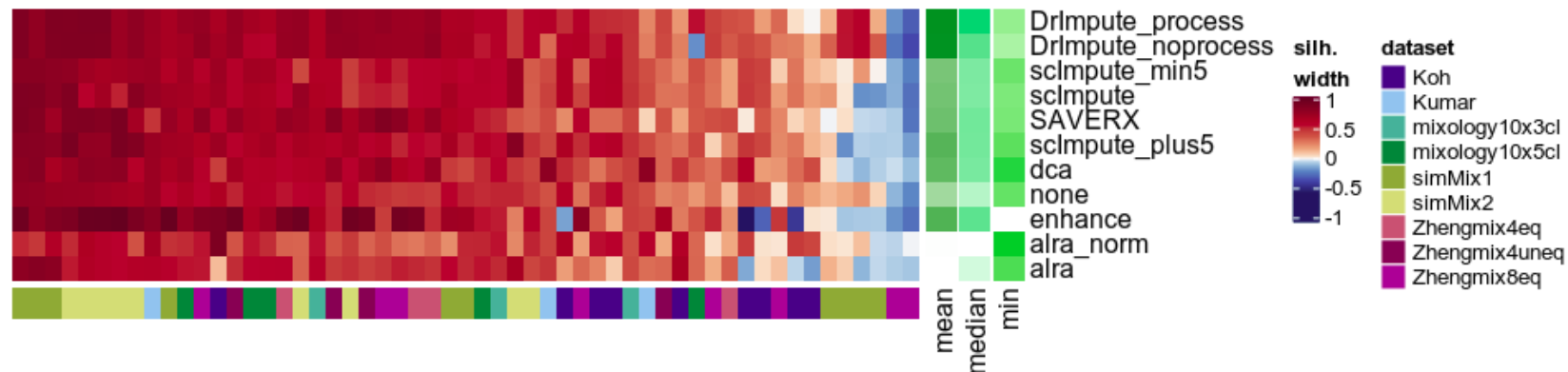
# Estimating the number of dimensions



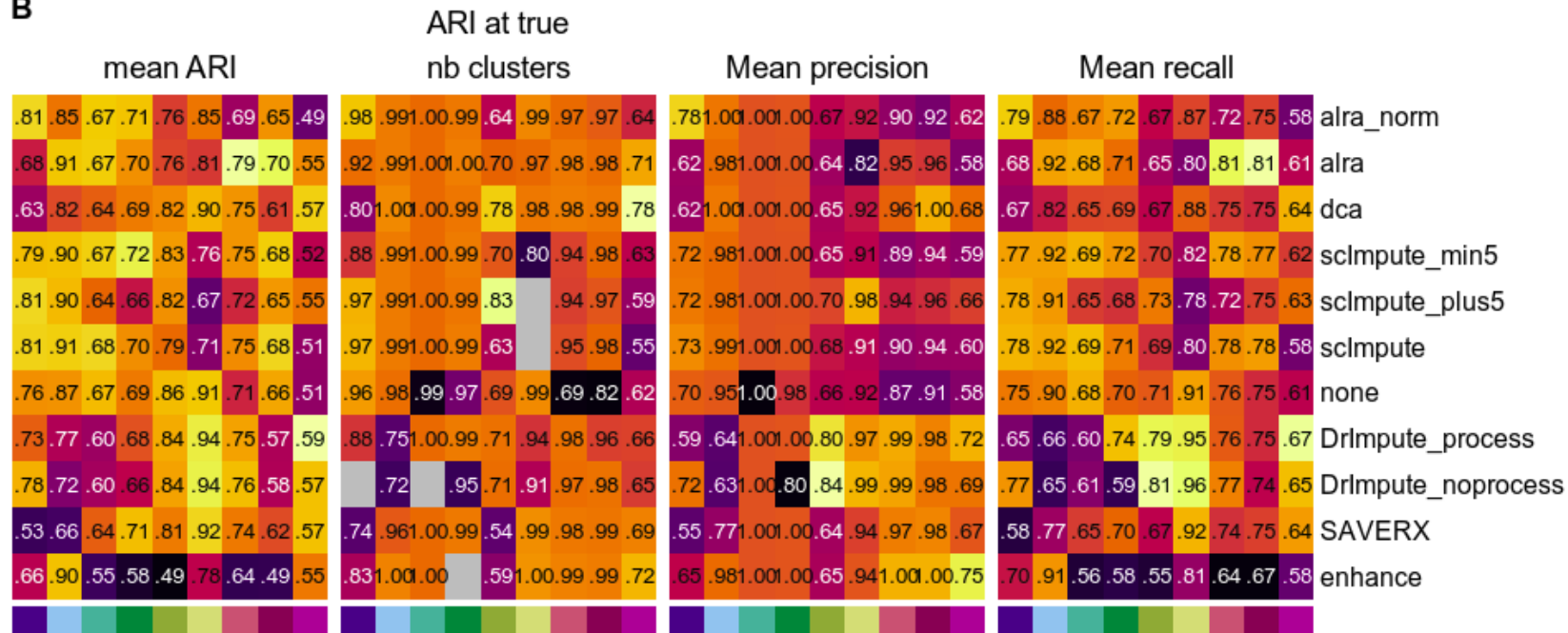


A

Mean silhouette width

Denoising/  
Imputation

B

(with  
Anthony  
Sonrel)





Meet me at the  
(not-quite-up-to-date)  
**poster** for  
feedback/discussion!